International Journal of Information Security (2021) 20:833–847
https://doi.org/10.1007/s10207-020-00534-3

REGULAR CONTRIBUTION

Web access monitoring mechanism via Android WebView for threat analysis

Yuta Imamura[1] · Rintaro Orito[1] · Hiroyuki Uekawa[1,2] · Kritsana Chaikaew[3,4] · Pattara Leelaprute[4] · Masaya Sato[1] · Toshihiro Yamauchi[1]

Published online: 19 January 2021

Abstract

Many Android apps employ WebView, a component that enables the display of web content in the apps without redirecting users to web browser apps. However, WebView might also be used for cyberattacks. Moreover, to the best of our knowledge, although some countermeasures based on access control have been reported for attacks exploiting WebView, no mechanism for monitoring web access via WebView has been proposed and no analysis results focusing on web access via WebView are available. In consideration of this limitation, we propose a web access monitoring mechanism for Android WebView to analyze web access via WebView and clarify attacks exploiting WebView. In this paper, we present the design and implementation of this mechanism by modifying Chromium WebView without any modifications to the Android framework or Linux kernel. The evaluation results of the performance achieved on introducing the proposed mechanism are also presented here. Moreover, the result of threat analysis of displaying a fake virus alert while browsing websites on Android is discussed to demonstrate the effectiveness of the proposed mechanism.

Keywords Android · WebView · Web access monitoring · Web security · Threat analysis · Fake virus alert

1 Introduction

Mobile devices (e.g., smartphones) have been widely used around the world for many years now. Android devices have held the biggest share in the global smartphone market since 2011 [1]. In addition, as mobile devices have become more popular, mobile web browsing has surpassed desktop browser use and the number of mobile malware cases has increased. Although it took 20 years to reach two million malware samples on the personal computer (PC) environment, it took only 5 years to reach the same number of samples on mobile devices [2]. The methods of infiltrating Android devices with malware include malvertising and scams [3]. Moreover, the Google Play Store can also be under attack, especially in the form of ad click frauds, which is the most common scam targeting users [3]. It is assumed that mobile malware authors have set their sights firmly on monetization. In addition, the attacks on mobile devices mostly use scamming strategies, whereas the attacks on PCs infect them with malware directly (i.e., drive-by-download). Thus, it is necessary to take preventive measures on mobile devices against these attacks.

Web browser apps or embedded browsers (e.g., WebView) are used for browsing web pages on Android devices. They are developed by recognized companies that can be trusted. Additionally, a conventional web browser app can use plugins and its own security function, indicating that it can protect web access via the conventional web browser app. On the other hand, the use of WebView depends on Android app developers, which differs from use of the web browser app.

✉ Pattara Leelaprute
pattara.l@ku.ac.th

✉ Toshihiro Yamauchi
yamauchi@cs.okayama-u.ac.jp

Masaya Sato
sato@cs.okayama-u.ac.jp

[1] Graduate School of Natural Science and Technology, Okayama University, 3-1-1 Tsushima-naka, Kita-ku, Okayama 700-8530, Japan

[2] NTT Secure Platform Laboratories, Tokyo, Japan

[3] Faculty of Engineering, Okayama University, Okayama, Japan

[4] Faculty of Engineering, Kasetsart University, 50 Ngamwongwan Rd., Ladyao, Chatuchak, Bangkok 10900, Thailand

Many Android apps use WebView to display webpages and advertisements inside the apps. In the Android app store managed by Google, WebView was used by approximately 86% of Android apps as of 2011 [4] and 85% of Android apps as of June 2014 [5]. The implementation of WebView depends on Android app developers, and most Android apps that use WebView are not developed by recognized companies; their trustworthiness is therefore not guaranteed [4]. Therefore, when Android app developers develop an app without considering the security, attackers may exploit the security vulnerabilities of the app to target innocent users. Therein lies the difference between web browsers and WebView.

Although WebView can use only Google Safe Browsing, this measure alone is not enough to protect web access via WebView, especially from fake virus alerts, which use malvertising to redirect the users to web pages and scam the users into installing the suspicious Android app. Moreover, previous studies have reported attacks exploiting WebView and presented countermeasures against these attacks [4–16]. Some studies have presented access control methods to prevent malicious JavaScript codes from exploiting the vulnerabilities of WebView [6–8] and to prevent app-repackage attacks in Cordova-based hybrid applications [9].

The following are some existing studies: OS-level mitigation against a new attack that exploits WebView instances between each Android app by a malicious JavaScript code [10] and automatic analysis methods focused on event handlers [11]. Previous studies have also developed mitigation efforts against an attack exploiting AdSDK [12]. The target of these countermeasures is the prevention of specific attacks except for malvertising and scams. Moreover, social media platforms are increasingly used by attackers to infect mobile devices through malware [17] and social media threats are surging [18]. As most social media apps (e.g., Facebook and Twitter) use WebView, users may experience damage by websites linked to social media apps.

The studies mentioned so far have proposed access control mechanisms for attacks exploiting WebView, mitigations against these attacks, and analysis of Android apps at the source code level. However, these studies did not focus on web access via WebView or propose measures against threats such as malvertising and scams for Android WebView.

To address these threats on WebView, we propose a web access monitoring mechanism for Android WebView (hereinafter referred to as "WebView Monitor") as a means of monitoring web access via WebView and describe its design and implementation. WebView Monitor can monitor all forms of web access with HTTP. Additionally, WebView Monitor can acquire the HTTP request and HTTP response, which is encrypted by SSL/TLS, as plain text, and the Android app package name to identify the Android app. Furthermore, this mechanism does not require any modification of the Android Framework or the Linux kernel, so that there is an advantage that it can be introduced by just replacing the built-in WebView implementation with or installing a modified version. Although rooting is required in Android 5 and Android 6, we consider that WebView Monitor is easier to use compared to other monitoring tools (i.e., MITM proxy) for Android communication. Thus, the target of WebView Monitor is not only researchers or security analysts but also ordinary civilians with little to no technical backgrounds who can understand the contents of communication logs. Therefore, WebView Monitor is useful in threat analysis of web access via WebView. Moreover, to show the effectiveness of proposed mechanism, we describe the results of a threat analysis for malvertising and scams via WebView using WebView Monitor. In particular, we investigated fake virus alerts. This paper also reports the evaluation results for WebView Monitor.

In summary, our study makes the following contributions:

- Monitoring method of web access via WebView WebView Monitor is a new method of monitoring all forms of HTTP-based web access via WebView. WebView Monitor does not require any modification of the Android system for installation and can therefore be easily installed on various versions of Android devices.
- Acquisition of plaintext contents in encrypted communication WebView Monitor can acquire the HTTP request and HTTP response, which is encrypted by SSL/TLS, as plain text. This is because WebView Monitor acquires the HTTP request before encryption and the HTTP response after decryption.
- Analysis of Android apps using WebView based on WebView communication logs WebView Monitor saves the information described in Sect. 3.4 to the internal storage of Android device for each Android app using WebView. In addition, WebView Monitor acquires the information in a unique format to ensure ease of analysis and preservation and saves the information in internal storage. This makes it possible to analyze the Android app that uses WebView by focusing on WebView communication logs for every Android app using WebView.
- Threat analysis of fake virus alerts using WebView Monitor We performed analysis based on the content of web access via WebView to prevent attacks using fake virus alerts. In particular, we revealed the mechanism of displaying a fake virus alert and redirection flow while browsing websites via WebView.

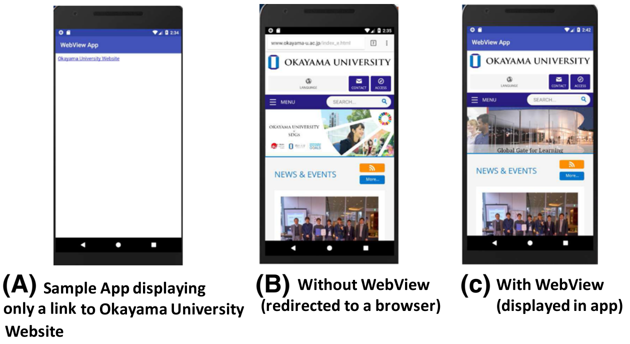
2 Background

2.1 WebView

WebView is a component that makes it possible to display web content on Android apps without redirecting the user to a web browser. Figure 1 shows a difference in page display behavior with the use of WebView. The Android app ((A) in Fig. 1) displays a link to the top page of Okayama University's website. When the app is not using WebView ((B) in Fig. 1), the user is redirected to a web browser, whereas when the app is using WebView ((C) in Fig. 1), the website is displayed within the Android app. Recently, numerous well-known and widely used Android apps have been utilizing WebView. Examples of such apps include social media apps (e.g., Twitter and Facebook), email apps (e.g., Gmail), and mapping apps (e.g., Uber). This indicates that WebView is an essential component of Android apps.

WebView has used different browser engines in each Android version. WebView implementation from Android 4.1 to Android 4.3 uses WebKit [19], whereas Android 4.4 onward, Chromium WebView is used [20]. WebView implementations up to Android 4.4 are contained in the Android Framework. However, after Android 5.0, it is separated from the Android Framework and is instead as the Android System WebView app. This allows us to update WebView on Google Play without having to update the Android version. Although only a single installation of WebView is allowed in Android 5 and Android 6, multiple installations of WebView are allowed from Android 7 onward. This enables us to switch WebView implementation in the Android device settings after an implementation is installed by the user.

WebView comes with a number of Application Programming Interfaces (or "APIs"). Using WebView, Android apps can easily embed a powerful browser inside. Moreover, this makes it possible not only to display web content but also to interact with web servers with a number of APIs. To display web content and interact with web servers, Android app developers often use the following APIs:



(A) Sample App displaying only a link to Okayama University Website

(B) Without WebView (redirected to a browser)

(c) With WebView (displayed in app)

Fig. 1 Difference in page display behaviors with the use of WebView

(1) `loadUrl` API
This API loads and displays a webpage within an Android app when given a URL string.

(2) `setJavaScriptEnabled` API
This API enables the execution of JavaScript downloaded within WebView.

(3) `addJavascriptInterface` API
WebView provides a mechanism for JavaScript code loaded within WebView to invoke the Android app's Java code. Android apps can register Java objects to WebView through this API. Moreover, this API makes it possible to invoke all the public methods in these Java objects from the JavaScript loaded within WebView.

2.2 Network stack of chromium WebView

Figure 2 (taken from [21]) 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., factors related to the implementation such as communication processing exhibit almost no differences among platforms. The implementation is shared and composed of a set of interfaces as shown in Fig. 2. Moreover, all web access by Chromium-based web browsers starts with the URLRequest interface.

Additionally, Chromium WebView has been developed by the Chromium project. Therefore, web access by Chromium WebView similarly starts with the URLRequest interface. Chromium WebView is developed using Java and C++, and it consists of the Java layer and the C++ layer. The C++ layer of 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 workflow for web access via WebView, and the following describes each step in the figure.

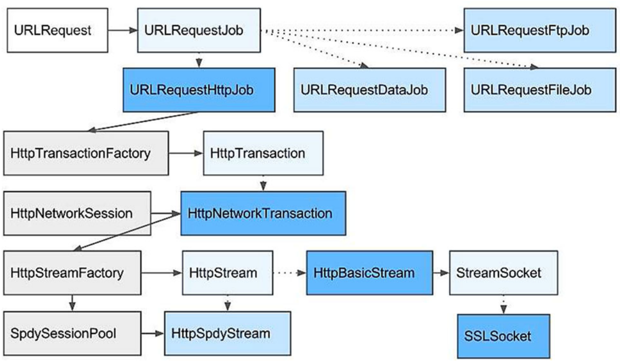


Fig. 2 Network stack of a Chromium WebView [21]

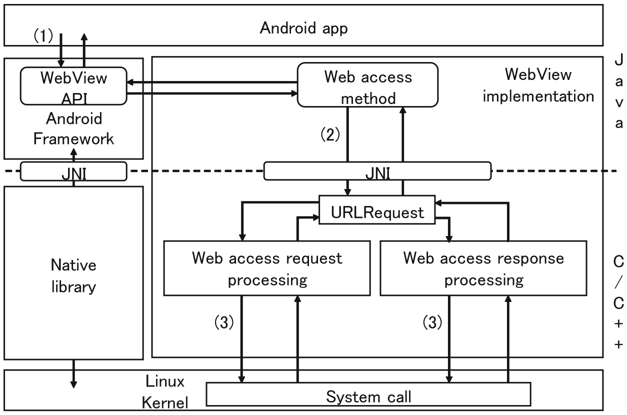


Fig. 3 Process of web access via WebView

(1) Calling APIs for web access
An Android app that uses WebView calls an API (or Java method) for web access. Here, the methods `loadUrl()`, `loadData()`, `loadDataWithBaseURL()`, and `postURL()` are used for web access [8]. In addition, the processing of these Java methods is delegated to the WebView implementation from the Android Framework.
(2) Calling C++ method via JNI
The Java method called in step (1) invokes the corresponding C++ method (via JNI), which manipulates the URLRequest interface. 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 of the WebView implementation.
(3) Issuing system calls
For TCP/UDP communication with a web server, some system calls are issued during the processing of the C++ layer in the WebView implementation.

2.4 Security issue

The use of WebView depends on the Android app developer, which is different from that of a web browser app (e.g., Chrome and Firefox). As mentioned in Sect. 1, most Android apps using WebView are not developed by recognized companies, and their trustworthiness is thus not guaranteed. Therefore, when Android app developers create an Android app without considering app security, attackers can exploit the security vulnerabilities.

There are four ways in which attackers have been known to infiltrate Android devices with malware;
malvertising and scams constitute some of these methods [3]. Malvertising is the practice of inserting malware into legitimate online ad networks to target a broad spectrum of end users. Scams are common tools used by attackers to infect mobile devices with malware, and they rely on a user being redirected to a malicious web page, either through a web redirect or a pop-up screen. Moreover, social media are used by the attackers to infect mobile devices with malware [17]. WebView is used to display ads on most Android apps. Additionally, many social media apps (e.g., Facebook and Twitter) use WebView, which indicates that mobile devices may be infected with malware via WebView when users click on a link to the malicious website. Although web browsers can prevent web access to malicious websites and their malicious ads by security functions and ad blocker extensions (e.g., Adblock), WebView cannot use these security functions except for Google Safe Browsing, which is not enough to protect mobile devices from malware infection.

To the best of our knowledge, although some countermeasures based on access control for the attacks exploiting WebView have been reported, they cannot mitigate threats such as malvertising and scams, which are extensively used in mobile devices by attackers. To address these threats on WebView, analyzing web access via WebView and considering countermeasures based on the analysis results are necessary. However, no previous studies have reported results on the threat analysis of WebView focusing on web access via WebView itself and no web access monitoring mechanism for WebView has been reported. Web access on Android can be monitored using a HTTP proxy or a packet capture tool. However, these tools cannot gather the communication log of WebView because they monitor all of the web access on Android devices, so they cannot distinguish access between WebView and others. Moreover, these methods cannot analyze the communication content encrypted by SSL/TLS, either. Thus, to address the above problems, a web access monitoring mechanism that can distinguish WebView communication from others and acquire plaintext content in encrypted communication is necessary.

3 WebView Monitor

3.1 Purpose and concept

This paper proposes a web access monitoring mechanism for Android WebView as a means of monitoring web access via WebView. WebView Monitor focuses on web access via WebView exclusive of other web access mechanisms available on Android and aims to monitor web access via WebView.

To realize WebView Monitor, it is necessary 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:

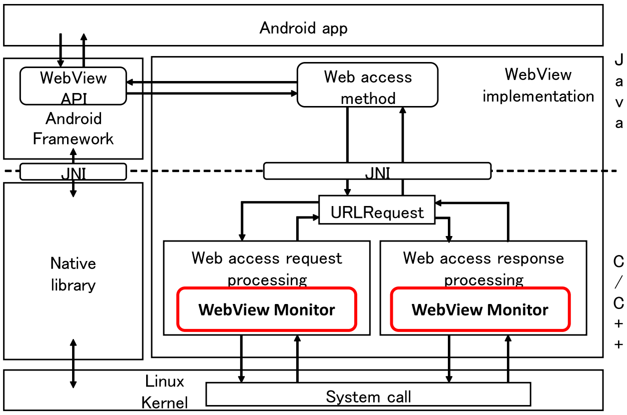


Fig. 4 Overview of WebView Monitor

(a) Android Framework
(b) WebView
(c) Linux kernel

The aims of this work were to analyze Android apps using WebView and detect and prevent the malicious web access inside WebView with information acquired intrinsically based on analysis results. To achieve these goals, we introduce WebView Monitor into numerous Android devices, and needed to collect extensive data. Therefore, we needed to consider the ease of introducing WebView Monitor to an Android device.

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 WebView Monitor by simply substituting a custom WebView implementation for the built-in one. For Android 6, root access is required to install our implementation with WebView Monitor. Based on the above considerations, we implemented the monitoring function to Chromium WebView which is one of the WebView implementations.

3.2 Design

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

3.3 Challenges

To implement WebView Monitor, the following challenges must be considered.

C1 Information to be acquired
The purpose of this work was to analyze Android apps using WebView based on web accesses via WebView. Therefore, we consider the information that WebView Monitor needs to acquire to analyze Android apps using WebView.

C2 Storage format and location of the acquired information
To analyze information acquired using WebView Monitor, we need to consider the storage format and location of the information considered in C1.

C3 Acquiring communication content communicated by SSL/TLS as plaintext
To make it possible to analyze web access via WebView to enable the use of detailed information for analysis, WebView Monitor needs to acquire plaintext contents communicated by SSL/TLS. Therefore, it is necessary to consider how to gain a HTTP message communicated by SSL/TLS as plain text.

C4 Monitoring all forms of web access with HTTP
To monitor all web access via WebView which use HTTP and SPDY protocols, it is necessary to consider an interface at which WebView Monitor can monitor all web access.

3.4 Information to be acquired

WebView Monitor acquires the following information for analysis of Android apps using WebView.

(i) Information on web access content

- HTTP request and HTTP response
Transfer of the data between the web browser and the web server uses HTTP. Additionally, the data have to be acquired as it contains a considerable amount of information regarding web access. Therefore, it is possible to acquire the data of the communication content of web access via WebView in HTTP format. In view of the above, the HTTP request and HTTP response should be acquired.

(ii) Information on web access destination

- URL
URLs are vital in threat analysis and should be acquired because web accesses are based on URLs. In addition, the URL information enables us to use URL-based blacklists for analysis. Practically, the

URL of the access destination can be reconstructed from the information included in the HTTP request header. Therefore, WebView Monitor acquires and stores the URL of the web access as one item of request information.

- IP address
  The IP address of the communication destination as request information should be acquired. Some attack sites vary its IP address or domain name within a short period of time to bypass blacklists. Therefore, during analysis, it is difficult to identify the attack site based only on HTTP request and response, as there is a high possibility that the DNS registration information of the domain of the attack site has already changed. To enable the easy identification of the true attack site, WebView Monitor acquires the IP address at the time of communication.
- Port number of the web server
  For TCP and UDP in the network layer, port numbers are used as identifiers for designating the end points of inter-host communication. In web access using HTTP, usually, 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.

(iii) Information on Android app

- Android app package name When analyzing an Android app that uses WebView, it is necessary to specify the Android apps that accesses the web content via WebView. Therefore, to identify the Android app, the package name of the Android app should be acquired.

(iv) Other information

- Time stamp
  An HTTP request and a corresponding response are acquired independently because the processes of sending the request and receiving the response are independent of one another. For this reason, it is impossible to determine the correspondence between the two aforementioned processes. In WebView, an object, which handles a pair of HTTP request and response, is created for each web access. Therefore, it is necessary to obtain a time stamp (e.g. at the creation of the object) as an identifier that can distinguish each of the objects.

3.5 Storage format and location of the acquired information

As described in Sect. 3.4, HTTP requests and responses are acquired independently. However, if requests and responses are stored separately, finding request/response pairs in communication logs will be challenging. Therefore, it is necessary to store requests and responses as pairs for the ease of analysis. WebView Monitor uses the time stamp described in Sect. 3.4 as the file name for each of request/response pairs. In addition, it stores the acquired information in the internal storage in a simple format considering the lightweightness of processing. In preparation for analysis, the communication logs stored in the device storage are collected on the analyst's computer and converted to JSON format for the ease of analysis. Furthermore, WebView Monitor stores communication logs in a separate data area for each Android app. Consequently, communication logs can be collected separately for each Android app.

3.6 Acquisition of plaintext content communicated by SSL/TLS

Web access that uses HTTPS and SPDY is encrypted by SSL/TLS, which prevents eavesdropping and tampering of communication content. In the case the acquired information is encrypted when analyzing, it is necessary to decrypt the content. For the ease of analysis, WebView Monitor needs to acquire HTTP messages as plaintext even when SSL/TLS is used. Thus, WebView Monitor acquires the HTTP request before encryption and the HTTP response after decryption. This makes it possible to analyze such access without decryption.

3.7 Monitoring all forms of web access with HTTP

WebView uses HTTP and SPDY protocols when accessing websites. To analyze web access via WebView in detail, it is necessary for WebView Monitor to be able to monitor access using HTTP and SPDY protocols. The following can be considered as a way to monitor the access:

(1) Adding the monitoring function to an interface that uses both HTTP and SPDY protocols
(2) Adding the monitoring function to each interface that uses HTTP and SPDY protocols

There are some interfaces (e.g., URLRequest and HttpNetworkTransaction in Fig. 2) to always go through for WebView web access. Therefore, adding a monitoring function to one of these interfaces enables all forms of web access with HTTP to be monitored. However, only the URL can be acquired at the URLRequest interface, and the HTTP request body cannot be acquired at the HttpNetworkTransaction interface. Moreover, although all web access can be monitored when invoking a system call, the communication content using HTTPS and SPDY is encrypted by SSL/TLS.

In contrast, by adding the monitoring function to each interface that uses HTTP and SPDY protocols, all information shown in Sect. 3.4 can be acquired and communication content using HTTPS and SPDY can be acquired as plain text. Thus, we implemented WebView Monitor functions to each interface (e.g., HttpStreamParser and SpdyHttpStream) that uses HTTP and SPDY. This makes it possible to monitor all forms of web access with HTTP regardless of the protocol and to analyze web access via WebView in detail.

3.8 Flow of WebView Monitor

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

(1) When establishing a connection by the `connect()` system call fails, WebView Monitor acquires a connection error at the socket connection and saves it in the internal storage of Android device.

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

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

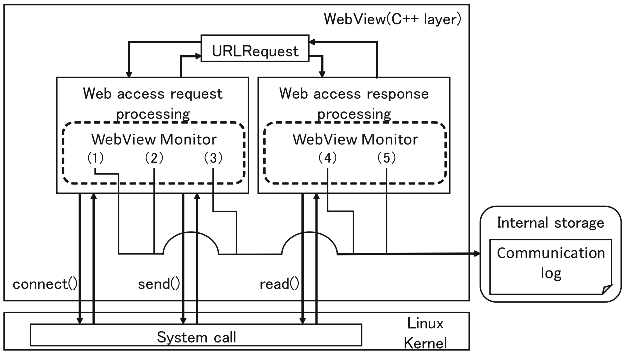


Fig. 5 Processing flow of WebView applying WebView Monitor

(4) After receiving the HTTP response header through the `read()` system call, WebView Monitor acquires the HTTP response header and saves it in internal storage.

(5) After receiving the HTTP response body through the `read()` system call, WebView Monitor acquires the HTTP response body and saves it in internal storage.

4 Implementation and evaluation

4.1 Implementation

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

(1) HttpStreamParser class
The HttpStreamParser class performs the web access request and response processing of the HTTP protocol. The number of steps in the modification portion is 163 lines.

(2) SpdyHttpStream class
The SpdyHttpStream class performs the web access request and response processing of the SPDY protocol. The number of steps in the modification portion is 147 lines.

WebView Monitor can acquire HTTP messages encrypted by SSL/TLS as plain text. In WebView, the HTTP request is encrypted before sending it and the HTTP response is decrypted after receiving it. Specifically, in the HttpStreamParser class and SpdyHttpStream class, the unencrypted HTTP message can be acquired regardless of the encryption of the message.

Thus, we simply added the processing mentioned in Table 1 to the HttpStreamParser class and the SpdyHttpStream class; in other words, adding the processing to decrypt the HTTP message is not necessary. Immediately after an HTTP request is generated, WebView monitor acquires the following information and stores them in the internal storage: 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 (processing (2) in Fig. 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 is then saved in the internal storage of the Android device (processing (3) in Fig. 5). The HTTP response header is acquired after completing reception of

Table 1 Processing of WebView Monitor

WebView Monitor	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	Immediately after generation of the HTTP request header
		Android application package name	
		HTTP request header	
		URL	
		IP address	
		Port number of the web server	
	(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

the HTTP response header and is then saved in internal storage (processing (4) in Fig. 5). 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 saved in internal storage (processing (5) in Fig. 5).

4.2 How to install our WebView implementation with WebView monitor

A built-in WebView implementation is installed as a system app. From Android 7 onward, it is done by simply installing it as a user app and switching WebView implementation in the Android device setting after installation. However, in Android 5 and Android 6, it is necessary to uninstall the built-in one. Here, when uninstalling a system app, it is necessary to gain root access on Android device. Gaining root access on Android device provides users with administrative privileges. Moreover, it is necessary to set the package name of our WebView implementation to the same name of the built-in one: "com.google.android.webview". This is because this name is hard-coded as the name of the only system WebView app required by the Android Framework.

4.3 Evaluation

To clarify the effectiveness and overhead of WebView Monitor, we evaluated the following items.

(1) Experiment to test the operation of WebView Monitor
Using Android Emulator, we compare the information acquired by WebView Monitor and tcpdump. Then, based on the comparison results, we verified whether WebView Monitor can acquire the information necessary for analysis. Additionally, we show the effectiveness of WebView Monitor based on the comparison results. Although we use the Android Emulator in this functionality evaluation to simplify the procedure, the results of the evaluation are meaningful because they are not dependent on the devices' characteristics.

(2) Performance measurement of WebView Monitor
We install our WebView app that includes the WebView Monitor to Android device and measured the overhead.

(3) Evaluation of communication log size
We measured the data size of the communication logs stored in the device storage, the number of created HTTP request/response, along with the content type and content length (included in HTTP response header).

Table 2 Host environment for 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

Model	Nexus 6P
OS	Android 6.0.1
CPU	Snapdragon 810 2.0 GHz (octa core)
Memory	3 GB

The evaluation environment is shown in Tables 2 and 3. The evaluation used a sample app developed by us. This app uses WebView and displays the top page of Okayama University's website.

Table 4 Comparison of results of communication logs

Number		WebView Monitor	tcpdump
1	Number of times web access was monitored	101	101
2	Communication content using HTTP	Acquired	Acquired
3	Communication content using HTTPS	Acquired (plaintext)	Acquired (encrypted)
4	Communication content using SPDY	Acquired (plaintext)	Acquired (encrypted)
5	Time stamp	Acquired	Acquired
6	Package name of the Android app	Acquired	Not acquired
7	URL	Acquired	Acquired
8	IP address	Acquired	Acquired
9	Port number	Acquired	Acquired

4.3.1 Experiment to test the operation of WebView monitor

We evaluated whether WebView Monitor can monitor web access via WebView by comparing the information acquired by WebView Monitor and tcpdump. Additionally, based on the comparison results, we validated WebView Monitor.

In this evaluation of tcpdump, we started only the test app and extracted its communication logs using 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 using Wireshark's search function.

Table 4 shows a comparison of the communication logs acquired (or not acquired) by WebView Monitor, and tcpdump.

"Acquired" in Table 4 means that the information has been acquired, and "Not acquired" means that the information could not be acquired.

Additionally, although tcpdump can acquire communication content using HTTPS and SPDY, the acquired information is encrypted [Acquired (encrypted) in Table 4]. From Table 4, the evaluation results show that WebView Monitor operates as designed and can acquire communication logs via WebView.

Based on the results of the above-mentioned experimental test, WebView Monitor has the following advantages.

(1) WebView Monitor can acquire the HTTP request and HTTP response encrypted by SSL/TLS as plain text; this is because it acquires the HTTP request before encryption and the HTTP response after decryption by WebView itself. This makes it possible to analyze the content of web access via WebView without decryption.

(2) WebView Monitor can monitor web access via WebView, whereas HTTP proxies or packet capture tools cannot gather only the communication logs of WebView because these tools monitor all of the web access on Android devices. In addition, it is difficult to distinguish whether web accesses are via WebView or not.

(3) WebView Monitor can acquire the Android application package names. This makes it possible for the monitor to specify and analyze the application that accesses web content via WebView. This in turn enables the monitoring of web access via WebView.

4.3.2 Performance measurement of WebView monitor

To evaluate the performance of WebView Monitor, we launched the test app described in Sect. 4.3 and measured the processing overheads and acquired data size per request/response for each of processing (2), (4), and (5) shown in Table 1.

Processing (2): Processing is executed immediately after generation of the HTTP request header.
Processing (4): Processing is executed after reception of the HTTP response header.
Processing (5): Processing is executed after reception of the HTTP response body.

We repeated this process five times and calculated the average overhead results. We did not measure processing (1) and (3) in Table 1 because they are not executed in the test app. The evaluation environment was the same as the contents that described in Sect. 4.3.

Tables 5 and 6 show the measurement results. The processing time of (5) is larger than the processing times of (2) and (4), and the data size of the information acquired by processing (5) is larger than the data sizes for processing (2) and (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 processing (5) is larger than that of the others. The HTTP response body may be divided and transmitted multiple times when the data size is large. When the HTTP response body is transmitted multiple times, on every reception of this infor-

Table 5 Average overheads and acquired data sizes for WebView Monitor per request/response (Communication using HTTP)

Processing	(2)	(4)	(5)
Processing overheads (unit: ms)	0.238	0.119	0.434
Data size of information acquired by each processing (unit: KB)	0.66	0.37	4.44

Table 6 Average overheads and acquired data sizes for WebView Monitor per request/response (Communication using SPDY)

Processing	(2)	(4)	(5)
Processing overheads (unit: ms)	0.342	0.222	1.187
Data size of information acquired by each processing (unit: KB)	0.71	0.60	42.48

Table 7 Example of the data size from the communication logs

Content type	Number of HTTP request/response	Content length (unit: bytes)	The data size stored in the device storage (unit: bytes)
text/html	3	41,198	
text/css	18	120,884	
text/javascript	6	543,468	
text/plain	1	2	
application/javascript	7	193,262	
application/json	2	168	
image/svg+xml	22	48,157	
image/jpeg	33	1,723,119	
image/png	9	69,810	
image/gif	1	66	
font/woff2	1	9,132	
font/ttf	1	14,353	
Total	104	2,763,619	2,850,863

mation, WebView Monitor 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 processing (2) is larger than that of processing (4) is reasonable. Processing (2) acquires the time stamp, the Android app package name, URL, IP address, and port number besides the HTTP request header and saves this information in the internal storage. Processing (4) simply acquires and saves the HTTP response header. Therefore, as the results in Tables 5 and 6 show, it is evident that the data size of the information acquired by processing (2) is larger than that of processing (4).

In addition, the total overhead of processing (2), (4), and (5) is 0.697 ms (Communication using HTTP) and 1.751 ms (Communication using SPDY), which are very short.

4.3.3 Evaluation of communication log size

Table 7 shows the data size of the communication logs stored by WebView Monitor in the device storage when accessing the top page of Okayama University's website, to confirm the data size increase in relation to the communication data. Table 7 also shows the number of created HTTP request/response when accessing the website, content type, and content length. WebView Monitor saves HTTP response header and HTTP response header information in text format in addition to HTTP response body. Therefore, compared to the communication data, the data size does not increase much.

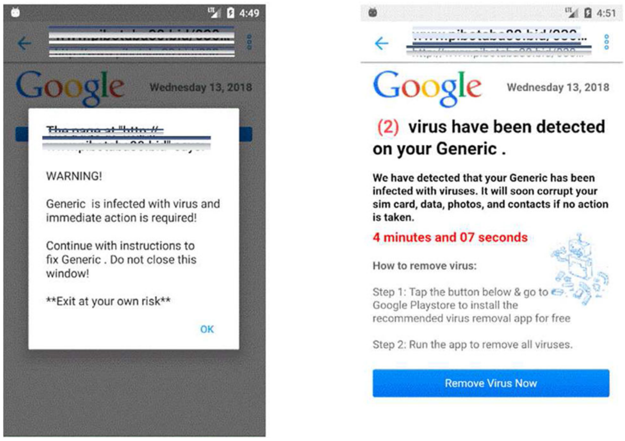
5 Threat analysis of fake virus alert

5.1 Purpose

The purpose of threat analysis fake virus alert is to evaluate the threat of web access via WebView in order to show the effectiveness of WebView Monitor against the threat of web access.

As mentioned in Sect. 2.4, malvertising and scams constitute some ways in which attackers infiltrate Android devices with malware, and a representative attack by these methods includes a fake virus alert.

WebView is used to display ads on most Android apps and some websites on many social media apps. Therefore,



4:49

[redacted]

Google Wednesday 13, 2018

[redacted]

WARNING!

Generic is infected with virus and immediate action is required!

Continue with instructions to fix Generic . Do not close this window!

Exit at your own risk

OK

4:51

[redacted]

Google Wednesday 13, 2018

(2) virus have been detected on your Generic .

We have detected that your Generic has been infected with viruses. It will soon corrupt your sim card, data, photos, and contacts if no action is taken.

4 minutes and 07 seconds

How to remove virus:

Step 1: Tap the button below & go to Google Playstore to install the recommended virus removal app for free

Step 2: Run the app to remove all viruses.

Remove Virus Now

Fig. 6 Examples of fake virus alerts

mobile devices may be infected with malware via WebView when users click on a link to a malicious website. WebView cannot use security functions and Adblock in the same way of web browser; it can only use Google Safe Browsing. Thus, it is necessary to analyze the web access via WebView and consider countermeasures for malicious attacks.

5.2 Threat analysis method of fake virus alert

Ad click fraud is the most common scam targeting users of the Google Play Store [2]. In this study, we analyzed the following threats:

- Fake virus alert:
  Several redirections may occur with a fake virus alert, which attempts to make the user install a suspicious app, which suddenly appears while the user visits a website. Attackers use malvertising to redirect users to websites that display the fake virus alert and scam the users into installing the suspicious Android app. Figure 6 shows examples of fake virus alerts, including the Google logo and claims that the Android device has a virus that must be removed immediately. Some fake virus alerts cause Android devices to vibrate and make sounds. Once this fake virus alert is displayed, the user cannot go to any other web page even if they click on the back button or any place on the screen. This makes the user believe that the Android device is infected with malware, and the user may click on the suggested button to remove the virus and install the suspicious Android app. To better understand the threat, we analyzed several websites that display these fake virus alerts. There are three ways to display a fake virus alert (malicious ads, fake app install screen, and posting messages on social media). In Sect. 5.3, we report on the analysis on fake virus alerts that use malicious ads; specifically,

  (i) redirection to a fake virus alert,
  (ii) redirected website,
  (iii) displaying a fake virus alert.

  The results are the same for each method of displaying the fake virus alert.

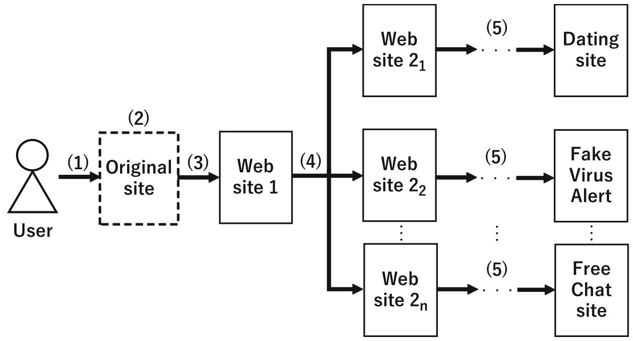


Fig. 7 Redirection flow of a fake virus alert

Here, we analyzed communication logs of web accesses with WebView Monitor. The targets of the threat analysis were 10 websites found either by web search or from Facebook and Twitter posts. These websites comprise media sites, free video sites, and free cartoon sites. When the user taps anywhere on the screen displaying these websites, a redirection occurs to the website displaying the fake virus alert. Additionally, we used Facebook and Twitter apps to gather communication logs via WebView because these apps are used extensively and may be used for attacks. The analysis environment is the Android emulator (Android 6.0) and WebView (60.0.3094.2).

5.3 Threat analysis result of fake virus alert

5.3.1 Redirection to the fake virus alert

Figure 7 shows the flow of redirection to a fake virus alert; the steps are described below.

(1) Visiting the original website of redirection

(2) Tapping anywhere on the screen
The user taps anywhere on the original website of redirection.

(3) Redirecting to website 1
The original website of redirection detects the user's tapping and redirects the user to website 1. Additionally, the user is forced to move to website 1 regardless of where they tap on the original website of redirection.

(4) Redirecting to website 2
Website 1 redirects the user to website 2. This redirection uses JavaScript code "window.location.replace." This JavaScript code can redirect the user to the speci-

fied URL without leaving a trace in the browser's history. This makes it impossible for the user to move to the previous website even if they click the back button.

(5) Redirecting to the website that displays the fake virus alert
Website 2 redirects the user to the website that displays the fake virus alert.

The redirects used in (3) and (5) are caused by various redirect codes of HTML, PHP, and JavaScript. In addition, the redirection code used here is decided by the original site shown in Fig. 7.

5.3.2 Redirected website

The redirection occurs multiple times while landing on the fake virus alert. Furthermore, we found that some redirected websites use SPDY protocols. Therefore, WebView Monitor can completely acquire all communication logs of redirections to fake virus alerts.

The URL of website 2 is generated by executing the JavaScript code of website 1 by either of the following two operations.

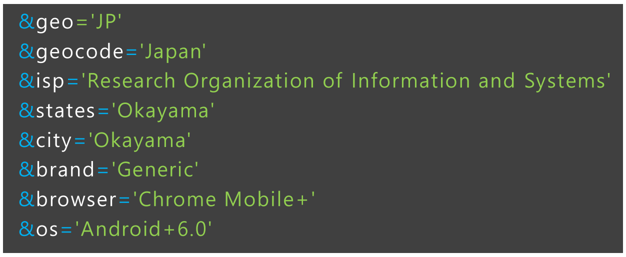
(1) The JavaScript code of website 1 creates 10 URLs that are a combination of a specified URL with Base 36 strings created at random. Moreover, the redirecting destination from website 1 creates a random number and uses it to select the redirecting website from 10 URLs.
(2) The JavaScript code of website 1 creates one URL, and the number that are part of this generated URL increase by one each time the user accesses website 1.

Therefore, redirecting to different websites on each access occurs with high probability. We suppose that this is a countermeasure against URL blacklists of security tools.

As shown in Fig. 7, website 2 redirects the user to various websites, such as dating websites and free live-chat websites. Moreover, the redirection destination and number of redirections on such a redirection are not the same every time. In fact, we observed that in some cases, the redirection to the fake virus alert does not occur.

On the original site, attackers set one of the following three conditions to make redirection occur when the user taps anywhere on the original website.

(1) Web browsers and WebView do not store the cookie of the original site.
(2) The stated time set by attackers for each original site has passed since the original site was displayed.
(3) The user has not tapped on the original site since this site was displayed.



```
&geo='JP'
&geocode='Japan'
&isp='Research Organization of Information and Systems'
&states='Okayama'
&city='Okayama'
&brand='Generic'
&browser='Chrome Mobile+'
&os='Android+6.0'
```

Fig. 8 Information used for displaying a fake virus alert

5.3.3 Displaying the fake virus alert

The analysis of the communication logs gathered using WebView Monitor revealed a JavaScript code that acquires the user information, such as the OS information, browser version, Java version, and Flash version. This JavaScript code acquires the user information shown in Fig. 8 (e.g., geocode, states, city, and OS information) from the user–agent contained in the HTTP request header.

We also detected the JavaScript file that displays the fake virus alert. This JavaScript file creates the fake virus alert based on acquired information. Moreover, this JavaScript file contains the code to make the user device vibrate and uses the countdown timer. This information and the method described in Sect. 5.3.1 are used to display the fake virus alert specific to that user. This can make the user believe that the displayed virus alert is legitimate. Furthermore, there is a button on the displayed virus alert that prompts the user to install the malicious Android app. We also confirmed that the language on the fake virus alert is based on the language set by the user on the Android device. In this experiment, we confirmed that the fake virus alert is displayed in English, Japanese, or Chinese.

5.4 Effectiveness of WebView monitor

We clarified the characteristics of the fake virus alert implementing threat analysis using WebView Monitor. From this analysis, we show the following effectiveness of WebView Monitor.

(1) WebView Monitor can acquire the HTTP request and HTTP response communicated by SSL/TLS as a plain text. Thus, we can analyze communication content communicated by SSL/TLS.
(2) 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.
(3) We can analyze characteristics of malicious communication and attacks by using the gathered data. Moreover, WebView Monitor is designed inside WebView. Thus,

we can detect the malicious communication and attacks based on the analysis results.

(4) WebView Monitor acquires the information in a unique format considering ease of analysis and lightweight preservation and saves the information in internal storage. Thus, we can identify the communication content for each Android application. In addition, analysis is easy because the information is stored in a unique format.

6 Related work

Exploiting Vulnerability of WebView WebView provides a mechanism for JavaScript code loaded within WebView to invoke the Android app's Java code. The API used for this mechanism is called `addJavascriptInterface` API. Android apps can register Java objects to WebView through this API. Moreover, this API makes it possible to invoke all the public methods in these Java objects from the JavaScript loaded within WebView. However, some attacks exploiting this API have been reported [4,6–8,13,14]. To address these attacks and improve the security of WebView, some previous studies have proposed some access control mechanisms.

Jin et al. [6] have proposed a fine-grained access control for hybrid apps so as to control access to the device resource from the JavaScript code. Moreover, Android app developers can set access permissions by specifying a resource's URL. Yu et al. [7] have proposed an access control mechanism that can control the access to security-sensitive APIs in Android from the JavaScript code by controlling the registration of Java objects through `addJavascriptInterface` API. Draco [8] can control access to device resources from web content, which are of different web origins, so as to prevent the attack that exploits the `addJavascriptInterface` API. Moreover, Android developers can define policies to specify the desired access characteristics of web origins in a fine-grained fashion.

App-repackage attack Kudo et al. [9] have presented a novel app-repackaging attack that repackages Cordova apps with malicious code and have proposed a novel runtime access control mechanism that restricts access based on the mobile user's judgment, to present a novel app-repackaging attack that repackages hybrid apps with malicious code.

Malicious JavaScript Attacks due to malicious JavaScript code loaded within WebView have been reported [10,11]. Li et al. [10] have reported a new attack exploiting WebView instances by malicious JavaScript code and proposed an OS-level mitigation as a countermeasure for this attack. Yang et al. [11] have proposed EOEDroid, which automatically vets event handlers in a given hybrid app using selective symbolic execution and static analysis.

Cross Site Scripting Attack on WebView WebView can execute JavaScript as well as web browser apps. Therefore, a cross-site scripting attack, which is an attack exploiting vulnerabilities in a web application, is also effective for WebView, and this attack on a Hybrid app is more powerful than on a web application [14–16]. However, these previous studies have not presented countermeasures to attacks.

Exploiting AdSDK Son et al. [12] analyzed mobile ad libraries, which use WebView, and reported an attack exploiting them. They also proposed methods of mitigation to employ against this attack.

Difference from Our Study Related works do not focus on the communication content of web access via WebView. This current study is focusing on communication via WebView, and the purpose of this study is to analyze Android apps using WebView based on web accesses via WebView and clarify characteristics of malicious communication from the analysis result. Based on the analysis result, it may be possible to detect and prevent malicious communication at the endpoint with acquired information.

7 Conclusion

To monitor web access via WebView, we proposed a web access monitoring mechanism for Android WebView. The proposed mechanism can monitor all forms of web access with HTTP and makes it possible to analyze web access via WebView in detail. We implemented WebView Monitor on Chromium WebView version 60.0.3094.2 and evaluated this mechanism. WebView Monitor makes it possible to analyze the behavior of malware and malicious Android apps. Additionally, WebView Monitor can acquire the communication content encrypted by SSL/TLS as plain text and Android app package name as information to identify the Android app. Furthermore, WebView Monitor can detect and prevent suspicious web access via WebView with information acquired intrinsically.

This paper reported the evaluation results of WebView Monitor. Experimental operation of WebView Monitor shows that WebView Monitor can acquire the information necessary for analysis. Moreover, there is the advantage that WebView Monitor can acquire unencrypted HTTP message even with the communication using HTTPS and the package name of the Android app. In the performance evaluation, the evaluation results suggested that each processing time of WebView Monitor depends on the data size of the information acquired by each processing operation. Moreover, the total overhead of each processing is very short.

Furthermore, this paper reports the results of threat analysis of displaying a fake virus alert while browsing websites on Android to show the effectiveness of the proposed mechanism. The results of the threat analysis of fake virus alerts using WebView Monitor reveal the mechanism of redirecting to malicious websites. We found that three redirections occur before landing on a fake virus alert website. In addition, we analyzed how to redirect to such websites and the malicious activities. Finally, we showed a mechanism of displaying fake virus alerts in web access via WebView. From the results of threat analysis of fake virus alerts, we showed the effectiveness of WebView Monitor in analyzing web access via WebView in detail.

In our future work, we will support the latest version of Android and confirm that our WebView implementation can be installed as a user app on devices running Android 7 onward.

Acknowledgements 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.

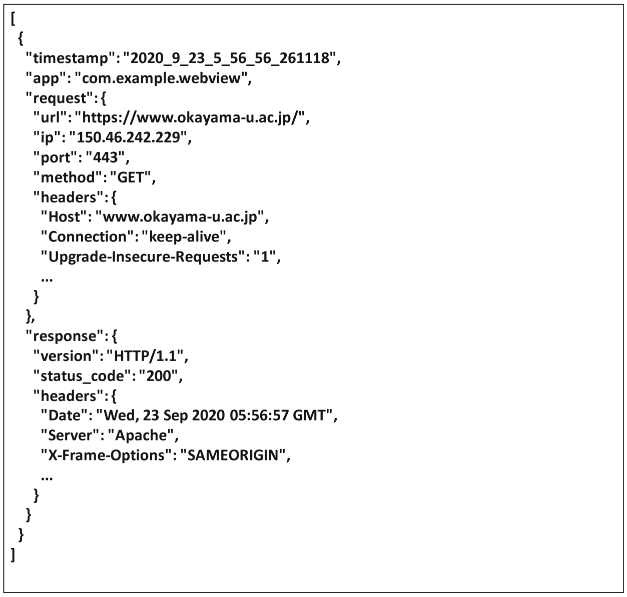
Funding Toshihiro Yamauchi has received research grants from National Institute of Information and Communications Technology (NICT), Japan, Japan Science and Technology Agency, and SECOM CO., LTD., Japan. He is a visiting scholar of Advanced Telecommunications Research Institute International (ATR), Japan.

Compliance with ethical standards

Conflict of interest The authors declare that they have no conflicts of interest.

A communication log

Figure 9 shows a communication log when accessing the top page of Okayama University’s website. WebView Monitor stores the acquired information in the internal storage in a simple format. After that, the stored communication logs are converted to JSON format, as shown in Fig. 9, for the ease of analysis.



```
[
 {
  "timestamp": "2020_9_23_5_56_56_261118",
  "app": "com.example.webview",
  "request": {
   "url": "https://www.okayama-u.ac.jp/",
   "ip": "150.46.242.229",
   "port": "443",
   "method": "GET",
   "headers": {
    "Host": "www.okayama-u.ac.jp",
    "Connection": "keep-alive",
    "Upgrade-Insecure-Requests": "1",
    ...
   }
  },
  "response": {
   "version": "HTTP/1.1",
   "status_code": "200",
   "headers": {
    "Date": "Wed, 23 Sep 2020 05:56:57 GMT",
    "Server": "Apache",
    "X-Frame-Options": "SAMEORIGIN",
    ...
   }
  }
 }
]
```

Fig. 9 Example of a communication log of JSON format stored by WebView Monitor

References

1. Wikipedia: Android (operating system). https://en.wikipedia.org/wiki/Android_$operating_system)#Market_share$ (2019). Accessed 24 Dec 2019
2. Mobile Threat Report: McAfee Mobile Threat Report Q1, 2018. https://www.mcafee.com/enterprise/en-us/assets/reports/rp-mobile-threat-report-2018.pdf (2018). Accessed 4 June 2019
3. Wandera: Android Malware: 4 Ways Hackers are Infecting Phones with Viruses. https://www.wandera.com/malware-on-android/ (2018). Accessed 4 June 2019
4. Luo, T., Hao, H., Du, W., Wang, Y., Yin, H.: Attacks on WebView in the Android system. In: Proceedings of the 27th Annual Computer Security Applications Conference (ACSAC’11), pp. 343–352 (2011)
5. Mutchler, P., Doupé, A., Mitchell, J., Kruegel, C., Vigna, G.: A large-scale study of mobile Web App security. In: Proceedings of the Mobile Security Technologies Workshop (MoST’15) (2015)
6. Jin, X., Wang, L., Luo, T., Du, W.: Fine-grained access control for HTML5-based mobile applications in Android. In: Proceedings of the 16th Information Security Conference (ISC’13), pp. 309–318 (2013)
7. Yu, J., Yamauchi, T.: Access control to prevent malicious JavaScript code exploiting vulnerabilities of WebView in Android OS. IEICE Trans. Inf. Syst. **E98–D**(4), 807–811 (2015)
8. Tuncay, G.S., Demetriou, S., Gunter, C.A.: 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 (CCS’16), pp. 104–115 (2016)
9. Kudo, N., Yamauchi, T., Austin, T.H.: Access control mechanism to mitigate cordova plugin attacks in hybrid applications. J. Inf. Process. **26**, 396–405 (2018)
10. Li, T., Wang, X., Zha, M. et al.: Unleashing the walking dead: understanding cross-app remote infections on mobile WebViews. In: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security (CCS’17), pp. 829–844 (2017)
11. Yang, G., Huang, J., Gu, G.: Automated generation of event-oriented exploits in Android Hybrid Apps. In: Proceedings of the

Network and Distributed System Security Symposium (NDSS'18), pp. 1–15 (2018)

12. Son, S., Kim, D., Shmatikov, V.: What mobile ads know about mobile users. In: Proceedings of the Network and Distributed System Security Symposium (NDSS'16), pp. 1–15 (2016)
13. Neugschwandtner, M., Lindorfer, M., Platzer, C.: A View to a Kill: WebView Exploitation. In: Proceeding of the 6th USENIX Workshop on Large-Scale Exploits and Emergent Threats (LEET'13) (2013)
14. Luo, T., Du, W., Wang, Y.: Attacks and countermeasures for webview on mobile systems. Ph.D. Dissertation. Syracuse University (2014)
15. Bhavani, A.B.: Cross-site scripting attacks on android WebView. Int. J. Comput. Sci. Netw. **2**(2), 1–5 (2013)
16. Bao, W., Yao, W., Zong, M., Wang, D.: Cross-site scripting attacks on android hybrid applications. In: Proceedings of the 2017 International Conference on Cryptography, Security and Privacy (ICCSP'17), pp. 56–61 (2017)
17. Trend Micro: Social media malware on the rise. https://blog.trendmicro.com/social-media-malware-on-the-rise/ (2015). Accessed 4 June 2019
18. CalyptixSecurity: Social Media Threats: Facebook Malware, Twitter Phishing, and More. https://www.calyptix.com/top-threats/social-media-threats-facebook-malware-twitter-phishing/ (2017). Accessed 4 June 2019
19. WebKit: Open Source Browser Engine. https://webkit.org/ (2019). Accessed 6 Feb 2019
20. Google: The Chromium project. https://www.chromium.org/. Accessed 6 Feb 2019
21. Google: The Chromium project, NetworkStack. https://www.chromium.org/developers/design-documents/network-stack/. Accessed 6 Feb (2019)

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.