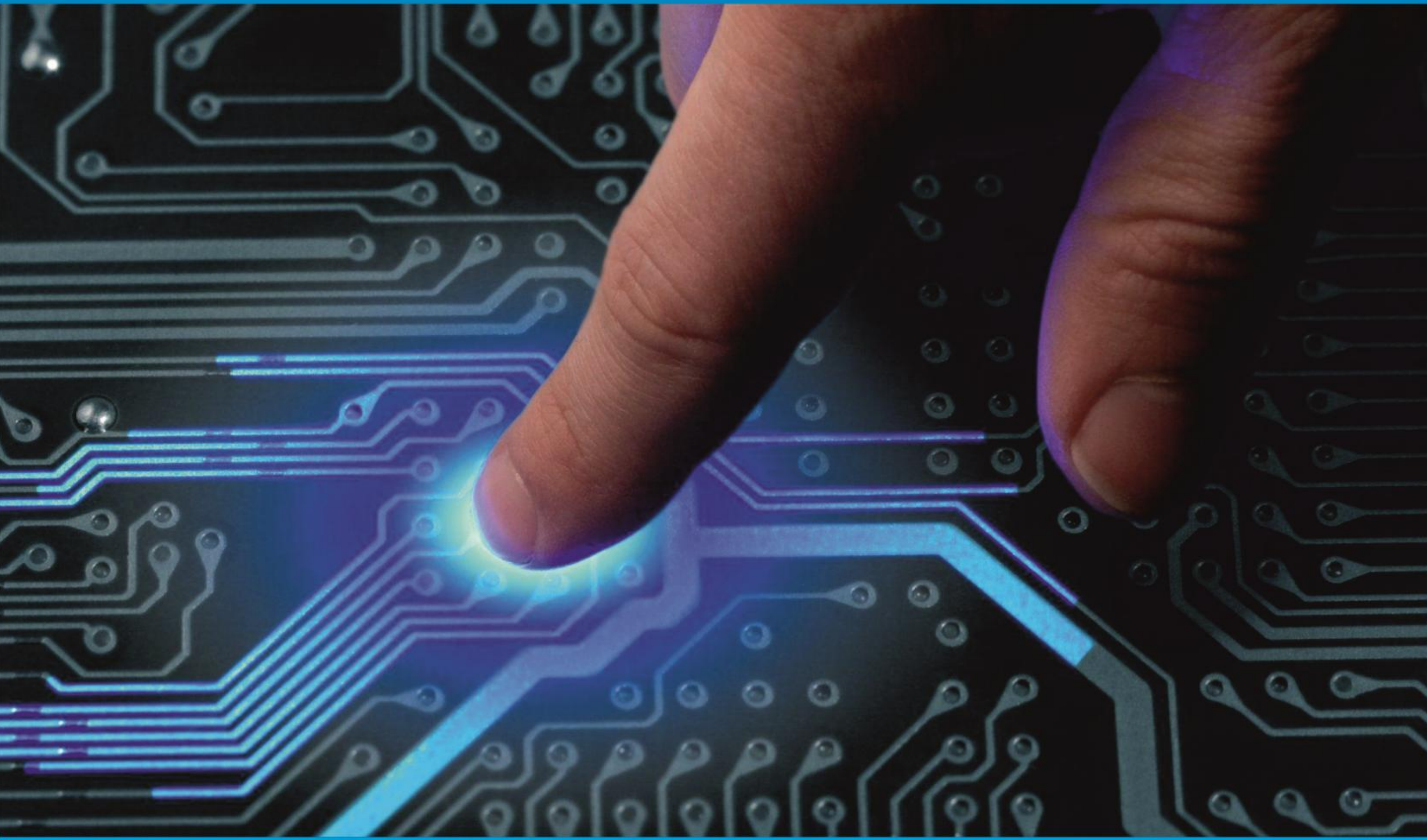
IJIRCCE

e-ISSN: 2320-9801 | p-ISSN: 2320-9798



INTERNATIONAL JOURNAL OF INNOVATIVE RESEARCH

IN COMPUTER & COMMUNICATION ENGINEERING

Volume 9, Issue 5, May 2021

INTERNATIONAL STANDARD SERIAL NUMBER INDIA

Impact Factor: 7.488

International Journal of Innovative Research in Computer and Communication Engineering

| e-ISSN: 2320-9801, p-ISSN: 2320-9798| www.ijircce.com | |Impact Factor: 7.488 |

|| Volume 9, Issue 5, May 2021 ||

| DOI: 10.15680/IJIRCCE.2021.0905247 |

Recommending Security Requirements for the Development of Android Applications based on Sensitive APIs

Bhawna Goel, Gargi Darwhatkar, Mihika Gavali, Aditya Panchal, Prof. S.S Vanjire

Department of Computer Engineering, Sinhgad Academy of Engineering, Pune, India

ABSTRACT: Smartphones are used by billions of people that means the applications of the smartphone is increasing, it is out of control for applications marketplaces to completely validate if an application is malicious or legitimate. Therefore, it is up to users to choose for themselves whether an application is safe to use or not. It is important to say that there are differences between mobile devices and PC machines in resource management mechanism, the security solutions for computer malware are not compatible with mobile devices. Consequently, the anti-malware organizations and academic researchers have produced and proposed many security methods and mechanisms in order to recognize and classify the security threat of the Android operating system. By means of the proposed methods are different from one to another, they can be arranged into various classifications. In this review paper, the present Android security threats is discussed and present security proposed solutions and attempt to classify the proposed solutions and evaluate them.

KEYWORDS: Security, Sensitive APIs, hijacking, HTTP-based API.

I. INTRODUCTION

Modern mobile apps use cloud-hosted HTTP-based API services and heavily rely on the Internet infrastructure for data communication and storage. To improve performance and leverage the power of the mobile device, input validation and other business logic required for interfacing with web API services are typically implemented on the mobile client. However, when a web service implementation fails to thoroughly replicate input validation, it gives rise to inconsistencies that could lead to attacks that can compromise user security and privacy. Developing automatic methods of auditing web APIs for security remains challenging. In this paper, we present a novel approach for automatically analysing mobile app-to-web API communication to detect inconsistencies in input validation logic between apps and their respective web API services. We present our system, Android, which implements a static analysis-based web API reconnaissance approach to uncover inconsistencies on real world API services that can lead to attacks with severe consequences for potentially millions of users throughout the world. Our system utilizes program analysis techniques to automatically extract HTTP communication templates from Android apps that encode the input validation constraints imposed by the apps on outgoing web requests to web API services. The proliferation of mobile devices has resulted in an extensive array of mobile applications (apps) that serve diverse needs of our connected society. Today's modern lifestyle increasingly depends on mobile apps that serve a wide spectrum of functionality including military applications, critical business services, banking, entertainment, and other diverse functionality. Mobile apps are often built as front-ends to services hosted in the cloud infrastructure and accessible through web API services. The web platform, through the use of HTTP and HTTPS [1], serves as the main conduit for communication between mobile applications and their respective web API services. Previous research work in the mobile space has mostly focused on security and privacy of the mobile device and data stored locally on the device. However, remote HTTP based services form an integral part of the mobile application ecosystem and deserve similar scrutiny with regard to security and privacy concerns. This fact is evidenced by the placement of Weak Server Controls as the top vulnerability in the OWASP top 10 mobile vulnerabilities.

II. PROBLEM STATEMENT

While mobile apps may have robust input validation and access control logic implemented in their native code, those are often not equally replicated on the server side for data sent to a web API. As a result, an attacker can bypass client side controls and exploit a web API service to extricate data or inject malicious data without proper authorization. This is noted in the recent paper by Sudhodananet. al.. In this paper we aim to systematically study and (semi-)automatically detect the inconsistencies between data validation logic in a mobile app and data validation logic implemented at a remote web API server. While this is inspired by previous work on web parameter tampering, we

| e-ISSN: 2320-9801, p-ISSN: 2320-9798| www.ijircce.com | |Impact Factor: 7.488 |

|| Volume 9, Issue 5, May 2021 ||

| DOI: 10.15680/IJIRCCE.2021.0905247 |

address challenges in uncovering web API data validation logic in mobile apps, where client-to-server communication is not as inherent as on the web platform. We also highlight the real world security impact of inconsistent app-to-web validation on the mobile ecosystem caused by loose coupling between mobile and web validation logic. Transactions between mobile apps and web API services require careful coordination of data validation logic to ensure that security controls are consistently implemented. For example, if a mobile app restricts the data type of a user input field, we expect that the server should also implement a similar restriction to ensure consistency. Unfortunately, it is difficult or impossible to ensure complete consistency between controls built into the mobile app and controls actually enforced at the server side. In many cases, the server should enforce more constraints than the client (such as enforcing uniqueness of usernames, for example). In this paper, we assume that the server is at least as strict as the client. Remote web API service implementations are often shared among different user agents (mobile and browser), giving rise to further inconsistencies in the implementations of the application logic between different apps that use the same backend web API. For the sake of scalability, web APIs may even skip input validation and defer that job to the apps. It is also not always feasible for remote web API services to authenticate all clients, giving rise to various replay attacks where attackers can impersonate legitimate clients or access functionality intended for legitimate clients without authentication or authorization.

III. LITERATURE REVIEW

Modern mobile apps use cloud-hosted HTTP-based API services and heavily believe the web infrastructure for digital communication and storage. to enhance performance and leverage the facility of the mobile device, input validation and other business logic required for interfacing with web API services are typically implemented on the mobile client. However, when an internet service implementation fails to thoroughly replicate input validation, it gives rise to inconsistencies that would cause attacks which will compromise user security and privacy. Developing automatic methods of auditing web APIs for security remains challenging. during this paper, we present a completely unique approach for automatically analyzing mobile app-to-web API communication to detect inconsistencies in input validation logic between apps and their respective web API services. We present our system, Android, which implements a static analysis-based web API reconnaissance approach to uncover inconsistencies on world API services which will cause attacks with severe consequences for potentially many users throughout the planet . Our system utilizes program analysis techniques to automatically extract HTTP communication templates from Android apps that encode the input validation constraints imposed by the apps on outgoing web requests to web API services. Android is additionally enhanced with black box testing of server validation logic to spot inconsistencies which will cause attacks. We evaluated our system on a group of 10,000 popular free apps from the Google Play Store. We detected problematic logic in APIs utilized in over 4,000 apps, including 1,743 apps that use unencrypted HTTP communication. We further tested 1,000 apps to validate web API hijacking vulnerabilities which will cause potential compromise of user privacy and security and located that many users are potentially affected from our sample set of tested apps.

The popularity of web services has largely influenced the way during which enterprise business is conducted. Since web services enable easy accessibility of knowledge , dynamic connections, and comparatively less human interventions, ensuring confidentiality and integrity of knowledge that's transmitted via web services protocols becomes more significant. If one service doesn't fulfil the service consumer requirements, it's necessary to compose several web services, which together satisfy the user requirements. Security attacks occur on SOAP messages that are communicated among web services while accessing a service or during service composition. Most of the prevailing works on web services security have provided solutions just for ensuring client authentication, confidentiality, and integrity of data in network layer and not in application layer. WS-Security and XML based web service security also provides message layer security in network layer and not in application layer. Hence, a completely unique approach that forestalls the message alteration attack on SOAP messages and a security solution that detects and overcomes XML injection attack are proposed during this paper. Our approach uses pluggable APIs within the service provider side and security services within the middleware side. The attacks were simulated and non-vulnerability of the proposed solutions to those attacks are verified.

IV. PROPOSED SYSTEM

The primary goal of WAR Droid is a novel application of static taint analysis and symbolic execution to uncover web API input validation constraints and reason about web API hijacking opportunities by evaluating inconsistencies. To achieve this goal, we extend Flow droid to comprehensively analyse web-related code paths and constraints in apps that lead to network APIs that generate HTTP(S) messages. We therefore model the web API's server-side validation logic using the mobile application validation logic. We can then detect inconsistencies by deriving invalid API requests that fail in our mobile application model but does not fail when testing on the actual server. We characterize the application validation logic as a symbolic path constraint on a static abstraction of the web request functionality which is a subset of the program dependence graph (PDG) of the app. We represent the constraints in the format of Z3 and utilize the Z3-Str library to generate both valid and invalid concrete API requests for testing through message replay.

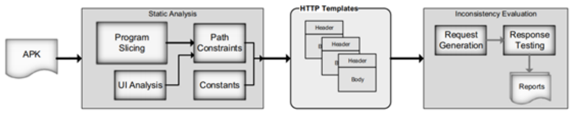


Fig.1. Proposed System

WAR Droid takes the application APK package as input and produces possible web API hijacking opportunities as output. First, we model the mobile app's web API communication into HTTP message templates. To accomplish this, we utilize program analysis techniques that analyse the app to extract the program slices that generate HTTP requests from each POI. The main task is to track all dependencies that eventually flow to network buffers through particular Android framework APIs. This allows us to extract the relevant path constraints and reason about the web API requests generated by the app. To this end, our system extracts and analyses the program slices that generate and process HTTP messages using data dependency analysis. We augment the resulting program dependence graph slices with information from the user interface (UI) resources in the app that define additional constraints imposed by UI elements on user input data that eventually make up part of the web API request.

V. RESULTS

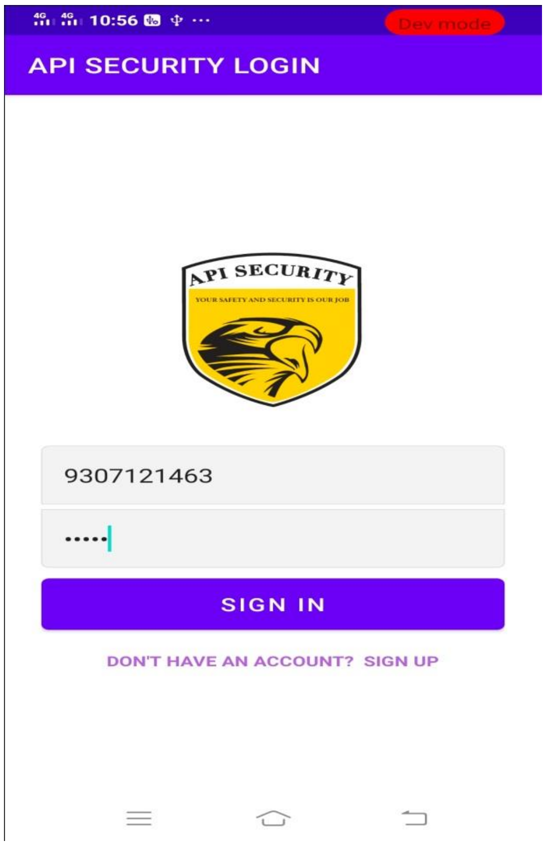

10:56
Dev mode
API SECURITY LOGIN
API SECURITY
YOUR SAFETY AND SECURITY IS OUR JOB
9307121463
•••••
SIGN IN
DON'T HAVE AN ACCOUNT? SIGN UP

Fig. 2. Login

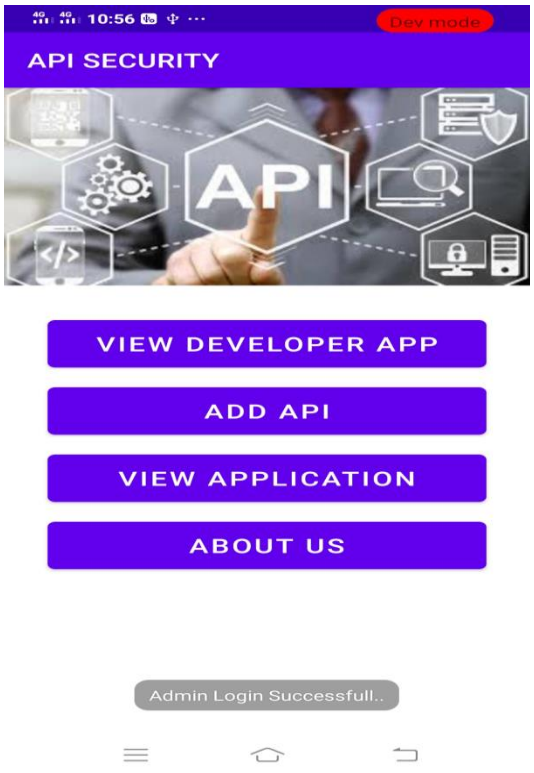
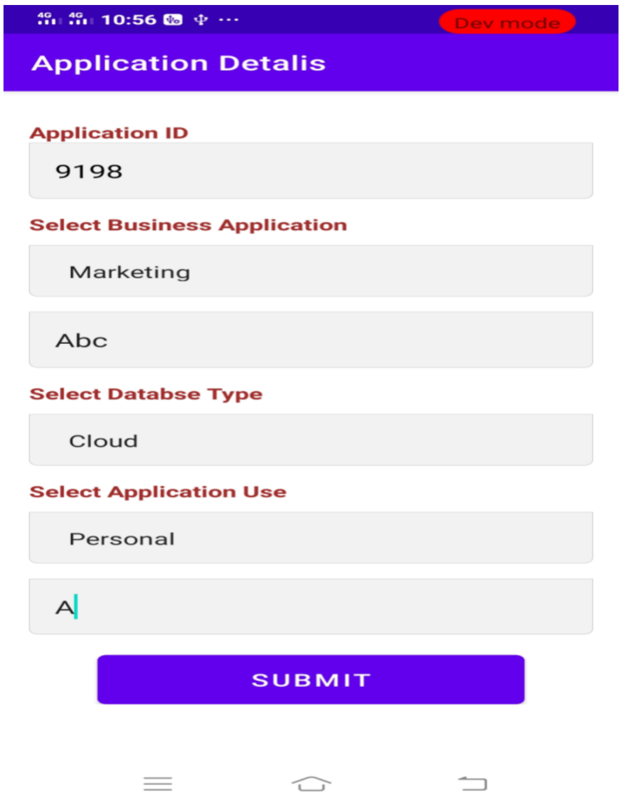


Fig.3. Home Page



Application Detalis

Application ID

9198

Select Business Application

Marketing

Abc

Select Databse Type

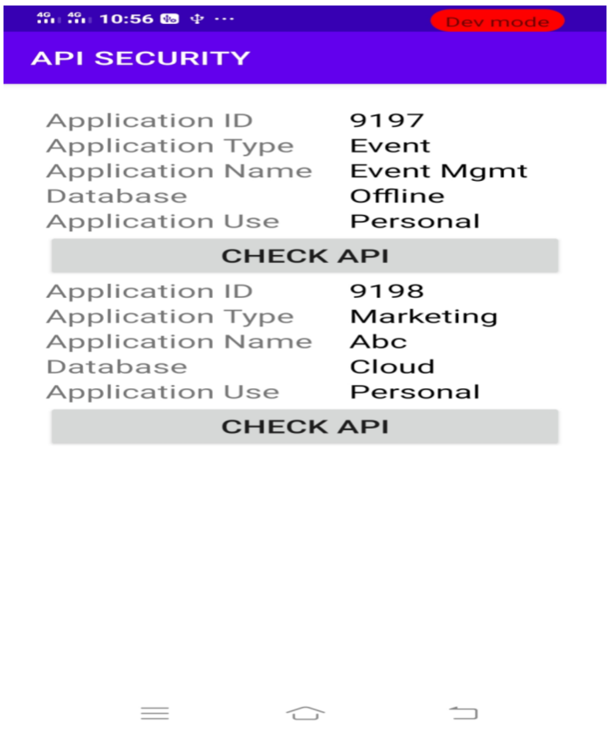
Cloud

Select Application Use

Personal

A

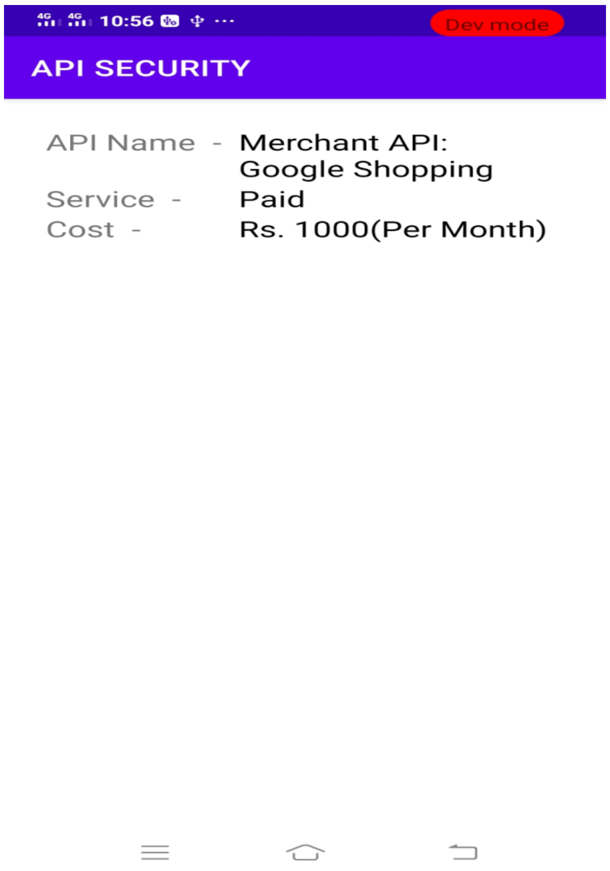
Fig.4. Add Project



Application ID	9197
Application Type	Event
Application Name	Event Mgmt
Database	Offline
Application Use	Personal

Application ID	9198
Application Type	Marketing
Application Name	Abc
Database	Cloud
Application Use	Personal

Fig.5. Add API



API Name -	Merchant API: Google Shopping
Service -	Paid
Cost -	Rs. 1000(Per Month)

Fig.6. Available API

| e-ISSN: 2320-9801, p-ISSN: 2320-9798| www.ijircce.com | |Impact Factor: 7.488 |

|| Volume 9, Issue 5, May 2021 ||

| DOI: 10.15680/IJIRCCE.2021.0905247 |

VI. **CONCLUSION**

Modern mobile applications rely on web services to enable their functionality through HTTP-based communication. Unfortunately, the disparate nature of the mobile and web platforms causes input validation inconsistencies that can lead to serious security issues. We presented WAR Droid, a framework that utilizes static program analysis and symbolic execution to model input validation logic between mobile apps and their remote web API servers. WAR Droid extracts and validates web API logic implementation in mobile apps and uncovers inconsistencies between the app and server logic. The uncovered inconsistencies are shown to expose serious vulnerabilities in web API servers that affect a diverse set of mobile apps. Our analysis of 10,000 apps uncovered a significant portion of apps with web API hijacking opportunities that can violate user privacy and security for millions of mobile app users. The inconsistency problem is not limited to Android apps, but any client that utilizes the deployed web API services, including iOS apps, Windows apps, and web applications. This work sheds light on the existence and pervasiveness of this important ongoing research problem, and our hope is that it will motivate further research in this area.

REFERENCES

[1] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee, "Hypertext transfer protocol–http/1.1," Tech. Rep., 1999.

[2] "OWASP Mobile Threats," https://www.owasp.org/index.php/Projects/ OWASP Mobile Security Project - Top Ten Mobile Risks.

[3] P. Bisht, T. Hinrichs, N. Skrupsky, R. Bobrowicz, and V. Venkatakrishnan, "Notamper: automatic blackbox detection of parameter tampering opportunities in web applications," in Proceedings of the 17th ACM conference on Computer and communications security. ACM, 2010, pp. 607–618.

[4] S. Stamm, B. Sterne, and G. Markham, "Reining in the web with content security policy," in Proceedings of the 19th international conference on World wide web. ACM, 2010, pp. 921–930.

[5] K. Singh, A. Moshchuk, H. J. Wang, and W. Lee, "On the incoherencies in web browser access control policies," in 2010 IEEE Symposium on Security and Privacy. IEEE, 2010, pp. 463–478.

[6] A. Mendoza, K. Singh, and G. Gu, "What is wrecking your data plan? a measurement study of mobile web overhead," in Computer Communications (INFOCOM), 2015 IEEE Conference on. IEEE, 2015, pp. 2740–2748.

[7] P. Bisht, T. Hinrichs, N. Skrupsky, and V. Venkatakrishnan, "Waptec: whitebox analysis of web applications for parameter tampering exploit construction," in Proceedings of the 18th ACM conference on Computer and communications security. ACM, 2011, pp. 575–586.

[8] A. Sudhodanan, A. Armando, R. Carbone, L. Compagna et al., "Attack patterns for black-box security testing of multi-party web applications." in NDSS, 2016.

[9] A. Barth, C. Jackson, and J. C. Mitchell, "Robust defenses for crosssite request forgery," in Proceedings of the 15th ACM conference on Computer and communications security. ACM, 2008, pp. 75–88.

[10] R. Vallee-Rai and L. J. Hendren, "Jimple: Simplifying java bytecode for analyses and transformations," 1998.