Verifying OAuth Implementations Through Encrypted Network Analysis

Josh Talkington
JoshTalkington@my.unt.edu
University of North Texas
Department of Computer Science
Denton, Texas

Ram Dantu
ram.dantu@unt.edu
University of North Texas
Department of Computer Science
Denton, Texas

Kirill Morozov
kirill.morozov@unt.edu
University of North Texas
Department of Computer Science
Denton, Texas

ABSTRACT

Verifying protocol implementations via application analysis can be cumbersome. Rapid development cycles of both the protocol and applications that use it can hinder up-to-date analysis. A better approach is to use formal models to characterize the applications platform and then verify the protocol through analysis of the network traffic tied to the models. To test this method, the popular protocol OAuth is considered. Currently, formal models of OAuth do not take into consideration the mobile environment, and implementation verification is largely based on code analysis. Our preliminary results are two fold; we sketch an extension to a formal model that incorporates the specifics of the Android platform and classify OAuth device types using machine learning on encrypted VPN traffic.

CCS CONCEPTS

• **Networks** → **Protocol correctness**; **Application layer protocols**; • **Security and privacy** → *Formal security models*; *Malware and its mitigation*;

KEYWORDS

Authorization; Android; Formal Models; OAuth; Network Analysis

ACM Reference Format:
Josh Talkington, Ram Dantu, and Kirill Morozov. 2019. Verifying OAuth Implementations Through Encrypted Network Analysis. In *The 24th ACM Symposium on Access Control Models and Technologies (SACMAT '19), June 3–6, 2019, Toronto, ON, Canada.* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/3322431.3326449

1 INTRODUCTION

Protocols can be hard to implement. Developers might not realize the importance of certain features, or find the protocol hard to follow . This can lead to vulnerabilities in the applications themselves, and subject the service they use to abuse. Given the massive amounts of user generated data, authorization is a property that is at the forefront of this atmosphere. One popular protocol that handles this interaction is Open Authorization (OAuth). OAuth has been that subject of many conversations that revolve around its various, and often vulnerable, implementations. Developers often confuse its role as an *authorization* protocol with an *authentication* protocol, or they use the wrong operation mode by not enabling important extensions [1]. This has promoted a response from the community for techniques to verify the implementations.

There have been several works in this regard. Efforts like Proverif [8] and those mentioned in [10] rely on formal methods. The work of Wang et al assess vulnerability in OAuth implementations but does so through code analysis [11]. While these methods have their advantages, they can lack agility in dealing with rapid development cycles and can require large resources of time and computation. We approach the problem from a network perspective. Network traffic analysis is attractive because it abstracts out many of the limits of other techniques. You do not need deep access to the source code, the same techniques can be applied across device types, protocol extensions, and implementation details. However, one hurdle to get over is the use of traffic encryption. This makes network traffic harder as it masks the data from analysis. To get over this challenge, we propose leveraging the formal protocol model to increase our ability to successfully analyze the encrypted traffic. What is presented here is a brief extension to an OAuth formal model that characterizes the Android platform, results from OAuth device fingerprinting.

The rest of this paper is organized as follows: in section 2 we briefly summarize OAuth, in section 3 we describe a baseline approach to OAuth device detection, in section 4 we sketch an Android Model extension, and in section 5 we conclude.

2 OAUTH

Open Authorization (OAuth) is a protocol that allows third party access to a users resources at another location [6]. It represents a mechanism which is commonly invoked by the prompts like "Login with Facebook" or "Login with Google". This protocol has several entities like the identity provider (IdP), the Relying Party (RP), and the web browser. The IdP manages users' credentials and private resources. The RP or client is a third party that is supposed to gain access to your data at the IdP. The browser is just a typical web browser that allows the user to visit the RP and IdP.

OAuth has various modes of operation: client credentials, resource owner password credentials, implicit, and authorization code mode. All modes have their uses, but the authorization code mode is the most encompassing one and hence it will be used as the example for the rest of this paper. This mode has the following steps: The user first visits the RP website where they can login with an existing identity. When they click on the "Login with..." button, they are redirected to the IdP. At the IdP, they login and

SACMAT '19, June 3–6, 2019, Toronto, ON, Canada

ACM ISBN 978-1-4503-6753-0/19/06.
https://doi.org/10.1145/3322431.3326449

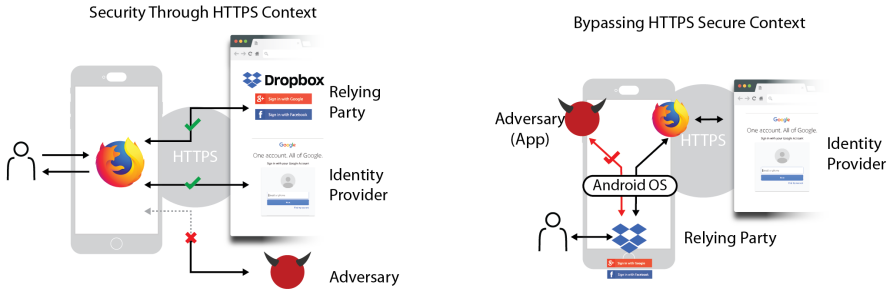

Figure 1: OAuth in the web vs. the mobile environment – bypassing HTTPS context on the mobile.

grant resource access to the RP. Once the user is verified with a user name and password, they are redirected to the RP with a code. This code is taken by the RP and exchanged for a token. The RP is supposed to access your resources (such as a profile) with this access token.

This base protocol works for the general web, but is not enough for the mobile phone. Figure 1 shows the difference in configuration of the entities in the web system as compared to that in a mobile phone. On the web, both the RP and the IdP are protected by the HTTPS protocol, confirming the identity of the sender (according to the certificate authority) and protecting the messages from tampering. There exists no comparable mechanism that is present at the operating system level. While an interception style attack is generally not attractive, this allows any application installed on the device to send messages to the RP. There are many more attacks on OAuth such as phishing, covert redirects, and repackaging. Lu et al. published a comprehensive overview of vulnerabilities related to single sign on, particularly on Android [9].

3 DETECTING DEVICES ON ENCRYPTED NETWORK

This section describes a brief experiment in which we classify network devices based on their type (desktop or rasberry pi) while they sign in with OAuth. Traffic will be collected from networked devices trying to access an external service over a VPN. They will act as a user requesting access to a resource. Their network traffic will be fed into the Weka program to perform machine learning classification [5].

The network contains a Raspberry Pi (model B) [4], a desktop with browser, and a proxy server connected via a home network router. The desktop and proxy server are virtual machines. The Raspberry Pi is running Rasbian, a Debian based OS. They connect via OAuth to an external web sever. The desktop acts as a control, allowing the traffic analysis to be compared to a standard OAuth scenario. It is running Ubuntu with the Chrome web browser. It will be run through the Chrome development tools backend server to make OAuth requests by loading crafted URLs using python.

The proxy will run Ubuntu server with the necessary network routes to collect all network traffic. The incoming traffic is from the Pi and virtual desktop and is encrypted with a Virtual Private Network (VPN). Before it leaves the network, the proxy collects the VPN packets for inspection. Each set of packets will be combined and labeled with contextual information like what step of OAuth is occurring. The traffic is then decrypted an sent externally as regular Internet traffic.

All of the end devices connect to the service that is external to the network above. This service will support OAuth for authorization. Devices will authenticate with the service using one of the OAuth flows defined in [6]. They will then request access to a resource, a profile for instance.

3.1 Features

To measure the system, a set of features will be derived from the network trace. For each OAuth run, there will be a number of packets. These packets will contain attributes, such as packet length, that will be analyzed to find averages, standard deviations, etc. Zhioua *et al* used interesting features in their TOR browser fingerprinting [12], and will be used as inspiration in these choices.

Currently, the list of features contains the following:

Entropy measures the Shannon entropy of the payload
Complexity is approximated via payload compression
Average Length of the packet
Average Time delta is the average time between packets
Pkt Count is the amount of packets

Figure 2 shows distinct clusters when visualizing the data. The y-axis is the Shannon entropy listed above with the x-axis of each graph being one of the features listed earlier. The clustering of data comes from the fact that the different devices utilize different configurations and values during OAuth, this effects measurement such as packet count. These preliminary results show the feasibility in detecting the different device types when they use OAuth over an encrypted network and provided a baseline for improvement. Using a formal model will help us describe, in detail, the OAuth characteristics of the devices involved and we can then use these to produce new features.

4 FORMAL MODELING

Up until now, modeling the security and safety of mobile phones has received little attention, to the best of our knowledge. Khan

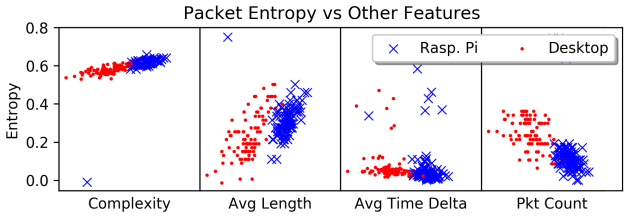

Figure 2: Differentiating device types when signing in to OAuth over a VPN

et al. model Android to further reason about application crashes [7]. Smith and Coglio proposes several features in their model such as activity history, event history, and memory stack [10]. These approaches are all static in nature as they do not consider future capabilities, like software updates, meaning when a new adversarial vulnerability arises, the model might completely break down.

Fett et al. [3] introduced a model that showed OAuth secure with respect to authentication and authorization. What this model did not account for is the mobile native app and its protocol extensions [2]. This was due to the inability of the model to accurately characterize the new type of domain and the adversaries that existed in it. The RP is now on the phone, directly interacting with the user, not the browser. It is subjected to Android interprocess communication with intents, as shown in Figure 1.

To get a sense of Fett et al's methodology and to provide a template, we can look at the definition provided for the Relying Parties's initial state. Definition 42 from [3] describes the RPs initial state:

Definition 42. A *state* $s \in Z^{\mathrm{r}}$ *of an RP* r is a term of the form $\langle$*DNSAddress, idps, serviceTokens, loginSessions, keyMapping, sslkeys, pendingDNS, pendingRequests, corrupt*$\rangle$ where *DNSAddress* $\in$ IPs, *idps* $\in$ [Doms $\times \mathcal{T}_{\mathcal{N}}$] is a directory of IdP registration records, *serviceTokens* $\in [\mathcal{N} \times \mathcal{T}_{\mathcal{N}}]$, *loginSessions* $\in [\mathcal{N} \times \mathcal{T}_{\mathcal{N}}]$ is a dictionary of login session records, *keyMapping* $\in [\mathbb{S} \times \mathcal{N}]$, *sslkeys* = *sslkeys*$^{\mathrm{r}}$, *pendingDNS* $\in [\mathcal{N} \times \mathcal{T}_{\mathcal{N}}]$, *oendingRequests*$\in [\mathcal{N} \times \mathcal{T}_{\mathcal{N}}]$, *corrupt* $\in \mathcal{T}_{\mathcal{N}}$.

An *initial state* s_0^r *of* r is a state of r with s_0^r.idps being a dictionary that maps each domain of all identity providers i to an IdP registration record for i at r, s_0^r.serviceTokens = s_0^r.loginSessions = {}, s_0^r.corrupt = $\bot$, and s_0^r.KeyMapping is the same as the keymapping for browsers above.

Looking at this definition of an initial state, we can see that it is built from ground terms akin to a web browser. Following the example above, we can sketch a initial state definition for an Android application.

Definition 1. an *intent* is a term in the form $\langle$*action*, *category*, *data*, *scheme*$\rangle$ Where *action* $\in A$ is the set of all actions for an activity, *category* $\in C$ is a category for an intent, *data* $\in D$ represents the data contained in the intent, *scheme* $\in S$ serves as the intent scheme chosen by the developer initiated at registration.

Definition 2. a *view* is a term in the form $\langle$*rid*, *onClickListeners*$\rangle$ Where *rid* $\in R$ is the set of all ids for an activity *category* $\in C$ is a category for an intent.

Definition 3. *eventHistory* is modeled via Androids hierarchical state machine

Definition 4. A *state* $s \in Z$ of an Android application A is of the form $\langle$*intents, views, listeners, eventHistory*$\rangle$ where *intents* $\in I$, *views* $\in V$, *listeners* $\in L$, and *eventHistory* $\in E$.

5 CONCLUSIONS AND FUTURE WORK

Presented here is motivation for new types of protocol implementation verification that can deal with changing atmosphere, a preliminary experiment showing how to differentiate device types on an encrypted network and sketching a formal model extension in order to produce new features for our network analysis.

Combining the formal model with the network analysis allows for a different approach to detecting flaws in OAuth implementations and even the standard itself. Future work would include fully extending the model to encompass all the features of an Android application, aiding in analysis. Then applying this technique across a data set of common applications.

REFERENCES

[1] Eric Y. Chen, Yutong Pei, Shuo Chen, Yuan Tian, Robert Kotcher, and Patrick Tague. 2014. OAuth Demystified for Mobile Application Developers. ACM Press, 892–903. https://doi.org/10.1145/2660267.2660323

[2] William Denniss and John Bradley. 2017. *OAuth 2.0 for Native Apps.* Number 8252 in Request for Comments. RFC Editor. https://doi.org/10.17487/RFC8252 Published: RFC 8252.

[3] Daniel Fett, Ralf KÃijsters, and Guido Schmitz. 2016. A Comprehensive Formal Security Analysis of OAuth 2.0. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security (CCS '16).* ACM, New York, NY, USA, 1204–1215. https://doi.org/10.1145/2976749.2978385

[4] Raspberry Pi Foundation. [n. d.]. Raspberry Pi âĂŤ Teach, Learn, and Make with Raspberry Pi. ([n. d.]). https://www.raspberrypi.org

[5] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H. Witten. 2009. The WEKA Data Mining Software: An Update. *SIGKDD Explor. Newsl.* 11, 1 (Nov. 2009), 10–18. https://doi.org/10.1145/1656274.1656278

[6] Dick Hardt. 2012. The OAuth 2.0 authorization framework. (2012). http://tools.ietf.org/html/rfc6749

[7] Wilayat Khan, Habib Ullah, Aakash Ahmad, Khalid Sultan, Abdullah J. Alzahrani, Sultan Daud Khan, Mohammad Alhumaid, and Sultan Abdulaziz. 2018. CrashSafe: a formal model for proving crash-safety of Android applications. *Human-centric Computing and Information Sciences* 8, 1 (Dec. 2018). https://doi.org/10.1186/s13673-018-0144-7

[8] Nadim Kobeissi, Karthikeyan Bhargavan, and Bruno Blanchet. 2017. Automated Verification for Secure Messaging Protocols and Their Implementations: A Symbolic and Computational Approach. IEEE, 435–450. https://doi.org/10.1109/EuroSP.2017.38

[9] Xing Liu, Jiqiang Liu, Wei Wang, and Sencun Zhu. 2018. Android single sign-on security: Issues, taxonomy and directions. *Future Generation Computer Systems* 89 (2018), 402 – 420. https://doi.org/10.1016/j.future.2018.06.049

[10] Eric Smith and Alessandro Coglio. 2016. Android Platform Modeling and Android App Verification in the ACL2 Theorem Prover. In *Verified Software: Theories, Tools, and Experiments*, Arie Gurfinkel and Sanjit A. Seshia (Eds.). Vol. 9593. Springer International Publishing, Cham, 183–201. https://doi.org/10.1007/978-3-319-29613-5_11

[11] Hui Wang, Yuanyuan Zhang, Juanru Li, Hui Liu, Wenbo Yang, Bodong Li, and Dawu Gu. 2015. Vulnerability Assessment of OAuth Implementations in Android Applications. ACM Press, 61–70. https://doi.org/10.1145/2818000.2818024

[12] Sami Zhioua. 2015. The web browser factor in traffic analysis attacks. *Security & Communication Networks* 8, 18 (Dec. 2015), 4227–4241. https://doi.org/10.1002/sec.1338

ACKNOWLEDGMENTS

This material is based upon work supported by the National Science Foundation under awards 1241768 and 1637291.