Web Access Monitoring Mechanism for Android WebView

Yuta Imamura
Hiroyuki Uekawa
Graduate School of Natural Science and Technology, Okayama University
Okayama, Japan

Yasuhiro Ishihara
Faculty of Engineering, Okayama University
Okayama, Japan

Masaya Sato
Toshihiro Yamauchi
Graduate School of Natural Science and Technology, Okayama University
Okayama, Japan
yamauchi@cs.okayama-u.ac.jp

ABSTRACT

In addition to conventional web browsers, WebView is used to display web content on Android. WebView is a component that enables the display of web content in mobile applications, and is extensively used. As WebView displays web content without having to redirect the user to web browsers, there is the possibility that unauthorized web access may be performed secretly via WebView, and information in Android may be stolen or tampered with. Therefore, it is necessary to monitor and analyze web access via WebView, particularly because attacks exploiting WebView have been reported. However, there is no mechanism for monitoring web access via WebView. In this work, the goals are to monitor web access via WebView and to analyze mobile applications using WebView. To achieve these goals, we propose a web access monitoring mechanism for Android WebView. In this paper, the design and implementation of a mechanism that does not require any modifications to the Android Framework and Linux kernel are presented for the Chromium Android System WebView app. In addition, this paper presents evaluation results for the proposed mechanism.

CCS CONCEPTS

• **Security and privacy** → **Network security**; *Mobile platform security*;

KEYWORDS

Android, WebView, Web access monitoring

1 INTRODUCTION

In addition to traditional web browser apps, Android applications (or "Android apps") can display web content in themselves. A component called WebView is used to display web content into an Android app without redirecting users to web browser apps. Mobile app developers heavily use WebView for displaying web content in their apps. A previous study shows that WebView is used by approximately 86% of the Android apps in the Android app store managed by Google, as of 2011 [1]. Additionally, as of June 2014, it was reported that 85% of Android apps use WebView [2]. We have also studied how many Android apps are using WebView as of June 2017. We scanned for the WebView-related description in the Java code of 32 Android apps in Google Play. As a result, we estimate that approximately 97% of Android apps use WebView, indicating that WebView is used extensively.

Monitoring web access via WebView is necessary because WebView may be used for the attacks. JavaScript downloaded from an external site via WebView can be executed, and it could be used for the attacks. In addition, although the attacks using JavaScript and countermeasures for these attacks have been reported, to the best of our knowledge, there is no mechanism for monitoring web access via WebView. Therefore, there is no means to understand what kind of access via WebView is performed. Moreover, in the attacks using JavaScript described above, we cannot understand what kind of communication is done. Thus, the mechanism that can monitor communication via WebView is necessary.

Web access on Android can be monitored by using HTTP proxies or packet capture tools. However, these methods cannot distinguish web access via WebView from other web access options on Android. There is no prior study that identifies WebView communication. Moreover, these methods cannot analyze the communication contents of web access encrypted by TLS/SSL because they cannot decrypt the communication contents.

In this paper, we propose a web access monitoring mechanism for Android WebView, and describe its design and implementation. The proposed mechanism can monitor all web access via WebView on Android. In addition, this mechanism does not require any modification of the Android Framework and the Linux kernel, so that there is advantage it can be introduced by just replacing WebView with a modified version. Furthermore, we describe the evaluation results for the proposed mechanism.

In summary, we made the following contributions:

- We have pointed out the problem that there is no web access monitoring mechanism for WebView despite of the necessity. In addition, we have reported that there is no previous study that analyze Android apps using WebView based on WebView communication logs, in our survey.
- We have designed the web access monitoring mechanism for Android WebView, which addresses the above problems. This makes it possible to distinguish web access via WebView from other access and to monitor it. Moreover, the proposed mechanism can acquire various information, specifically the communication contents of encrypted web access and the package name of the Android app.

ACSW 2018, January 30-February 2, 2018, Brisbane, QLD, Australia

ACM ISBN 978-1-4503-5436-3/18/01…$15.00
https://doi.org/10.1145/3167918.3167942

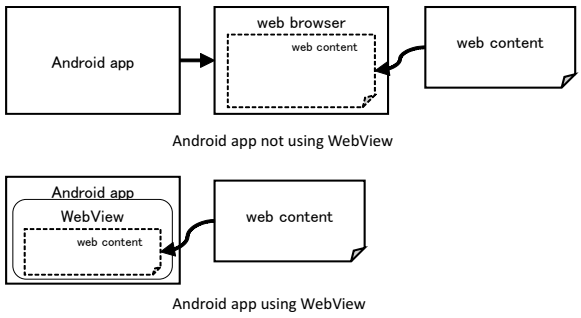

Figure 1: Method of displaying web content on Android

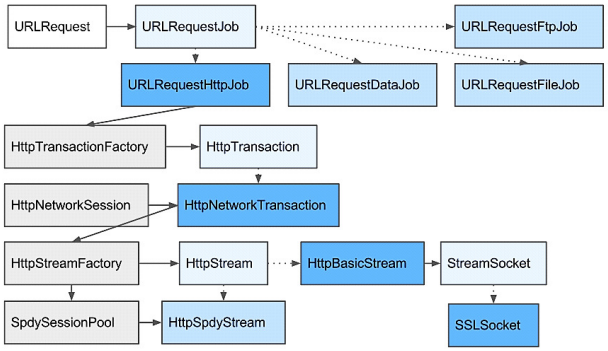

Figure 2: Chromium WebView [5]

- We have reported the effectiveness of the proposed mechanism and its performance overhead. Operation experiment showed that the proposed mechanism can acquire the information necessary for analysis. The performance measurement showed that the overheads in acquiring the information at sending the HTTP request, receiving the HTTP response header, and receiving the HTTP response body are 0.234 ms, 0.099 ms, and 1.978 ms, respectively. Moreover, this measurement showed that the data size of the acquired information are 0.62 KB, 0.35 KB and 225.28 KB, respectively. These results are reasonable, because above overheads respectively depend on the data size of the information acquired by each processing. In addition, the total overhead of each processing is approximately 2.3 ms, which is very short.

2 BACKGROUND

2.1 WebView

WebView is a component that makes it possible to display web content on Android apps without having to redirect the user to web browser apps. Figure 1 shows two methods of displaying web content on Android. As shown in Figure 1, when an Android app that does not use WebView attempts to display web content, the user is redirected to a web browser app, where the web content is displayed. Moreover, the Android app using WebView can display the web content without redirecting to a web browser app.

WebView has used different browser engines in each Android version. WebView implementation from Android 4.1 to Android 4.3 uses WebKit [3], whereas since Android 4.4 uses Chromium [4], which is called Chromium WebView.

WebView up to Android 4.4 was implemented within the Android Framework. On the other hand, WebView since Android 5.0 has been separated from the Android Framework and is implemented as the Android System WebView app. This change allows us to update WebView from Google Play without updating Android.

2.2 Network Stack of Chromium WebView

Figure 2 (cited from [5]) shows the network stack of a Chromium-based web browser. The Chromium project provides a web browser for many platforms. Although web browsers developed by the Chromium project differ in terms of front end and presence/absence of functions, etc., the implementation such as communication processing exhibits almost no differences among platforms. The implementation is shared and composed of class diagrams as shown in Figure 2. Moreover, all web access by Chromium-based web browsers start with the URLRequest class in Figure 2.

Additionally, Chromium WebView has been developed by Chromium project. Therefore, web access by Chromium WebView similarly starts with the URLRequest class. Chromium WebView is developed using Java and C++, and it consists of the Java layer and the C++ layer. The C++ layer in WebView is equivalent to implementation of a network stack in the Chromium-based web browser.

2.3 Processing flow of Web access via WebView

Figure 3 shows the processing flow of web access via WebView, and its details are described below.

(1) Call methods for web access
An Android app that uses WebView calls a method for web access. Here, the methods loadUrl(), loadData(), loadDataWithBaseURL(), and postURL() are used for web access [6].
(2) Call C++ layer method in WebView using JNI
The method called in step (1) calls C++ layer method in WebView (which uses JNI), which then calls the method for the URLRequest class. The method called in step (1) only displays the Web content of the specified URL, and web access processing is done in the C++ layer in WebView.
(3) Issue a system call
For web access, the C++ layer method in WebView issues a system call.

2.4 Security Issues

As mentioned in Section 1, WebView is used in many Android apps, while there is security issues in WebView. WebView is provided with various APIs. SetJavaScriptEnabled API enables the execution of JavaScript downloaded within WebView. In addition, addJavaScriptInterface API registers Java objects to WebView. This make it possible to all the public methods in these Java objects can be called by JavaScript from inside WebView. In [1, 6–8], the attacks that steal the information in Android by using the addJavaScriptInterface API have been reported. In [8–10], cross-site scripting attacks targeting Android apps using WebView have also

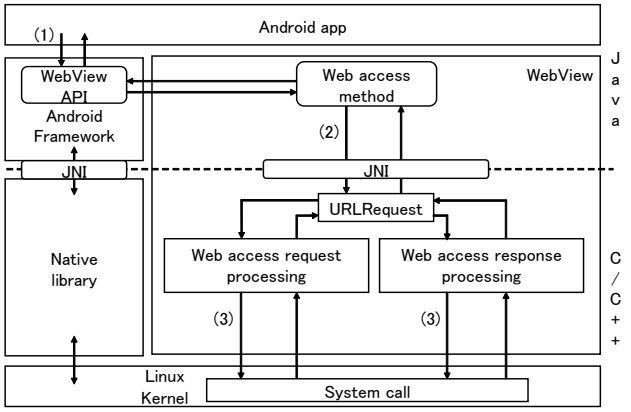

Figure 3: Process of Web access via WebView

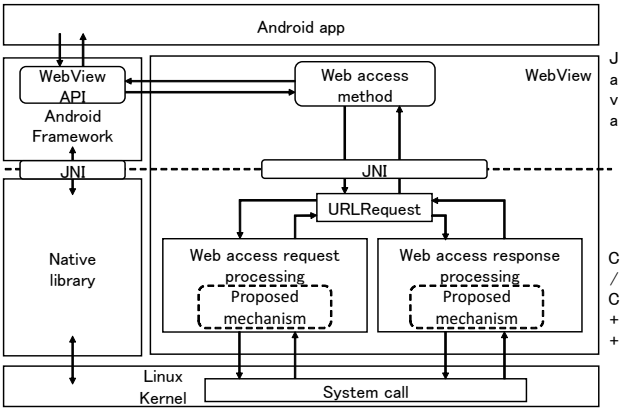

Figure 4: Overview of the proposed mechanism

been reported. In addition, reference [11] reports the attacks that steal a user's confidential information from an Android app by using AdSDK. From above, if a malicious Android app uses WebView, it is possible that the information in Android will be taken or tampered with by an unauthorized entity. Moreover, because JavaScript downloaded from an external site can be executed, there is possibility that users will be damaged by a malicious JavaScript execution.

Some countermeasures based on access control have been presented for the attacks exploiting WebView. In [12], the authors present an access control mechanism that restricts access to device resources based on the user's judgement for mitigating app-repackaging attacks and cross-site scripting attacks. In [6], the authors provide uniform and fine-grained access control for web code running on Android apps using WebView. Moreover, in [13], the authors propose an access control on security-sensitive APIs at the Java object registered within WebView using addJavaScriptInterface.

Although these countermeasures have been presented, to the best of our knowledge, there is no report that focuses on the communication contents of web access via WebView and no web access monitoring mechanism for WebView. Web access on Android can be monitored by using a HTTP proxy or a packet capture tool. However these methods cannot monitor access of WebView and others with the communication distinguished. Moreover, these methods cannot also analyze the communication contents encrypted by TLS/SSL. Thus, in order to address above problem, the web access monitoring mechanism which can distinguish WebView communication from others and analyze encrypted communication contents is necessary.

3 WEB ACCESS MONITORING MECHANISM FOR ANDROID WEBVIEW

3.1 Purpose and Concept

We propose a web access monitoring mechanism for Android WebView as a means of monitoring web access via WebView. The proposed mechanism focuses on web access via WebView exclusive of other web access mechanisms available on Android and aims to monitor the web access.

In order to realize the proposed mechanism, we need to add a function to monitor web access during the processing of web access via WebView. The following can be considered as points at which the function can be added:

(1) Android Framework
(2) WebView
(3) Linux kernel

In this work, our goals are to analyze Android apps using WebView and verify the security of WebView based on WebView communication logs collected by the proposed mechanism. To achieve our goals, we introduce the proposed mechanism into numerous Android devices, and need to collect much data. Therefore, we need to consider ease of introduction. Among the above points, when adding the monitoring function to the Android Framework and the Linux kernel, it is necessary to modify the Android Framework and the Linux kernel for each Android device. On the other hand, when adding the monitoring function to WebView, it is possible to introduce the proposed mechanism by just replacing WebView with a modified one. However, when introducing a new version of WebView containing the proposed mechanism, it is necessary to gain root access on Android. Based on the above considerations, we implemented the monitoring function to WebView.

3.2 Design

Figure 4 shows an overview of the proposed mechanism. As described in Section 2.3, web access via WebView is performed by web access request processing and web access response processing in the C++ layer in WebView. Therefore, we add the monitoring function to web access request and response processing to implement the proposed mechanism. This makes it possible to monitor web access via WebView without changing the processing flow.

3.3 Challenges

To implement the proposed mechanism, the following challenges must be considered.

C1 Information to be acquired
The purpose of this work is to analyze Android apps using WebView based on WebView communication logs collected by the proposed mechanism. Therefore, we consider the information that we need to acquire in order to analyze Android apps using WebView.

C2 Storage format and location of the acquired information
In order to achieve the purpose of this work, we need to consider the storage format and location of the information considered in C1.

3.4 Information to be acquired

We acquire the following information to analyze an Android app using WebView.

(i) HTTP request and HTTP response
Transfer of the data between the web browser and the web server uses HTTP. Additionally, WebView also uses HTTP for data transfer. Therefore, it is possible to acquire the communication contents of web access via WebView by acquiring the data which is HTTP format. In view of the above, the HTTP request and HTTP response is acquired.

(ii) Time stamp
In web access via WebView, the instance, which sends the HTTP request and receives the HTTP response, is created. Moreover, this access is performed asynchronously. Therefore, it is impossible to figure out the correspondence between the HTTP request and the HTTP response. In order to grasp this correspondence, it is necessary to get the information that can distinguish each instance, which sends and receives the HTTP message. Thus, a time stamp is acquired in each this instance and used as an identifier.

(iii) Android app package name
When analyzing an Android app that uses WebView, it is necessary to specify the Android apps which accesses the web content via WebView. Therefore, to identify the Android app, the package name of the Android app is acquired.

(iv) URL
From the scheme name, host name, and path name included in the HTTP request header, the URL of the access destination can be acquired. However, considering analyzing the Android app using WebView, it is better to acquire the URL of the access destination. Therefore, we acquire the URL of the web content as request information.

(v) IP address
The IP address of the communication destination as request information is acquired.
In attacks exploiting web content, some attacks make it difficult to take countermeasures with blacklists by causing the IP address or domain name of an attack site to transition within a short period of time. Therefore, when analyzing an attack site based only on (i) HTTP request and response, it is difficult to identify the attack site, as there is a high possibility that the DNS registration information of the domain of the attack site has changed. In order to identify the attack site, it is necessary to acquire the IP address before the DNS registration information is changed. From the above, by acquiring the IP address of the communication destination, then even if the DNS registration information is changed, the possibility of identifying the true attack site is increased.

(vi) Port number of the web server
For TCP and UDP in the network layer, port numbers are used as identifiers for designating end points of inter-host communication. In web access using HTTP, normally port 80 is used, while in web access using HTTPS, port 443 is used. In this manner, the port number used for each protocol is different. Therefore, by acquiring the port number of the web server, it is possible to determine which protocol is used for web access.

(vii) Connection error during socket connection
In attacks exploiting web content, there is an attack that makes the attack site's IP address or domain name transition within a short period of time. For this reason, there is the possibility of a connection error occurring when accessing the attack site. Therefore, we also acquire connection error information on the socket connection. This increases the possibility of tracing the falsified web content from the legitimate web site to the attack site.

3.5 Storage format and location of the acquired information

In web access via WebView, the instance, which sends and receives the HTTP message, is created each Request/Response, which is a pair of HTTP request and response. Additionally, the communication via WebView is performed asynchronously. Therefore, it is not possible to figure out the correspondence between the HTTP request and the HTTP response. In order to analyze the Android app that uses WebView, it is necessary to understand this correspondence. Thus, to address the above problem, the proposed mechanism saves communication logs each Request/Response. In order to save the acquired information each Request/Response, the proposed mechanism uses the time stamp at generation of the HTTP request header as the file name each Request/Response. This makes it possible that the proposed mechanism acquires the information described in Section 3.4 and saves these information to the file for each Request/Response. In addition, the proposed mechanism acquires the information in a unique format considering ease of analysis and lightweight of preservation and saves the information into internal storage. When analyzing an Android app using WebView, we convert communication logs to JSON format on the analytical computer. Furthermore, the proposed mechanism saves communication logs in the data area allocated for each Android app. This makes it possible to collect communication logs for each Android app using WebView.

3.6 Flow of the Proposed Mechanism

Figure 5 shows the processing flow of web access via WebView applying the proposed mechanism. Table 1 shows the processing of the proposed mechanism and the process timing. As shown in the Figure 5, the proposed mechanism monitors web access via WebView as follows:

(1) When establishment of a connection by the connect() system call fails, the proposed mechanism acquires a connection error

Table 1: Processing of the proposed mechanism

The proposed mechanism	Processing	Information to be acquired	Process timing
Web access request processing	(1)	Connection error during socket connection	Immediately after the connect() system call processing completion
	(2)	Time stamp Android application package name HTTP request header URL IP address Port number of the web server	Immediately after generation of the HTTP request header
	(3)	HTTP request body	Before sending the HTTP request body
Web access response processing	(4)	HTTP response header	After reception of the HTTP response header
	(5)	HTTP response body	After reception of the HTTP response body

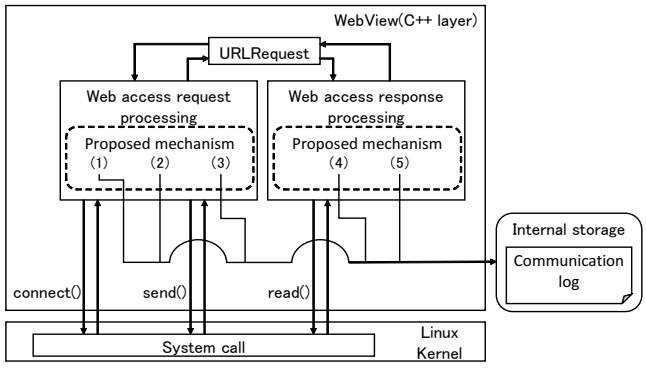

Figure 5: Processing flow of WebView applying the proposed mechanism

at the socket connection and saves in the internal storage of Android.

(2) Immediately after generating the HTTP request header, the proposed mechanism acquires the time stamp, the package name of the Android app, the HTTP request header, the URL, the IP address and port number of web server, and saves these information to internal storage.

(3) When using the POST method, the proposed mechanism acquires the HTTP request body before sending the HTTP request body and save it in internal storage of the Android device.

(4) After receiving the HTTP response header through the read() system call, the proposed mechanism acquires the HTTP response header and save it in internal storage.

(5) After receiving the HTTP response body through the read() system call, the proposed mechanism acquires the HTTP response body and save it in internal storage.

3.7 Effect

By introducing the proposed mechanism, the following become feasible.

E1 Monitoring of web access via WebView
In order to realize the proposed mechanism, we add the monitoring function to the C++ layer in WebView. This makes it possible to monitor web access via WebView on Android.

E2 Analysis of Android apps using WebView based on WebView communication logs
The proposed mechanism saves the information described in Section 3.4 to the internal storage of Android for each Android app using WebView. In addition, the proposed mechanism acquires the information in a unique format considering ease of analysis and lightweight of preservation and saves the information into internal storage. This makes it possible to analyze the Android app that uses WebView by focusing on WebView communication logs every Android app using WebView. Additionally, the following case can be considered as the use case of gathered data:
- Analysis of the Android app that uses WebView
  We can analyze the Android app using WebView based on WebView communication logs. Thus, we can analyze whether WebView accesses malicious contents, and verify the threat due to accessing the contents.
- Detection of malicious communication and attacks
  We can analyze characteristics of malicious communication and attacks by using gathered data. Thus, we can detect the malicious communication and the attacks based on the analysis results.

Moreover, the proposed mechanism is designed inside WebView. The proposed mechanism can analyze the communication on inside WebView, and might intercept malicious communication. Thus, the proposed mechanism may prevent malicious JavaScript execution and attacks.

E3 Analyzing communication contents encrypted by TLS/SSL
HTTPS is a protocol that protects the HTTP of the application layer, which encrypts communication between the web browser or WebView and the web server by TLS/SSL, and prevents eavesdropping and tampering of communication contents. The proposed mechanism can acquire the HTTP request and HTTP response encrypted by TLS/SSL as a plain text. This is because

the proposed mechanism acquires the HTTP request before encrypted and the HTTP response after decrypted. Therefore, WebView executes the encryption processing and the decryption processing of the HTTP message.

4 IMPLEMENTATION AND EVALUATION

4.1 Implementation

We implemented the proposed web access monitoring mechanism on Chromium WebView 60.0.3094.2. To implement the proposed mechanism, we modified the following two classes of the C++ layer in WebView.

(1) HttpStreamParser class
(2) SocketPosix class

(1) The HttpStreamParser class generates the data in the format of HTTP/1.0 or HTTP/1.1, and performs web access request and response processing. However, the HttpStreamParser class cannot monitor web access using HTTP/2.0. Therefore, monitoring web access via WebView that uses HTTP/2.0 by the proposed mechanism is considered future work.

The proposed mechanism can acquire the HTTP message encrypted by TLS/SSL as a plain text. In WebView, the HTTP request is encrypted before sending it and the HTTP response is decrypted after receiving it. Moreover, in HttpStreamParser class, the unencrypted HTTP message can be acquired. Thus, we just added the processing in Table 1 to the HttpStreamParser class; that is, adding the processing to decrypt the HTTP message is not necessary.

Immediately after an HTTP request in the form of HTTP/1.0 or HTTP/1.1 is generated, the proposed mechanism acquires the time stamp at the time of generating the HTTP request header, the package name of the Android app, the HTTP request header, the URL, the IP address and the port number of the web server, and then saves these information in internal storage (processing (2) in Figure 5). Among this information, the URL, the IP address and the port number of the web server are acquired from instances of another class. Additionally, when sending data to the web server using the POST method, the HTTP request body is acquired before sending it and then is saved in the internal storage of Android (processing (3) in Figure 5).

The HTTP response header is acquired after completing reception of the HTTP response header and then is saved in internal storage (processing (4) in Figure 5). However, when acquiring the HTTP response header, it is necessary to calculate the offset of the HTTP response header. This is because when receiving the HTTP response header from the web server, the HTTP response header and part of the HTTP response body are received. The offset of the HTTP response header is calculated by the FindAndParseResponseHeaders() method.

When the size of the HTTP response body is large, it is transmitted from the Web server in sections. Therefore, after completing reception of each HTTP response body, the HTTP response body is acquired and then is saved in internal storage (processing (5) in Figure 5).

(2) SocketPosix class issues a system call for web access. The proposed mechanism acquires the return value of the connect() system call and then saves it in internal storage (processing (1) in Figure 5). However, an error (Operation now in progress) is returned immediately after issuing the connect() system call. This is because web access via WebView is performed asynchronously. Therefore, it is necessary to acquire the return value of the connect() system call again immediately after the connect processing is completed.

Table 2: Evaluation environment of Android Emulator

OS	Ubuntu 16.04 LTS
CPU	Intel(R) Xeon E5-2609V4 (8 cores)
Memory	64 GB
Kernel	Linux 4.4.0-92-generic (64 bit)
Android Emulator	Android 6.0

Table 3: Evaluation environment of an Android device (Nexus 6P)

OS	Android 6.0.1
CPU	Snapdragon 810 2.0 GHz (octa core)
Memory	3 GB

4.2 How to introduce WebView applying the proposed mechanism

The existing WebView is installed as an Android system app. When introducing the modified WebView to Android device, it is necessary to uninstall the existing WebView. Here, when uninstalling the Android system app, it is necessary to gain root access on Android. Gaining root access on Android allows users administrative privileges. Moreover, it is necessary to set the package name of WebView with the proposed mechanism to "com.google.android.webview". This is because the package name of the Android app to be used as WebView requires this name.

4.3 Evaluation

4.3.1 Content and Environment. In order to clarify the effectiveness and overhead of the proposed mechanism, this paper evaluates the following items.

(1) Experiment to test the operation of the proposed mechanism
Using Android Emulator, we compare the information acquired by the proposed mechanism and tcpdump. Then from the comparison results, we verify whether the proposed mechanism can acquire the information necessary for analysis. Additionally, this paper shows the effectiveness of the proposed mechanism based on the comparison results.
(2) Performance measurement of the proposed mechanism
We introduced WebView applying the proposed mechanism to Android device and measured the overhead.

The evaluation environment is shown in Table 2 and Table 3. For the evaluation, we used our own test app. This app uses WebView and displays the top page of Okayama University's web site.

4.3.2 Experiment to test the operation of the proposed mechanism. We evaluated whether the proposed mechanism can monitor web access via WebView by comparing the information acquired by the proposed mechanism and tcpdump. Additionally, based on the

Table 4: Comparison results of communication logs

	Proposed mechanism	tcpdump
Communication contents using HTTP	Acquired	Acquired
Communication contents using HTTPS	Acquired	Acquired (encrypted)
Time stamp	Acquired	Acquired
The package name of the Android app	Acquired	Not acquired
URL	Acquired	Acquired
IP address	Acquired	Acquired
Port number	Acquired	Acquired

comparison results, we validate the proposed mechanism. Note that this evaluation is performed using Android Emulator.

In this evaluation, we started only the test app and extracted communication logs of the test app which is collected by tcpdump. To extract communication logs of the test app, we retrieved the "X-Requested-With" header included in the HTTP request header of the test app by using Wireshark's search function.

Table 4 shows the comparison results of communication logs acquired by the proposed mechanism and tcpdump. "Acquired" in Table 4 means that the information could be acquired, and "Not acquired" means that the information could not be acquired. Additionally, although tcpdump can acquire the communication contents using HTTPS, the acquired information is encrypted (Acquired (encrypted) in table 4). From Table 4, the evaluation results show that the proposed mechanism operates as designed and can acquire communication logs via WebView.

We evaluated the number of requests in the test app. As the results, tcpdump could monitor 24 web access by HTTP and 21 access by HTTPS. The proposed mechanism could monitor the same number of times web access as tcpdump. Thus the proposed mechanism can acquire the information necessary for analysis described in Section 3.4 without omission. Additionally, tcpdump cannot analyze the communication contents encrypted by TLS/SSL. The proposed mechanism can acquire the HTTP request and HTTP response encrypted by TLS/SSL as a plain text. This is because the proposed mechanism acquires the HTTP request before encrypted and the HTTP response after decrypted.

In summary, the proposed mechanism has following advantage:

- The proposed mechanism can acquire the package name of the Android app. This makes it possible to specify and analyze the Android app which accesses web content via WebView.
- In the case of the acquired information is encrypted when analyzing the Android app using WebView, it is necessary to decrypt the contents. The proposed mechanism can acquire the HTTP message encrypted by HTTPS as a plain text. This makes it possible to analyze the communication contents via WebView without decryption.

4.3.3 Performance measurement of the proposed mechanism. To evaluate the performance of the proposed mechanism, we started the test app and we measured the processing overheads and the acquired data size per Request/Response of (2), (4), and (5) shown in Table 1.

Table 5: Average overheads and the acquired data size of the proposed mechanism per Request/Response

Processing	(2)	(4)	(5)
Overheads of the processing(unit: ms)	0.234	0.099	1.978
Data size of the information acquired by each processing (unit: KB)	0.62	0.35	225.28

Processing (2): This processing is executed immediately after generation of the HTTP request header.

Processing (4): This processing is executed after reception of the HTTP response header.

Processing (5): This processing is executed after reception of the HTTP response body.

Then we performed this process repeatedly five times, and calculated the average overhead results. This paper does not measure the processing (1) and (3) of Table 1 because they are not executed in the test app.

Table 5 shows the measurement results. From Table 5, the processing time of (5) is larger than the processing time of (2) and (4), and the processing time of (2) is larger than the processing time of (4). Additionally, the data size of the information acquired by processing (5) is larger than this one by processing (2) and (4). The data size of the information acquired by processing (2) is larger than this one by processing (4). Thus it can be inferred that each processing time depends on the data size of the information acquired in each processing operation.

It seems reasonable to suppose that the overhead of the processing (5) is larger than that of the other processing. The HTTP response body may be divided and transmitted multiple times when the data size is large. When the HTTP response body is transmitted in multiple times, on every reception of this information, the proposed mechanism executes processing (5). Thus, the number of executions of processing (5) may be greater than the number of executions of processing operations (2) and (4), and the processing time becomes longer.

The result that the overhead of the processing (2) is larger than that of the processing (4) is reasonable. The processing (2) acquires the time stamp, the Android app package name, URL, IP address and port number besides the HTTP request header and saves these information in the internal storage. The processing (4) just acquires and saves the HTTP response header. Therefore as the results in Table 5 show, it is evident that the data size of the information acquired by the processing (2) is larger than that of the processing (4). Moreover, the number of the method invocations to acquire the information of the processing (2) is greater than that of the processing (4). Thus, the process time of the processing (2) is longer than that of the processing (4).

In our future work, we will measure the overheads of other Android apps in Google Play.

5 CONCLUSION

In order to monitor web access via WebView, this paper proposed a web access monitoring mechanism for Android WebView. The proposed mechanism monitors all web access via WebView. We implemented the proposed mechanism on Chromium WebView version 60.0.3094.2, and evaluated this mechanism. Additionally, the proposed mechanism makes it possible to analyze the behavior of malware and malicious Android apps. Moreover, the proposed mechanism can acquire the communication contents encrypted by TLS/SSL as a plain text.

We experimented with the operation of the proposed mechanism on an Android app. This evaluation results show that the proposed mechanism can acquire the information necessary for analysis. Moreover, there are advantages that the proposed mechanism can acquire unencrypted HTTP message even with the communication using HTTPS and the package name of the Android app. In the performance evaluation, we measured the overhead and the acquired data size of the proposed mechanism per Request/Response. The results of performance evaluation infer that each processing time of the proposed mechanism depends on the data size of the information acquired by each processing operation. Moreover, the total overhead of each processing is approximately 2.3 ms, which is very short.

Gathering the communication logs via WebView and analysis of the Android apps that use WebView are our future work.

ACKNOWLEDGMENTS

The research results have been achieved by “WarpDrive: Web-based Attack Response with Practical and Deployable Research InitiatiVE”, the Commissioned Research of National Institute of Information and Communications Technology (NICT), JAPAN.

REFERENCES

[1] T. Luo, H. Hao, W. Du, Y. Wang, and H. Yin, Attacks on WebView in the Android system, In *Proceedings of the 27th Annual Computer Security Applications Conference*. ACM, pp. 343–352, 2011.
[2] P. Mutchler, A. Doupé, J. Mitchell, C. Kruegel, and G. Vigna, A Large-Scale Study of Mobile Web App Security, In *Proceedings of the Mobile Security Technologies Workshop (MoST)*, 2015.
[3] WebKit, Open Source Browser Engine. https://webkit.org/.
[4] The Chromium project, https://www.chromium.org/.
[5] The Chromium project, NetworkStack, https://www.chromium.org/developers/design-documents/network-stack/.
[6] G. S. Tuncay, S. Demetriou, and C. A. Gunter, Draco: A System for Uniform and Fine-grained Access Control for Web Code on Android, In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. ACM, pp. 104–115, 2016.
[7] M. Neugschwandtner, M. Lindorfer, and C. Platzer, A View to a Kill: WebView Exploitation, In *Proceeding of the 6th USENIX Workshop on Large-Scale Exploits and Emergent Threats (LEET)*, 2013.
[8] T. Luo, W. Du, and Y. Wang, ATTACKS AND COUNTERMEASURES FOR WEBVIEW ON MOBILE SYSTEMS, Ph.D. Dissertation. Syracuse University, 2014.
[9] A. B. Bhavani, Cross-site Scripting Attacks on Android WebView, *arXiv preprint arXiv:1304.7451*, 2013.
[10] W. Bao, W. Yao, M. Zong, and D. Wang, Cross-site Scripting Attacks on Android Hybrid Applications, In *Proceedings of the 2017 International Conference on Cryptography, Security and Privacy*. ACM, pp. 56–61, 2017.
[11] S. Son, D. Kim, and V. Shmatikov, What Mobile Ads Know About Mobile Users, In *Proceedings of the Network and Distributed System Security Symposium (NDSS 2016)*, 1–15, 2016.
[12] N. Kudo, T. Yamauchi, and T. H. Austin, Access Control for Plugins in Cordova-based Hybrid Applications, In *the 31st IEEE International Conference on Advanced Information Networking and Applications (AINA–2017)*, pp. 1063–1069, 2017.
[13] J. Yu and T. Yamauchi, Access Control to Prevent Malicious JavaScript Code Exploiting Vulnerabilities of WebView in Android OS, *IEICE Transactions on Information and Systems*, vol. E98-D, no. 4, pp. 807–811, 2015.