WUP! THERE IT IS

Privacy and Security Issues in QQ Browser

By Jeffrey Knockel, Adam Senft, and Ron Deibert

MARCH 28, 2016
RESEARCH REPORT #72

munk school
OF GLOBAL AFFAIRS & PUBLIC POLICY

UNIVERSITY OF TORONTO

THECITIZENLAB

Copyright

Suggested Citation

Jeffrey Knockel, Adam Senft, and Ron Deibert. "WUP! There It Is: Privacy and Security Issues in QQ Browser," Citizen Lab Research Report No. 72, University of Toronto, March 2016.

Acknowledgements

The authors would like to thank Sarah McKune and Masashi Crete-Nishihata for assistance and peer review on this report. Jeffrey Knockel's research for this project was supported by the Open Technology Fund's Information Control Fellowship Program and Adam Senft's research from the John D. and Catherine T. MacArthur Foundation (Ronald J. Deibert, Principal Investigator).

About the Citizen Lab, Munk School of Global Affairs and Public Policy, University of Toronto

The Citizen Lab is an interdisciplinary laboratory based at the Munk School of Global Affairs and Public Policy, University of Toronto, focusing on research, development, and high-level strategic policy and legal engagement at the intersection of information and communication technologies, human rights, and global security.

We use a "mixed methods" approach to research that combines methods from political science, law, computer science, and area studies. Our research includes investigating digital espionage against civil society, documenting Internet filtering and other technologies and practices that impact freedom of expression online, analyzing privacy, security, and information controls of popular applications, and examining transparency and accountability mechanisms relevant to the relationship between corporations and state agencies regarding personal data and other surveillance activities.

Contents

QQ浏览器存在的隐私与安全隐患

Key findings

- **Both Windows (v9.2.5478) and Android (v6.3.0.1920) versions of web browser QQ Browser transmit personal user data to QQ servers without encryption or with easily decryptable encryption, and are vulnerable to arbitrary code execution during software updates.**
- **The Android version of QQ Browser transmits personally identifiable data, including a user's IMEI, IMSI, nearby WiFi access points, search queries entered into the address bar, URLs of pages visited, and Android ID, without encryption or with easily decryptable encryption.**
- **The Windows version of QQ Browser transmits personally identifiable data, including the URLs of visited websites, hard drive serial number, MAC address, and machine hostname, without encryption or with easily decryptable encryption.**
- **The software updating processes of both the Android and Windows version of QQ Browser have vulnerabilities that leave them susceptible to an attacker executing arbitrary code.**
- **Please see the "Update: Analysis of updated versions of QQ Browser" section at the end of this report for our analysis of the latest versions (Windows version 9.3.6872 and Android version 6.4.2) released prior to publication, following our disclosure to the vender.**

Introduction

QQ Browser (QQ浏览器) is a free web browser for the Android, Windows, Mac, and iOS platforms, developed by Chinese Internet giant Tencent. The application offers a number of features beyond those offered by built-in browsers, such as tabbed windows and integration with other chat platforms.

This report provides a detailed analysis of how the Windows and Android versions of QQ Browser transmit user data during their operation. This analysis reveals that both versions of QQ Browser transmit a number of personally identifiable user data points either with no encryption or with easily decryptable encryption. We use the phrase "easily decryptable encryption" to refer to the improper implementation of encryption algorithms. For a full discussion, see the "Easily decryptable' encryption" textbox in our report Baidu's and Don'ts: Privacy and Security Issues in Baidu Browser.

This insecure data transmission means that any in-path actor (such as a user's ISP, a coffee shop WiFi network, or a malicious actor with network visibility across

any of these type of access points) would be able to acquire this personal data by collecting traffic and performing any necessary decryption.

In addition to this insecure data transmission, both tested versions of the application perform software updates in a manner that is vulnerable to execution of arbitrary code by an attacker. This means that a malicious actor would be able to spoof a software update in order to install malicious code on a user's device.

This report is a continuation of Citizen Lab research on the privacy and security of mobile applications in Asia. Our previous work includes reports that identified similar concerns with mobile browsers UC Browser and Baidu Browser, which were both found to transmit sensitive user information with either no encryption or easily decryptable encryption. The security issues discovered in UC Browser were also identified in documents leaked by Edward Snowden that indicated the Five Eyes intelligence alliance (NSA, GCHQ, CSE, ASD, and GCSB) had used these vulnerabilities as a means of identifying and tracking users. We have also published a primer on mobile security and privacy, entitled The Many Identifiers in Our Pockets, which provides further background on the types of personal data commonly collected and transmitted by mobile devices.

In addition, we have conducted research into keyword censorship and surveillance in TOM-Skype and keyword censorship in messaging platform Sina UC, as well as a comparative analysis of mobile chat applications popular in Asia, including WeChat, LINE and KakaoTalk. We have also examined censorship practices in Tencent's other flagship application, WeChat. The overall aims of this research are to employ a mixed methods approach, including reverse engineering and other technical analysis methods, to better inform users about the security and privacy risks of the applications they use, and, where relevant, to engage the companies who produce these applications in a process of responsible disclosure to mitigate risks to users.

On March 17, 2016 we sent detailed questions to Tencent inquiring about the possible reasons for the collection and insecure transmission of user data to QQ servers. Those questions can be found here. As of the date of publication, we have not received a reply. At the end of the report, we discuss several possible underlying causes for the strikingly similar issues we found in the three web browsers produced by China-based companies that we have examined.

QQ Browser Background

QQ Browser is a web browser for the Android, Windows, OS X, and iOS platforms, developed by Tencent. The Android version of QQ Browser was first released outside of China in November 2011, and a version for the OS X platform was released in March 2012. Alongside competitors such as UC Browser and Baidu Browser, QQ Browser is one of a number of third-party mobile browsers which are particularly popular in Asia. QQ Browser offers a number of features tailored for mobile users, such as image compression to conserve data usage.

The app has been very popular, particularly in China, where in January 2013 it was the eighth most-installed application in both the iOS and Android categories. By December 2015, the browser was estimated to have a penetration rate among Chinese mobile browser users of 48.3%. Recent statistics for usage outside of China are difficult to come by, but the application had 16 million non-Chinese users in 2012, with the vast majority based in other countries in Asia.

Tencent is one of China's largest technology companies, with 2015 revenues exceeding 102 billion RMB (USD$15.8b) and an April 2015 market valuation of USD$206 billion. Amongst its many online offerings, the company has developed two of the world's most popular instant messaging platforms: WeChat (known in China as 微信, or *Weixin*) and QQ. Tencent reported that QQ had 853 million monthly active users (MAUs) in 2015, while WeChat/Weixin has 640 million MAUs during the same period.

Tencent's messaging applications have been the focus of controversy in the past, with China-based dissidents expressing concern that their WeChat communications may have been monitored by Chinese authorities. In response, Tencent stated, "We have taken user data protection seriously in our product development and daily operations, and at the same time, like other international peers, we comply with relevant laws in the countries where we have operations."

Like many companies, Tencent has both a Terms of Service and Privacy Policy that describe the types of user data collected by their applications and services and the conditions under which that data can be shared. The English-language version of the Privacy Policy states that "We use a variety of security technologies and procedures for the purpose of preventing loss, misuse, unauthorised access or disclosure of Information. In some of our services, we will use encryption technology (such as SSL) to protect certain sensitive Information provided by you to us."

In addition, the Privacy Policy states that "[y]ou agree that we or our affiliate companies may be required to retain, preserve or disclose your Personal Information: (i) in order to comply with applicable laws or regulations; (ii) in order to comply with a court order, subpoena or other legal process; (iii) in response to a request by a government authority, law enforcement agency or similar body (whether situated in your jurisdiction or elsewhere); or (iv) where we believe it is reasonably necessary to comply with applicable laws or regulations."

Responsible Disclosure & Notification

Tencent operates a Security Response Center, which describes the process for submitting security vulnerabilities, and the types of vulnerabilities that are considered in scope. On February 5, 2016, we used this site to submit a security vulnerability report to Tencent. We indicated that we would publish our report no sooner than 45 days after notification, in line with international standards on vulnerability disclosure.

On or before March 14, 2016, Tencent released version 9.3.6872 of the Windows version of QQ Browser. On March 2, 2016, Tencent released version 6.4.2.2075 of the Android version of the application. We performed an analysis of both updated versions to determine if the issues we identified had been resolved. The results of that analysis are described in the "Update: Analysis of updated versions of QQ Browser" section at the end of this report.

We have documented all correspondence with Tencent related to these security issues in an Appendix at the end of this report.

Technical Analysis

We analyzed version 6.3.0.1920 of the Android version and version 9.2.5478 of the Windows version of QQ browser using a number of tools. We used tcpdump and Wireshark to capture and analyze network traffic, and we used machine code and bytecode disassemblers, decompilers and debuggers, including JD, JADX and IDA, to analyze program behaviour.

We found the browsers communicate back to their servers using a common mechanism that leaks different kinds of personal information, and we found them to have multiple security vulnerabilities in their self-updating processes.

Our technical analysis is split into three parts. The first part describes a basic structure used by both analyzed versions of QQ Browser to transmit data to QQ servers. The second part contains our analysis of the personal user data transmitted, as well as the software update process, for the Android version of the application. The third part describes our analysis of the same features in the Windows version of the application.

Part 1: QQ Browser Data Transmission

Both the Android and Windows QQ browsers we analyzed communicate with QQ's servers using something their software refers to internally as a *WUP request*.

WUP Requests

A WUP request is a binary format that can contain different kinds of values, including integers, floating point numbers, lists, strings, and recursive structures. These requests are sometimes encrypted before being embedded into the body of an HTTP POST request that is sent to its destined URL. We wrote python scripts to decrypt and parse these requests into a human-readable format that are available here. This code also contains all other scripts required to decrypt data that we discuss in this report.

Q-GUID, Q-UA, and Q-UA2 Fields

Q-GUID, Q-UA, and Q-UA2 are the names of fields that appear in the HTTP headers of WUP requests. However, in our description of different WUP requests below, we also use these terms to refer to instances when their respective values additionally appear in the payloads of WUP requests. In the HTTP header, these fields always appear without encryption, although when they appear in WUP requests their format may vary.

The Q-GUID field is populated by a value requested from QQ's servers via a WUP request at initial startup, and after it is received, it is retained by the browser and included without encryption as an HTTP header in most subsequent requests. It is also included in the payloads of many WUP requests in different ways. An example Q-GUID is

```
caed22d728efa6127d53bc0412f888cb
```

GUID likely stands for "globally unique identifier," a kind of 128-bit number used in software that is often generated randomly.

The Q-UA and Q-UA2 values include hard-coded information about the version of QQ browser installed and the type of hardware on which it is installed. Although UA likely stands for "user agent" and contains similar information to an HTTP user agent string, its format is distinct from the user agent HTTP field also included by QQ browser in HTTP headers.

Part 2: Analysis of QQ Browser - Android Version

We analyzed version 6.3.0.1920 of QQ browser for Android, which we downloaded from http://mb.qq.com/.[1] We found that after launching and on certain events, such as viewing a page or checking for software updates, the browser sends WUP requests to http://wup.imtt.qq.com:8080/. When encrypted, these requests are encrypted according to the scheme described below.

For each encrypted WUP request, an AES key is generated according to the following Java code:

```
int i = 10000000 + new Random().nextInt(89999999);
int j = 10000000 + new Random().nextInt(89999999);
return (String.valueOf(i) + String.valueOf(j)).getBytes();
```

Thus, the key is a 128-bit key consisting of 16 ASCII digits. Moreover, the first and ninth bytes can never be zero and neither the first eight nor the last eight bytes can be all nines, and so the keyspace, instead of normally being size 2^{128}, is only size $89999999^2 < 2^{53}$.

This key is then used to encrypt the WUP request with AES+ECB. The AES key is then encrypted with an 128-bit RSA public key with modulus 24540641757374088471004774745869965023463 and exponent 65537. The encrypted AES key is then included in the HTTP request in the *qbkey* HTTP header.

RSA is an asymmetric encryption algorithm, meaning that a different, private key is used for decryption, so the above RSA key cannot be immediately used by someone monitoring traffic to decrypt the AES key and thus the WUP request itself. However, the security of RSA is dependant on the difficulty of the prime factorization of the encryption key's modulus. Once factored, the decryption key can be easily recovered. The above RSA public key is only 128 bits, which is small enough to be easily factored. (RSA keys are traditionally recommended to be at least 2048 bits.) We were able to use Wolfram Alpha, an online mathematics engine, to factor the modulus in less than one second:

1 It is notable that all of the locations from which we downloaded the clients used unencrypted HTTP connections, which presents another potential security concern.

http://www.wolframalpha.com/input/?i=factor+24540641757374088471004774586996502346 3

which yielded the following two prime factors:

14119218591450688427 x 17381019776996486069

Using these factors, any man-in-the-middle monitoring traffic can easily decrypt the AES key of every encrypted WUP request and use that AES key to decrypt the request itself.

We monitored traffic sent by the browser and used this key to decrypt all of the WUP requests sent by the browser. We found multiple WUP requests that transmit easily decryptable personal information. In Figure 1, we show an example of a decrypted WUP request that has been parsed into more readable form by a script that we wrote.

Figure 1: Example of decrypted WUP request as presented by our tool. Sensitive numbers have been manually replaced with "#" symbols.

Below we identify some of the most significant of these requests followed by the personal information that each transmits:

WUP Request	Time Observed	Encryption
profileInfo.profileInfo	Browser startup	Easily decryptable
hotword. getAssociationalWords	Typing into address bar	Not encrypted
Security.doSecurityReqest [sic]	Page view	Easily decryptable
proxyip.getIPListByRouter	Browser startup	Easily decryptable
pkgcenternew.checkUpdate	Update check	Easily decryptable

Data point	Description of data point	Encryption	WUP Requests
IMEI	The International Mobile Equipment Identifier is a string of numbers that is unique for every device.	Easily decryptable	profileInfo. profileInfoSecurity.do
			SecurityReqest
			proxyip. getIPListByRouter
			pkgcenternew. checkUpdate
IMSI	The International Mobile Subscriber Identification number uniquely identifies the user.	Easily decryptable	profileInfo.profileInfo
Q-GUID	Unique string used by QQ Browser to identify a particular user.	Not encrypted	hotword. getAssociationalWords
			Security. doSecurityReqest
			proxyip. getIPListByRouter
Q-UA2	A value used by QQ Browser that identifies the version of the application used and the type of hardware on which it is installed.	Not encrypted	hotword. getAssociationalWords
			Security. doSecurityReqest
QQ username	The user's QQ username.	Easily decryptable	profileInfo.profileInfo
Screen pixel dimensions	The dimensions in pixels of a user's device screen.	Easily decryptable	profileInfo.profileInfo
WiFi MAC address	A Media Access Control address uniquely identifies wireless transmitters like Bluetooth and Wi-Fi chips in the device.	Easily decryptable	profileInfo.profileInfo*
In-range WiFi access point MAC addresses	The Media Access Control addresses of all nearby WiFi access points.	Easily decryptable	profileInfo.profileInfo
			proxyip. getIPListByRouter

Data point	Description of data point	Encryption	WUP Requests
SSID of connected WiFi access point	The name of the WiFi access point to which the user is connected.	Easily decryptable	proxyip.getIPListByRouter
Android ID	A unique number generated when the operating system is first run that can be used to track users.	Easily decryptable	pkgcenternew.checkUpdate
Address bar contents	The contents of the address bar typed in by a user (e.g., a search query).	Not encrypted	hotword.getAssociationalWords
Full page URL	The full URL of each page visited in the browser.	Easily decryptable	Security.doSecurityReqest

**Wifi MAC address is encrypted with DES+ECB with key "\x25\x92\x3c\x7f\x2a\xe5\xef\x92"*

The responses to WUP requests were also easily decryptable. WUP responses were not encrypted using the asymmetric algorithm described earlier but instead used a purely symmetric algorithm, and thus we were not required to factor any key. Namely, they were encrypted using MTEA+MCBC with the following hard-coded, ASCII-encoded key:

```
"sDf434ol*123+-KD"
```

Interestingly, this encryption process employed in QQ Browser utilizes the same non-standard MTEA+MCBC implementation we observed in our study of Baidu Browser. (See Figure 4 and accompanying text in our report.)

Since the algorithm is symmetric, the same key is used to both encrypt and decrypt these responses. Thus, any man-in-the-middle can use this key to perform an active attack by spoofing the response from QQ servers. We demonstrate this by attacking QQ browser's self-updating process.

Vulnerable software update process

A *pkgcenternew.checkUpdate* request, as described earlier, indicates that a software update is available. The response to this request may contain a link to a new APK to download, an MD5 hash of that APK, and a textual description of the changes contained in that update. Android does not allow an APK to upgrade an app if the APK is signed with a different digital signature than that of the currently installed app, and so this attack cannot be used to replace QQ browser with an arbitrary APK; however, it may still be used to install a new app, and a properly crafted APK using the name and logo of QQ browser could be used to deceive a user into installing a malicious APK (see Figure 2).

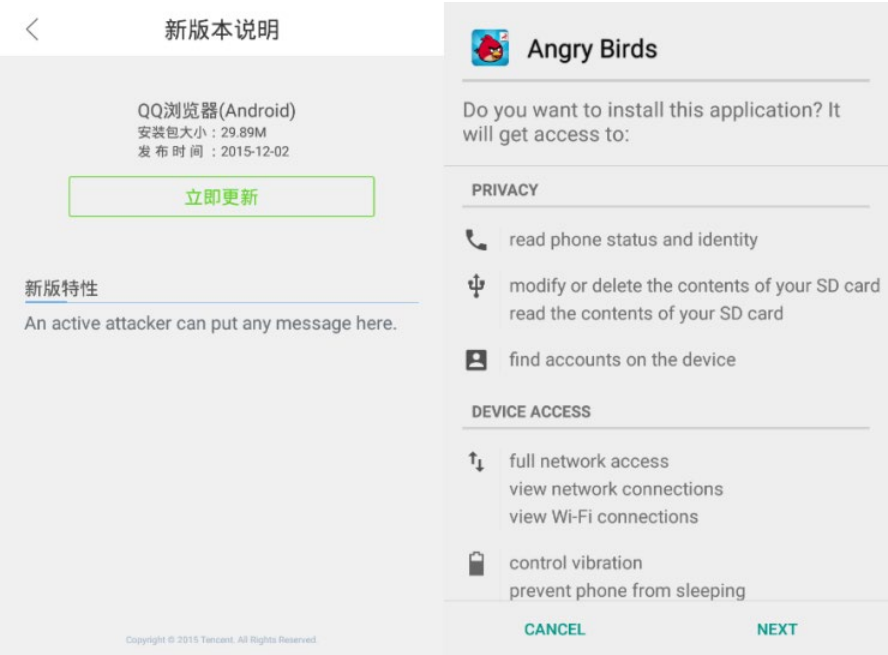

Figure 2: Example man-in-the-middle attack on QQ Browser's updater. On the left, we injected a custom update description. On the right, after the update is downloaded, the browser prompts the user to install the Angry Birds APK (an actual attacker might instead craft an app called "QQ Browser" with an icon similar to that of QQ Browser to further convince the user to install it).

It is worthwhile noting that the current unavailability of the Google Play store in China creates the need for Android applications targeting users in China to find alternative methods of updating. Without the option of using the Play store's software update process, developers are required to implement their own auto-updating mechanism, which as demonstrated in this case can introduce new opportunities for vulnerabilities in an app's update process. It has been rumoured that a version of the Play store for the Chinese market will be launched in 2016.

Part 3: Analysis of QQ Browser - Windows Version

We analyzed version 9.2.5478 of QQ browser for Windows, which we downloaded from http://browser.qq.com/. Although the Windows version also communicates with QQ's servers using WUP requests, it differs from the Android client in how and when communications are encrypted. Moreover, the Windows version uses the MTEA+MCBC algorithm to encrypt WUP requests, a symmetric encryption algorithm, rather than the asymmetric RSA-based algorithm used by the Android version. (Encrypted WUP responses use MTEA+MCBC in both the Android and Windows versions.)

Although we observed the Android version sending WUP requests solely to http://wup.imtt.qq.com:8080/, we observed the Windows version sending WUP requests to a variety of URLs including http://qbwup.imtt.qq.com, http://wup.html5.qq.com, and http://wup.imtt.qq.com:8080.

The Windows version of the browser also tracks what we henceforth call the machine's *hardware fingerprint*, which we found included in many of the WUP requests sent. The hardware fingerprint is the MD5 hash of the concatenation of the machine's

1) Network MAC address
2) Hard drive disk serial number
3) Hard drive disk model number
4) Hard drive disk controller version number

e.g.,

```
md5("080027B09CC2" + "VB7c666e15-ef97c40b" + "VBOX HARDDISK" + "1.0").
```

Since MTEA+MCBC is purely symmetric, any man-in-the-middle observing traffic can use the hard-coded encryption key to easily decrypt all encrypted WUP requests. As before, we monitored traffic sent by the browser and decrypted all of the WUP requests. We found multiple WUP requests that leak easily decryptable personal information. We have listed below the most significant of these requests followed by the personal information that they transmit:

WUP Request	Time Observed	Encryption
devicesniffer. DeviceSnifferHandle	Browser startup	Easily decryptable
login.login	Browser startup	Easily decryptable
qbkpireportbak.stat	Browser startup	Easily decryptable*
qbpcstat.stat	Browser startup	Easily decryptable*
qbindexblacklist.testUrl	Search query or URL entered into address bar	Not encrypted

** The WUP request is itself not encrypted but contains a nested WUP payload that is encrypted with DES+ECB (a symmetric, easily decryptable algorithm) using the key "\x62\xe8\x39\xac\x8d\x75\x37\x79".*

Data point	Description of data point	Encryption	WUP Requests
Hardware fingerprint	Hash of network MAC address, hard drive disk serial number, hard drive disk model number, hard drive disk controller version number.	Not encrypted	login.login qbkpireportbak.stat qbpcstat.stat qbindexblacklist.testUrl
Q-GUID	Unique string used by QQ Browser to identify a particular user.	Not encrypted	devicesniffer.DeviceSnifferHandle login.login* qbkpireportbak.statqbpcstat.stat qbindexblacklist.testUrl
Q-UA	A value used by QQ Browser that identifies the version of the application used and the type of hardware on which it is installed.	Not encrypted	login.login qbindexblacklist.testUrl
Machine IP Address	The Internet Protocol address of a user's device.	Easily decryptable	devicesniffer.DeviceSnifferHandle
Machine hostname	The Windows hostname of the user's computer.	Easily decryptable	devicesniffer.DeviceSnifferHandle
Gateway MAC address	The Media Access Control address of the gateway used by the user's computer.	Easily decryptable	devicesniffer.DeviceSnifferHandle
Windows version and build	The version and build of Windows running on the user's computer.	Not encrypted	qbkpireportbak.stat qbpcstat.stat qbindexblacklist.testUrl
Internet Explorer version	The version of Internet Explorer installed on the user's computer.	Easily decryptable	qbkpireportbak.stat Qbpcstat.stat
QQ Browser version	The version of QQ Browser installed on the user's computer.	Not encrypted	qbindexblacklist.testUrl
Hard drive serial number	The unique serial number of a user's hard drive.	Easily decryptable	qbkpireportbak.stat qbpcstat.stat

Data point	Description of data point	Encryption	WUP Requests
Windows user security identifier	Unique identifier Windows randomly generates for each Windows user.	Easily decryptable	qbkpireportbak.stat qbpcstat.stat
Full page URL	The full URL of each page entered into the address bar.	Not encrypted	qbindexblacklist.testUrl

**Q-GUID encrypted with 3DES+ECB with key "\x63\xd7\x90\x63\x3c\x0e\x2f\xc3\x46\xef\x85\x37\x42\x1f\x9d\x4a\x46\x3d\x58\xf3\x8a\x95\xec\x84" with plaintext first interleaved with random bytes such that the 1st, 3rd, 5th, etc. are the 1st, 2nd, 3rd, etc. bytes of Q-GUID and 2nd, 4th, 6th, etc. bytes of plaintext are randomly chosen.*

We found that the Windows version also leaked personal information outside of WUP requests when a user visits a page. We found that the full URLs of every viewed page, whether entered into the address bar or reached via a link or another means, were sent using MTEA+MCBC encryption to http://masterconn.qq.com/ using the key:

```
"\x8a\x0d\x75\x73\x90\x03\x4a\xd2\xb5\x25\xab\xe2\x31\xe2\x9f\x6f"
```

Vulnerable software update process

Requests checking for software updates are sent via JSON to http://update.browser.qq.com/qbrowser. While we found that similar to the Android version both the request for updates and the server's subsequent response are not encrypted, we found that unlike the Android version the Windows version verifies the digital signature of the downloaded update. However, we found two attacks against the update process that any man-in-the-middle performing an active attack could still utilize to remotely run code on a QQ Browser user's machine.

The first attack is a type of directory traversal attack. Normally when an update is available, QQ's servers respond with the URL of an EXE to download, its MD5 hash, a textual description of the new features and fixes provided by the update, and the filename and location where the EXE will be saved. We found that the filename is not sanitized to remove directories from its name and so by including directory traversal, an active attacker can overwrite any file to which the user has permission to write. For instance, by naming the saved file

```
../../../../../../../../../program files/tencent/qqbrowser/qqbrowser.exe
```

we overwrote QQ Browser with an arbitrary program that would execute the next time the user attempted to run QQ Browser. While in our testing the program we overwrote QQ Browser with was a benign program, a malicious attacker could use this attack to install hidden spyware or malware.

The second attack demonstrates that checking for digital signatures is in and of itself insufficient for verifying the authenticity of a software update. A digital signature verifies that the downloaded EXE was written and signed by Tencent, but it does not verify that it will update QQ Browser to a newer version--it could be any EXE that has ever been signed by QQ. We found an older Web installer for QQ Browser that performs no digital signature checks (itself only using symmetric cryptography), and so on update, we had the user's QQ Browser "upgrade" to the vulnerable Web installer for QQ Browser, which then proceeded to download and execute an arbitrary EXE of our choosing (see Figure 3).

Figure 3: Example man-in-the-middle attack on QQ Browser's self-updater by first injecting a vulnerable Web installer and then injecting our arbitrary program. A benign program that displays "Oh Hai There" was used as the payload, but any arbitrary program such as spyware or malware could be injected.

Discussion

This report raises a number of serious security issues for QQ Browser users. The application collects and transmits personally identifiable data points in a manner that leaves this data vulnerable to surveillance by third parties. Further, deficiencies

in the software update process leave users vulnerable to having arbitrary code, such as a malicious spyware program, inserted by a third party and executed on their devices. Most troubling is the fact that users would generally be unaware of these risks -- unaware that such data is being collected and transmitted, and potentially unaware that a properly crafted malicious software update attack could lead to malicious code being installed on their devices.

However, as our previous research has shown, problems of this nature are not unique to any one particular application, operating system, or company. Our analyses of QQ Browser, Baidu Browser, and UC Browser have shown that all three -- popular browsers made by three of the biggest tech companies in the world -- contain strikingly similar security vulnerabilities. Therefore, QQ Browser is not unique in collecting this sensitive user data and transmitting it either without encryption or with easily decryptable encryption methods. In light of these similarities, the security concerns raised need to be evaluated through a broader context of mobile and application security generally, rather than focusing on any one particular company or application.

Web browsers are trusted to carefully handle sensitive information inputted by users and securely transmit it to Web servers. However, QQ Browser and the other browsers studied violate this standard of trust by not only collecting sensitive user data themselves, but then also insecurely transmitting it. Even in cases where asymmetric cryptography is used to transmit sensitive user data, it is used inconsistently. The Android version of QQ Browser, which used the asymmetric RSA algorithm, used a key size that was too small to be effective and did not meet the recommended practice of using 2048 bit keys. This shortcoming illustrates the need for developers to use well-tested implementations of well-studied protocols, such as OpenSSL, a widely-used and well-accepted method of transmitting sensitive data in a more secure manner.

Beyond the criticism about the methods these applications use to transmit personal data, these findings also raise bigger questions about why such data is being collected and transmitted in the first place. Mobile devices transmit a large range of uniquely revealing identifiers, the collection of which can raise serious privacy and security concerns for users. While these myriad data points available on a user's device can permit developers to deliver efficient, highly customized services, the breadth of data points collected by these mobile browsers is arguably excessive, and would likely raise concerns among the users of these applications were they aware of it -- especially when vendors are unable to properly secure such data. The

collection of such fine-grained information about a user, a user's device, and a user's online behavior (and its insecure transmission) would be especially concerning for high-risk users, which in China could include democracy activists, journalists, human rights advocates, lawyers, and others.

Evaluating underlying causes for the similarities

That the three China-based browser applications we have examined all evince strikingly similar data gathering and insecure data handling problems raises an obvious question of whether there is some underlying cause for the similarities. There are at least four possible explanations, all of which require further research.

1) The underlying similarities could simply be the result of coincidence -- in other words, no underlying cause. It is possible that the engineers all settled on the same design choices independently. However, in the case of QQ Browser and Baidu Browser, where the same non-standard MTEA+MCBC algorithm was found to be used, both companies independently creating exactly the same encryption algorithm is highly unlikely to be a coincidence, and so coincidence seemingly cannot explain our findings entirely.

2) There could be common engineering norms or industry standards which the browser developers are following, and which are particularly loose in terms of privacy and security with respect to China's industry. After all, data overreach -- in the form of excessive requested permissions -- is a common characteristic of the application sector worldwide. Targeted advertising is a primary motivation for developing applications, and so it is not surprising to find that there are industry norms and pressures to build in as much functionality to gather up as much information about users as possible -- to err on the side of excess, in other words. It is entirely possible that design choices were made as a result of these industry norms and practices, especially in China, where there is a rapidly expanding and highly dynamic user base for application development. Developers may not have faced outside pressure to implement strong security protections in their applications, and norms regarding what constitutes private or personally identifiable data may vary or be poorly appreciated. These applications' lax attention to security, combined with aggressive information gathering, may simply

be the product of industry norms, of which the China case is an extreme example at the far end of the spectrum.

3) There could be directives from the government, or informal pressure coming from state security officials on company executives, and by extension the engineers, to build in a kind of "surveillance by design." To be sure, we have no explicit evidence that the government of China directed these specific design choices. And, the questions we asked the companies about government directives or influence have not been directly answered. However, we know that China maintains an extensive censorship and surveillance regime and all companies are required by law to follow state regulations in this respect.[2] Last year, state-run Xinhua News Agency reported that police officers would be stationed within the country's major technology companies to fight criminal activity online. The Chinese government has also asked major U.S. tech companies to sign a pledge committing that, amongst other things, its products will be "secure and controllable," raising fears about legal requirements to implement surveillance backdoors. There is a strong expectation that China's information and communications technology sector will responsibly police their networks, as illustrated in Article 19 of China's Counter-Terrorism Law: "Telecommunications operators and internet service providers shall, according to provisions of law and administrative regulations, put into practice network security systems and information content monitoring systems, technical prevention and safety measures, to avoid the dissemination of information with terrorist or extremist content. Where information with terrorist or extremist content is discovered, its dissemination shall immediately be halted, relevant records shall be saved, and the relevant information deleted, and a report made to public security organs or to relevant departments." In such a climate, it is reasonable to hypothesize that company officers put in place wide-reaching data gathering functionalities either at the request of, or to appease the

2 China's Counter-Terrorism Law, which came into effect on January 1, 2016, includes requirements for telecommunications operators and Internet service providers to "provide technical interfaces, decryption, and other technical support assistance to public security organs and state security organs conducting prevention and investigation of terrorist activities in accordance with law". While the final text of the law appeared to back away from controversial requirements in earlier draft versions of the law, which required companies to provide backdoor access and submit encryption keys to authorities, the passed version of the law still requires companies to provide technical assistance and potentially decrypt user communications. While the precise definitions of what types of companies are included and what types of assistance they would be required to provide are still forthcoming, in all likelihood "a broad range of companies with an internet presence in China" will be included.

preferences of, China's security services. More research is needed to evaluate this hypothesis.

4) Finally, it is possible that the design choices are a subtle combination of points 2 and 3 above. In other words, a culture of "collect as much as possible" and lax data transmission security reinforce each of the industry's and government's needs, but in an unspoken and largely informal way. In this case, companies and their engineers are following industry norms, which also serve to benefit the interests of government surveillance while complying with the broad spirit of applicable laws. If this were an accurate reading, only when and if industry standards were tightened up by the companies would government authorities feel compelled to intervene and enforce some discipline on them and their engineers (much in the same fashion that Apple has faced pressure from the U.S. Department of Justice after Apple's tightening up of device security). It is noteworthy, in this respect, that government signals intelligence practices as evidenced by what has appeared publicly (e.g., the Snowden disclosures) already make extensive use of the type of data leaked from applications that we document in these reports. Having this type of data collected and archived by private companies on servers inside mainland China and transiting through China-based networks would conveniently enable such signals intelligence collection practices for Chinese security agencies, and would likely be looked upon favourably by authorities.

Regardless of which of the above is the answer, the effect is the same: the many millions of users of the applications we have studied are at risk of serious privacy and security compromises.

Questions for Tencent

On March 17, 2016, we sent a letter to Tencent with additional questions about the security vulnerabilities we identified. We sent the letter again on March 23, 2016. The letter is reproduced here.

Update: Analysis of updated versions of QQ Browser

We notified Tencent of the security issues in QQ Browser on February 5, 2016. To address these issues, on March 2, 2016, Tencent released version 6.4.2 of the Android

version, and on or before March 14, 2016, Tencent released version 9.3.6872 of the Windows version.

Analysis of Android version 6.4.2

Our analysis of version 6.4.2 of the Android version shows that some of our reported issues have been partially resolved and some remain unresolved. The Android version now uses a 1024-bit RSA key instead of a 128-bit RSA key to encrypt session keys. Moreover, session keys are now sampled from the entire 128-bit AES keyspace instead of being restricted to certain ASCII digits. This greatly increases the strength of the encryption used to transmit sensitive data. However, while a 1024-bit RSA key cannot be easily factored, we recommend using at least a 2048-bit key. Moreover, due to their use of "plain RSA," their implementation may still suffer from other vulnerabilities stemming from, for example, their lack of any key padding, such as OAEP.

We also found that the server now encrypts its responses using the session key instead of a hardcoded key. This protects the server's responses from being easily decrypted and injected, which makes any man-in-the-middle attack on QQ's update process more difficult. However, the strength of the encryption of these responses and their resistance to man-in-the-middle attacks also suffers from the caveats mentioned in the earlier paragraph.

We found that text typed into the address bar for searching or to go to a URL is still sent unencrypted.

Analysis of Windows version 9.3.6872

Our analysis of version 9.3.6872 of the Windows version shows that some of the issues we reported have been resolved and some remain unresolved. We found that all WUP requests sent by the Windows browser still use the same symmetric, easily decryptable algorithm. However, among the five different WUP requests that we describe the Windows version as making in this report, we did not observe three (*qbindexblacklist.testUrl*, *qbkpireportbak.stat*, and *login.login*) in the latest version; however, we observed the browser still sending the other two (*devicesniffer.DeviceSnifferHandle* and *qbpcstat.stat*). This means that almost all sensitive identifiers that we originally reported, including MAC addresses, hard drive serial numbers, and Windows user security identifiers are still being sent using symmetric cryptography that can be easily decrypted.

We also found that the URL for each page visited is still sent to http://masterconn.qq.com using the same easily decryptable encryption; however, now only the protocol and domain of each page visited are sent, not the full URL.

Software updates are now checked via HTTPS instead of HTTP. This secures users against both of the attacks on the Windows version's update process that we describe in this report by preventing attackers from being able to perform man-in-the-middle attacks.

Appendix

We have documented all correspondence with Tencent related to these security issues here:

Date	Contact
February 5, 2016	We submitted a security disclosure to Tencent via their online disclosure mechanism at: http://en.security.tencent.com/.
February 16, 2016	Status of report changed to "confirmed," comment is left thanking us for our report.
February 17, 2016	We inquire what steps will be taken to resolve the reported issues and what the timeline will be for their resolution.
February 21, 2016	They indicate that a new version fixing the vulnerabilities will be released in March.
February 24, 2016	They indicate that a new version fixing the vulnerabilities will be released "next week."
March 3, 2016	They ask for us to leave an address to send a bug bounty gift. (Their bug bounty is described on their security disclosure site as company swag such as "Tencent dolls").
March 8, 2016	They state that fixed versions have been released.
March 10, 2016	We report our findings analyzing version 6.4.2 of the Android version. We also report that we could not find any changes in the Windows version and inquire whether we are analyzing the right version.
March 14, 2016	Tencent responds providing a link to the latest Windows version saying that they have fixed a number of the issues we reported. They report that they have upgraded the update check to use HTTPS. In addition, they report that they will only send the domain of viewed pages instead of the full URL, as a means of judging if a website is malicious. They also mention that they will still send the GUID since it is not personally identifying.
March 18, 2016	We respond confirming the changes in the Windows version and saying that we can still see other sensitive information in many WUP requests such as MAC addresses and hard drive serial numbers.
March 20, 2016	They respond saying that they have tried their best to resolve all reported problems, and they inquire as to whether we have any new problems to report.

Date	Contact
March 22, 2016	We say that aside from the problems we have already reported we have no new issues to report. We say that we will be publishing our findings on March 28.
March 22, 2016	They justify the collection of hard drive serial numbers by saying that “Hard drive serial number is use for identifying independent user, so that QQ Browser can offer personalized service.” They then inquire as to where we will be releasing our published findings.
March 23, 2016	We say that we will release our findings on https://citizenlab.ca/ . We also link to the letter that we sent to Tencent and ask if they know of an appropriate contact to answer the letter’s questions.
March 24, 2016	They thank us for our feedback.