Mobile SDNs: Associating End-User Commands with Network Flows in Android Devices

Shuwen Liu, Craig A. Shue, Joseph P. Petitti, Yunsen Lei, and Yu Liu

***Abstract*—Mobile devices pose several distinct challenges from a security perspective. First, they have varied and ephemeral network connections, often using a cellular provider network as a backup option when connectivity is not available via wireless local access networks. This varied network connectivity makes it difficult to comprehensively deploy in-network solutions, such as firewalls or intrusion detection systems, since they would have to be active in every network the device would use. Second, with personally-owned devices, the device owner may have security goals and privacy priorities that are distinct from organizations that provide connectivity or data assets, such as employers or schools. These complex relationships may complicate efforts to protect the devices.**

This paper explores a technique that runs on the mobile device endpoints to learn about the usage patterns associated with the device, in order to enforce network policy. We explore sensors that examine the mobile device's user interface, using physical inputs via finger taps, and that link them with the network activity on the device. We incorporate with allow-list policies that can be provided by organizations to make on-device access control decisions. Using IP address and DNS host name allow-lists as a baseline, we explore the accuracy of interface-aware allow-lists. We find the interface-aware allow-lists can reach over 98.5% accuracy, even when user-specified destinations are used, greatly exceeding the baseline accuracy. Our performance evaluation indicates our approach introduces a median of 3.87 ms of overall delay with low CPU usage.

***Index Terms*—Android, Software-Defined Networking, Network Profiling, User Interface**

I. Introduction

MOBILE devices can be personally-owned while still playing a vital role in organizations associated with the owner, such as employers and schools. The technical knowledge of the device owner may vary. Further, the device owner may have strong or weak associations with organizations. In some cases, device owners and organizations may have aligned goals, such as securing data, but they may have different perspectives when it comes to device control and privacy.

From an organization's perspective, understanding of a device's specification can help with solving problems and managing resources [1]. Unfortunately, low-level logs can be difficult to examine and determine the origins of problems and security risks [2]. With mobile devices often leaving an organization's network, in-network monitoring and security devices may have an incomplete view. To form a complete picture, security tools need to be installed on the mobile device itself.

Shuwen Liu (corresponding author, email address: sliu9@wpi.edu), Craig A. Shue, Joseph P. Petitti, Yunsen Lei, and Yu Liu are with the Computer Science Department, Worcester Polytechnic Institute, Worcester, Massachusetts, USA.

While an on-device security tool may be needed, such tools must avoid impractical requirements (e.g., recompiling the Android kernel or gaining root access to their device, although those may incur its own security risks [3]), which were prerequisites in prior work [4], [5]. Our prior work [6] found that software-defined networking (SDN) tools can operate as useful endpoint sensors on the Windows operating system. That work used sensors to locally collect information and consulted an SDN controller to decide whether to authorize connections. This provides significant visibility and control of the device. However, while that prior work could leverage kernel device drivers to perform its operations, the mobile device space places greater constraints on sensing software and access.

In this work, we explore SDN endpoint sensors that work within standard APIs and permissions in Android devices. This paper expands upon our prior research efforts in mobile device security [7] which investigated the research question: *Can UI interaction and network activity be used to successfully predict and associate network flows with user actions on Android devices?* Given the resource constraints in mobile phones [8], this paper asks an associated research question: *What impact would endpoint sensors and SDN techniques have on CPU, memory, battery, and network delay on mobile devices?*

In exploring these research questions, we make the following contributions:

- **Create a UI-aware Android Network Access Control System:** We build a new application for Android OS, called APPJUDICATOR. It utilizes the built-in Android virtual private network (VPN) API to intercept network traffic and the accessibility service API to monitor UI interactions. It fuses UI information with SDN techniques to make an access control system based on user activity.
- **Characterize Network Profiling Potential for UI-aware Access Control Systems:** We investigate the UI-aware access control system's capability to execute real-time network profiling on mobile devices. We compare the UI-aware system with traditional access control systems relying on IP and DNS information. We configure these systems with dynamic allow-lists and examine their accuracy in network profiling. The results of our experiments show APPJUDICATOR outperforms other systems in network profiling by increasing the accuracy from 22.4% to 98.6%.
- **Evaluate the Performance of our APPJUDICATOR Prototype:** The costs of our system include the resource overhead of running the app (e.g., CPU cycles, battery power, memory consumption), additional network delay

for access control decisions, and resources in running a controller server. In our experiments, APPJUDICATOR adds 3.87 milliseconds to the median overall end-to-end round-trip time of intercepted packets. We anticipate that this delay will have only a minor impact on the user experience.

We organize this paper in the following structure. In Section II, we discuss the security issues and current solutions for mobile devices. In Section III, we explain our approach and design considerations. In Section IV, we present the results of evaluation regarding security effectiveness. In Section V, we evaluate the performance of the system in terms of resource usage and latency. In Section VI, we conclude with discussion.

II. BACKGROUND AND RELATED WORK

Our approach is designed to address security issues in mobile devices and is related to prior research in profiling systems, user interfaces, and end-point SDN techniques. To provide context for our work, we now provide background in each of these areas.

A. Security Issues and Mitigation for Mobile Devices

Mobile devices, particularly those personally-owned by employees, present a distinct challenge for IT managers in organizations. While these devices can significantly boost productivity, their remote usage and frequent network transitions introduce a delicate balance between security and user privacy. The balance remains a complex issue, as users seek autonomy and privacy while employers must manage risk. Recognizing this vulnerability, attackers have transformed mobile devices into weapons targeting both users and organizations.

The “Dresscode” malware is specifically designed to infiltrate corporate networks [9]. It disguises itself as a legitimate application in order to steal data and add infected devices to a botnet [10]. Once the infected devices are connected to the enterprise network, the malware can launch lateral propagation attacks. The “xHelper” malware can automatically download and install arbitrary software specified by an attacker, and persists in the infected device even after a factory reset [11]. An illicit market for malware-as-a-service called “Black Rose Lucy” even offers control of infected Android devices to paying customers, giving any malicious actor an entry point to a secure network [12]. Hazum et al. [13] found that a new variant of mobile malware quietly infected around 25 million devices. That “Agent Smith” pretends to be a Google-related application. Subsequently, it automatically replaces installed apps on the device with malicious versions without the user’s interaction. The “TrickMo” malware intercepts second-factor authentication codes from banks and transfers that information to a command-and-control server [14]. These second-factor codes are also used by organizations to protect sensitive transactions.

To mitigate these threats, prior work proposes a variety of perimeter, endpoint, and device management tools. These tools perform well in the specific scenarios, recognizing that no one solution is perfect. Perimeter defenses like intrusion detection systems, firewalls, and deep-packet inspection tools secure enterprise networks by analyzing network traffic. For example, Palo Alto Networks introduced a “next-generation firewall” [15] that allows policy customization for specific applications and user groups. However, policy creation for Palo Alto can be complex, requiring extensive training [16]. These systems base their behavior inference on packet headers and authentication server interactions, making them susceptible to adversarial manipulation [17].

Endpoint detection and response (EDR) tools like anti-virus software, endpoint firewalls, allow-lists, and behavioral analysis tools, offer detailed system insights. They monitor fine-grain interactions such as file accesses, socket events, and system calls. However, they are restricted by a high volume of false alarms, the overwhelming amount of low-level system logs, and the resource burden of log retention [18].

TABLE I
COMPARISON BETWEEN THE NETWORK ACCESS CONTROL OF THREE MDM SOLUTIONS

		Mobile Device Management Solutions		
Scope	Information Used	APPJUDICATOR	Microsoft Intune	IBM MaaS360
Device	Hostname	✓	✓	✓
	Port	✓	✗	✓
	UI Interaction	✓	✗	✗
	User Membership	✗	✓	✗
	Network Interface	✗	✗	✓
	Device Compliance	✗	✓	✗
Application	Package Name	✓	✗	✓
	Application Role	✓	✓	✓

Microsoft’s Intune [19] offers both mobile application management (MAM) for individual app protection and mobile device management (MDM) for device-wide settings. MAM safeguards specific apps, leaving other apps at risk, while MDM secures the entire device and could breach more user privacy. IBM MaaS360 [20] also offers MDM solutions for Android devices. Using the tools’ documentation, we constructed Table I to compare features of these tools with APPJUDICATOR. Where the tools described functionality, we provide check mark. We mark an X when there is no evidence that the feature is supported. APPJUDICATOR was the first MDM system on Android to leverages UI interactions to control network access for IoT devices work [21]. APPJUDICATOR filters unauthorized traffic and enforces network policy with the goal of offering a middle ground for employee-owned devices that respects end-user control and privacy. The results of our effectiveness and performance evaluation on APPJUDICATOR show that it improves the accuracy of the access control system while adding a minimal latency to the end-to-end delay.

B. Network Profiling Systems on Android

Previous research has demonstrated the value of analyzing communication patterns on Android. A recent paper [22] examines the effectiveness of endpoint network logging by exploring end-to-end network paths using browser data to distinguish between on-path failures and attacks. Netsight [23] applies network profiling to help locate root causes related to network failures. ProfileDroid [24] profiles Android network traffic to identify privacy issues. They distinguish ad-related

traffic from application traffic by leveraging coarse-grain user-level information.

While network profiling works well on Android devices, the requirement of network interception via root privileges in the Android OS increases potential security risks [25]. NetGuard [26] and NoRoot [27] bypass this requirement by leveraging the Android VPN service [28] to forward local application traffic to a proxy program that offers a firewall and profiling service. Meddle [29] also utilizes the VPN interface to inspect remote network traffic. Our work similarly applies the VPN service as a local proxy to intercept Android network traffic without root privileges. We further extract users' actions from UI information and allow the network profiling system to identify the network flow initiated by users.

Network logging on mobile devices has been used in network analysis [30]. Sipola et al. [31] propose a method that implements deep learning algorithms in network logs to detect anomalous behaviors. However, network logging is hindered by limited types of log source. Our work monitors and adds UI information to traditional network logging, providing context for network analysis.

Android also supports the profiling of local system activities. Lanzi et al. [32] reconstruct user-initiated actions by analyzing system calls in Android system log. The limitation of system log analysis is the requirement for root privileges or a debugger to obtain the full logs. AppContext [33] analyzes system logs to identify patterns associated with malicious applications. Since our goal is to build a real-time profiling system for Android OS, such later analysis is not viable.

C. UI and Accessibility Services

The user interface plays an important role in mobile devices. Users interact with the interface to conduct almost every operation in mobile devices. Therefore, researchers can use such user interactions with mobile devices to construct the user context. As an example, GUILeak [34] tracks user inputs and interactions with the device to identify potential privacy violations based on vendors' privacy policies. AppIntent [35] leverages the information from user interactions to conduct static analysis on data transmission activities. However, these methods are not applicable in a real-time network system. SmartDroid [36] shows the effectiveness of linking user interactions with system calls. As mentioned earlier, the methods to intercept system calls require root privileges which would impede deployment.

Prior efforts have used the UI with traces and logs to help identify application defects and to localize issues [37]–[40]. Software testing has widely implemented automatic UI operations. `Appium` [41] is designed to facilitate UI automation of applications mobile devices. We use `Appium` for automatic tests in experiments. The `Monkey` [42] tool is a program that runs on mobile devices and generates pseudo-random streams of user events, such as clicks and gestures, which help developers to stress-test applications.

The Android accessibility service API [43] was originally designed to assist users with disabilities. When it runs in the background, it receives accessibility events that indicate state transitions in the user interface, such as the focus changing or a button being clicked. Prior work [21] proposes a method that analyzes UI events from the accessibility service to secure smart home devices. In our approach, we recognize the value of UI information captured from the accessibility service and leverage that context for network profiling in mobile devices.

D. Host-based Software-Defined Networking

SDN is proposed to divide the traditional network routing platform into a control plane and a data plane. The control plane is required to communicate with the data plane through customized protocols. The OpenFlow [44] protocol enables switches to intercept network traffic and act as an SDN agent. The agent maintains a local flow table that matches conditions with associated actions. When the SDN agent finds a match in the local flow table, it processes the packets according to the associated action rules. Otherwise, it elevates the packet to a remote SDN controller for advice through a `PACKET_IN` message. The SDN controller analyzes the packet and determines the appropriate action for it. The controller commands the local SDN agent to install a rule through a `FLOW_MOD` message. It then sends an order for the specifically elevated packet back to the SDN agent. If a `FLOW_MOD` is used to install a rule, the SDN agent will apply that rule locally to subsequent packets in the flow.

Prior efforts have broadly and deeply explored SDN implementations in enterprise networks [45]–[48] and residential networks [49]–[51]. These works build the SDN agent in switches that connect with several endpoints. These switch-based SDN agents send aggregated network information to the control plane and overlook device-level information. Further, several works propose host-based SDN agents in endpoint systems [6], [52], [53]. These host-based SDN agents can monitor system logs in endpoints and elevate them to the control plane. Lei et al. [54] show how correlated signals from endpoints can reveal compromises on those endpoint devices.

Researchers have also explored SDN agents for Android systems. PBS-Droid [5] and meSDN [4] show the prototypes of SDN agents in the kernel of Android, which requires the deployer to recompile the operating system. HanGuard [55] implements SDN agents on residential routers and Android devices to build an access control system in home networks. However, these implementations require advanced end-user actions, which impede deployment. In our approach, we avoid the device rooting requirement and build the SDN agent as an independent and easy-to-use application for Android devices.

A well-known issue with SDN is the risk of a flow table overloading attack [56], in which an attacker generates malicious network flows that prompt continuous rule installation on the flow table. In our proposed approach, the flow table overloading attack would only affect one device since the flow table is device-specific. Further, we can track the application that is creating the majority of flow table entries. This isolates the attack to a single application and lets us warn the user that the application is problematic.

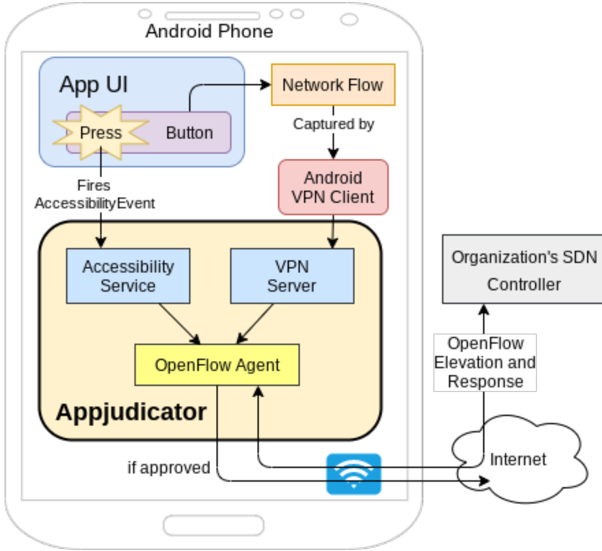

Fig. 1. APPJUDICATOR sends UI and network data to an OpenFlow agent.

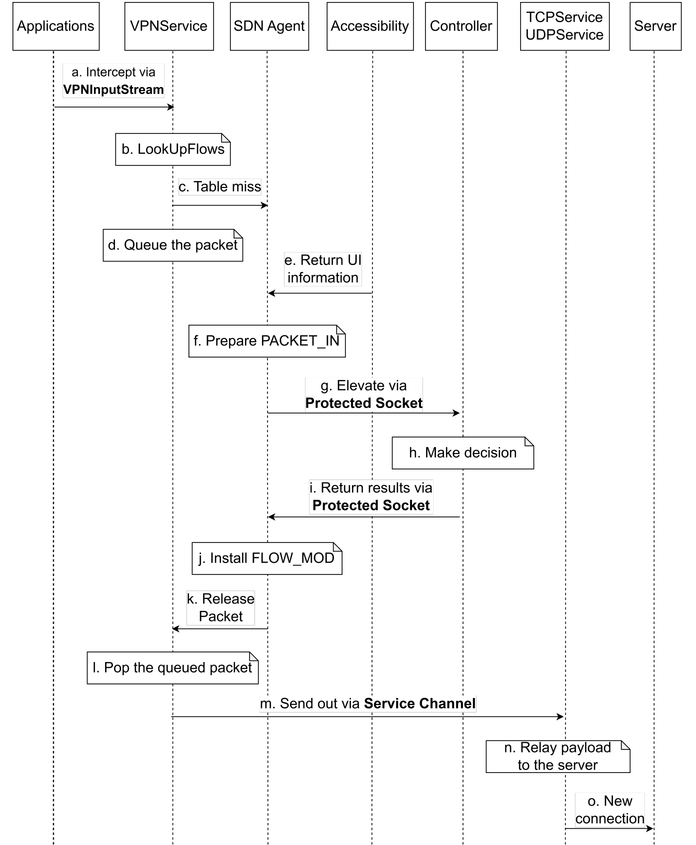

Fig. 2. Sequence diagram of APPJUDICATOR outbound packet processing

III. APPROACH: FUSION OF NETWORK, UI, AND SDN

We combine sensors for the user interface and the network at the mobile device endpoint. Since the Android OS dominates the market share for mobile devices, we use that operating system for our prototype. As we build sensors, we focus on options that can be accomplished using existing APIs and standard application installation procedures rather than requiring steps that would hinder deployability, such as rooting a device or recompiling the operating system.

This section introduces each component in our architecture. A proxy is required to intercept the network traffic of an Android device without root privileges [57]. It acts as an on-path entity between the Android device and remote servers. We use a VPN service (Section III-B) to intercept the packets between applications and remote application servers without rooting a device. The VPN service works with an SDN module (Section III-D). Our SDN agent fuses the captured network flows with UI information provided by an Accessibility Services module (Section III-C). The SDN agent works with a remote SDN controller that analyzes the flows and makes forwarding decisions for the network flows. Figure 1 visually depicts the components of our system and how they relate to each other.

A. Threat Model

We assume mobile devices may use various networks including residential networks, enterprise networks, and cellular provider networks. We assume mobile devices are mixed-use, with personal and work-related usage patterns. When used for work, a compromised device can serve as an entry point for attackers. Some smartphone malware opportunistically targets enterprise network assets when the mobile device connects to those networks [58]. Additionally, some malware remotely attacks enterprise assets, such as email servers, by utilizing credentials stored on the device.

In this work, we consider a personally-owned, mixed-use device with cross-application workflows. As an example, we consider an attacker that targets a victim, Company A, with a phishing attack disguised as a benefits update from Company X. The attacker sends a phishing email prompting the employees to install a new mobile application, which is actually malware. An employee installs the application on their personal Android phone, granting permissions. The malware replaces legitimate applications with Trojan versions to steal authentication values. This allows the attacker to access Company A's resources when the compromised device connects to the enterprise network.

B. Intercepting Android Traffic via the Built-in VPN Service

The Android built-in VPN service [28] has been used by prior works, described in Section II-B, to create a proxy application that intercepts network traffic in Android OS. Such a method is applicable to users since it does not require root privileges of Android devices. We implement the VPN service as a key component of APPJUDICATOR.

We first open an interface from the VPN service API. The interface works as the VPN server with a local address. The application that attempts to access the Internet works as the VPN client and must establish two streams with the VPN server. One is a `VPNInputStream` that forwards packets from applications to the VPN server. The other is a `VPNOutputStream` that the VPN server uses to sends response packets back. In Figure 2, we show the process of an application accessing the Internet when APPJUDICATOR is enabled. When the VPN server receives a packet from a local application, it makes a query via the SDN agent. If the packet

TABLE II
DESCRIPTIONS OF EACH STEP OF APPJUDICATOR OUTBOUND PACKET PROCESSING IN FIGURE 2

Step	Involved Module(s)	Description
a	Applications, VPN Service	The VPN Service uses the VPNInputStream to intercept network flows from all applications.
b	VPN Service	The VPN Service checks if the network flow matches any entry in the flow table.
c	VPN Service, SDN Agent	When a flow is not a match for any flow table entries, the VPN Service invokes the SDN Agent.
d	VPN Service	The VPN Service queues the flow's packet(s) and waits for a decision.
e	SDN Agent, Accessibility Service	The Accessibility Service sends related application UI information to the SDN Agent.
f	SDN Agent	The SDN Agent compiles the flow and UI information into a PACKET_IN message.
g	SDN Agent, SDN Controller	The SDN Agent sends the PACKET_IN message to the SDN Controller.
h	SDN Controller	The SDN Controller makes an allow or deny decision based on the flow and UI information.
i	SDN Controller, SDN Agent	The SDN Controller sends the decision message to SDN Agent.
j	SDN Agent	The SDN Agent adds a rule to the flow table to implement the action from the decision message.
k	SDN Agent, VPN Service	The SDN Agent informs VPN Service to release any queued packets associated with the flow.
l	VPN Service	The VPN Service dequeues the associated flow packet(s). It discards packets for a deny rule.
m	VPN Service, TCP/UDP Service	For allowed flows, the VPN Service sends the flow packet(s) to the TCP/UDP Service.
n	TCP/UDP Service	The TCP/UDP Service relays the payload of the flow packet.
o	TCP/UDP Service, Remote Server	The TCP/UDP Service starts a new connection with the Remote Server.

matches a table entry in the existing flow table of the SDN agent, it is forwarded to the designated service that connected to the destination (step m in Figure 2). If not, the packet is queued and the SDN agent queries a remote controller via the OpenFlow protocol (steps d through i in Figure 2).

As shown in steps n and o in Figure 2, when a TCP packet passes the SDN agent, it goes to the `TCPService`. There is also a respective `UDPService` in APPJUDICATOR for UDP packets. The `TCPService` forwards the TCP packets to the remote server via a `protected socket`. In doing so, it ensures that the traffic from this socket is not be intercepted by the VPN service, avoiding a potential loop. The Android VPN service API avoids the typical cryptographic or tunneling overheads associated with a VPN since the Android OS allows our application to simply read and write packets directly using a `ParcelFileDescriptor` instance. This allows us to use standard packet parsing libraries (e.g., `Pcap4J`).

Finally, when several remote servers send response packets back to the device, the VPN service needs to distribute each packet to the appropriate application. We achieve this using the user identity (UID) that is unique for every Android application. We leverage the Android `ConnectivityManager` and `PackageManager` APIs. In doing so, we first aggregate into flows with the five tuple (IP_{src}, IP_{dest}, protocol, $Port_{src}$, $Port_{dest}$). Then we use `ConnectivityManager` to find the UID associated with the flow by (IP_{src}, protocol, $Port_{dest}$) information. We further use `PackageManager` to get the application name and context related to the UID. With all of these components, including the VPN services, the SDN agent, and the `TCPService`, we compile a proxy program that operates in the middle between applications and their accessing destinations. Such a proxy program can conduct network profiling and access control by elevating network and potential UI information to a remote SDN controller.

C. Extracting UI events from Accessibility Services

The Android accessibility service API [43] provides the capability for querying the content of the active window and UI interactions. It is designed to assist users. As an example, a screen reader tool that uses the accessibility service can help visually impaired users. In this work, we aim to apply the accessibility service to analyze UI events on the mobile device in order to distinguish the user-initiated network activities.

Specifically, we import the accessibility service to APPJUDICATOR in a `Manifest` file and configure the permissions of `BIND_ACCESSIBILITY_SERVICE` and `canRetrieveWindowContent`. The accessibility service then runs in the background and receives callbacks from Android OS when accessibility events are generated. With the `intent-filter`, we can further decide to monitor specific applications. To eliminate potential privacy concerns, users can choose the scope of applications be tracked by the accessibility service.

The accessibility events denote some state transitions in the user interface, such as clicking buttons, acting gestures, changing focus, etc. In our prototype, we use the accessibility service to track all types of accessibility events. APPJUDICATOR can receive an overall context for every UI event. In doing so, it retrieves the hierarchy information from the `AccessibilityEvent.getSource()` function. That function provides both the current text of the UI event and detailed context information, such as the parent and child widgets of the UI event. For example, when a user opens a browser application and inputs a specific URL to access, a UI event, `TYPE_VIEW_TEXT_SELECTION_CHANGED`, is sent. APPJUDICATOR can extract the user inputs from the contents of an `EditText` UI object. APPJUDICATOR can also correlate the user inputs with the following network flows and achieve more accurate network profiling.

With the UI information, we can characterize the user interface and provide this data to the SDN controller. However, one challenge is to generate unique identifiers for UI information. Some UI elements have a unique resource identifier, and others do not. Accordingly, we use an established heuristic of combining the resource ID (if it exists), the hierarchical path between the UI element and the root widget in the application, and the properties of the UI element itself (class type and label text, if any). APPJUDICATOR has the capability of intercepting the network traffic and monitoring the UI events in the scope of selected applications. We apply the SDN paradigm to fuse the network and UI information to a comprehensive access control system.

D. Fusing Network Data and UI events via SDN

We create a host-based SDN agent module in APPJUDICATOR. The agent augments the network flow with additional UI information in the query message. It communicates with a remote SDN controller using a subset of the OpenFlow v1.0 specification [44]. We omit the implementation of VLANs and the physical ports since the SDN agent typically handles network operations in a single mobile device. As with traditional SDN agents, the APPJUDICATOR agent supports wildcards in the rule match fields and matches the closest entry in a flow table. The VPN service retrieves the correlated action from the SDN agent, and promptly handles queued packets and associated following packets.

When a new packet is not matched by any existing flow table rules, the SDN agent sends a query message to a remote controller. Such messages include the information about the original packet and its correlated UI information. The controller uses the UI information to build the context surrounding the sent packet. The controller replies with an action decision based on its policies. We illustrate three types of policies in Section IV. Our prototype currently only supports two action types: drop and forward, which perform key operations in an access control system.

Our `PACKET_IN` message adds UI information to the query message. APPJUDICATOR extracts UI events from the accessibility service and provides it to the SDN agent. The agent attaches the UI information to the `PACKET_IN` message in reverse-chronological order. The agent omits some types of UI events. As an example, the UI event of `TYPE_VIEW_CONTENT_CHANGED` is very common in the Android OS and provides little useful information. Instead, the SDN agent elevates the types of UI events that are most likely to reveal user intent to the controller.

Our `PACKET_IN` message contains an OpenFlow header, the encapsulation of the original packet, and as much UI information as possible while keeping the entire `PACKET_IN` message within 1500 bytes to avoid packet segmentation. The OpenFlow agent then sends the message to the remote SDN controller. We position the remote SDN controller in a server at our organization, which minimizes latency due to the short distance. According to prior research [59], 90% of residential users can reach potential OpenFlow controllers within a 50 millisecond round trip time, which causes a 2 second increase in web page loading time. This delay can be improved with the deployment of more computing nodes that are positioned closer to the SDN agents.

We have described three major components of APPJUDICATOR: the VPN service for intercepting network traffic, the accessibility service for extracting UI interactions, and the SDN paradigm. APPJUDICATOR can be a comprehensive network profiling and access control system. Next, we evaluate APPJUDICATOR's effectiveness and performance.

IV. EFFECTIVENESS EVALUATION: NETWORK PROFILING

We aim to answer the following research question: *Can UI interaction and network activity be used to successfully predict and associate network flows with user actions on Android devices?* While exploring the role of the UI context of APPJUDICATOR in network profiling, we form a hypothesis that insight into the end-user's actions can improve network traffic monitoring. We propose a UI-aware sensor that distinguishes user-initiated network flows. We assume that stealthy malicious connections in the background can be identified by the lack of UI activity. In doing so, we compare our proposed UI-aware sensor with two network-based sensors commonly used in enterprise networks. These sensors have respective policies that decide which network flow is allowed. From a series of experiments with both legitimate and malicious data, we determine the accuracy rate of these sensors in network profiling. We consider three sensors that originate from the allow-list policy:

- **IP Header Sensor**: The sensor uses policy built upon the IP header information of the packets during the idle and training phases. These policy rules allow a new packet based on whether the packet matches the same IP address and port number as seen previously.
- **DNS-aware Sensor**: The sensor incorporates IP header and DNS data from DNS queries information during the idle and training phases. These policy rules allow a new packet based on whether the packet matches the same port number, host name, or IP address if no host name.
- **UI-aware Sensor**: The sensor incorporates IP header and DNS headers, and uses UI policy built upon the UI interactions during the idle and training phases. These policy rules allow a new packet based on whether the packet matches the same IP address, port number, host name, and UI context.

We first explore the background activities of tested applications. We open these applications and leave them running and idle for two hours. All of the tested applications attempt to connect to vendor servers while the user only opens them and leaves them idle. We record all of the network activities that happen in the idle period in the `idle data` set.

We then conduct a training phase data collection by using `Appium` [41], which is an automation framework for software testing. `Appium` simulates user interactions with tested applications and invokes a series of UI events captured by Android accessibility service [43]. We select several types of UI events and use `Appium` to automatically execute a set of actions. As an example, `Appium` can open YouTube and click the symbol of searching with the UI event of `TYPE_TOUCH_INTERACTION_START`. It then inputs text characters into the search box with a UI event of `TYPE_VIEW_TEXT_SELECTION_CHANGED` and clicks the search button which incurs the UI event of `TYPE_VIEW_CLICKED` and subsequent network flows. When `Appium` conducts these operations, the user can observe it on the device screen. All of the network traffic and UI interactions in the above simulation have been collected as the `Training Data`.

We choose several commonly used applications in our evaluation: Gmail, YouTube, Chrome, Firefox and Termius. These applications cover a broad range of usage scenarios including email, video, browser and SSH terminal. For col-

TABLE III
ALLOW-LIST MATCH RATE WITH IP, DNS, AND UI SENSORS FOR LEGITIMATE, APPIUM-SUPPLIED DESTINATIONS. THE DNS ALLOW-LISTS CAN SUPPORT FIXED-DESTINATION APPLICATIONS (E.G., IN GMAIL AND YOUTUBE). ONLY THE UI SENSOR EFFECTIVELY ALLOWS DESTINATIONS THAT ARE DYNAMICALLY SPECIFIED (E.G., IN TERMIUS).

	Data Samples Count			Allow-List Match Rate		
Application	Idle	Train	Test	IP Sensor	DNS Sensor	UI Sensor
Termius	236	1,587	1,669	6.6%	6.9%	99.1%
Gmail	180	1,052	1,313	43.6%	98.8%	100.0%
YouTube	315	1,162	1,845	57.3%	100.0%	100.0%

lecting the `Training Data`, we create thirty workflows of `Appium` scripts for Gmail and YouTube. We note that they tend to communicate with a small set of servers that could be associated with vendors or advertisers. While Gmail and YouTube have relatively fixed destinations, Chrome, Firefox, and Termius have more flexible destinations specified by users. For these three applications, we develop a workflow that randomly accesses different destinations in a website list. We import the Top 500 websites list from `SimilarWeb` [60] and separate them into training and testing lists.

In the testing phase, we use `Appium` to operate the mobile device with the same scripts used in the training phase, while the remote controller deploys three types of sensors based on different policies generated from the idle and training data. Since the destinations of browser and SSH client are dynamically changed by users, the training data for the IP and DNS sensors is unlikely to include all future user-supplied destinations. Therefore, we expect the UI-aware sensor to address the issue by leveraging UI context data.

Table III shows the accuracy rates of the three sensors for Termius, Gmail, and YouTube with legitimate traffic. The legitimate traffic is tested in the same way as during the training phases and initiated by `Appium` to simulate user behaviors. An optimal sensor would achieve 100% match rates for such traffic. In out tests, the IP sensor has the lowest accuracy for these applications, possibly due to the influences of DNS load balancing or the effects of content distribution network (CDN) deployments. The DNS sensor performs well for Gmail and YouTube, achieving an accuracy rate higher than 98.8%. These results show that the awareness of DNS host name can dramatically increase the accuracy of network profiling for fixed destination applications. But since the policy rules generated from `Training Data` could not predict dynamic user inputs, both the DNS sensor and IP sensor perform worse with the SSH client, only reaching an accuracy rate between 6.6% to 6.9%. In contrast, the proposed UI-aware sensor achieves an accuracy rate above 99.1% for all tested applications. This difference highlights the importance of understanding the user-specified destinations when profiling traffic.

The denial of legitimate traffic is a significant concern for allow-lists and is a primary weakness for our technique. However, an allow-list that permits all traffic would score well in experiments if we only considered legitimate traffic. Accordingly, we next explore our proposed techniques and sensors when adding records that simulate malicious traffic when testing Chrome and Firefox. According to Rejthar [61] and Avast researchers [62], there are several malicious browser extensions, such as Video Downloader for Facebook and Instagram Story Downloader, which stealthily access URLs that host malware on popular cloud providers. We use the reported URLs to construct a set of similar URLs using a Python script. To simulate network connections from those browsers, we generate connection records purportedly from those web browsers to the malicious destinations, and insert these records into the controller's database prior to the controller's profiling step. This allows us to mimic malicious behavior in order to evaluate the controller's ability to detect the faux malicious traffic with different sensors.

In our analysis, we only examine the first connection triggered by each web page access. Typically web page retrievals download multiple resources, such as images, in response to inclusions in a web document. This analysis mimics allowing all traffic for a short duration from that application after an initial match. In Section VI-A, we discuss the additional instrumentation needed to address these limitations.

In Table IV, we show the accuracy rate in four scenarios using a confusion matrix. A sensor leads to a "correct decision" whenever it correctly matches (or allows) a legitimate flow and when it correctly denies a malicious flow. An incorrect decision occurs whenever the sensor allows a malicious flow or denies a legitimate one. We present the results as percentages. For legitimate traffic, the percentages of correctly allowing traffic and incorrectly denying traffic total to 100%. Likewise, the percentages of malicious traffic correctly denied and incorrectly allowed total to 100%. The overall accuracy of the sensor can be characterized by the relative amounts in the correct decision columns verses the amounts associated with incorrect decisions.

TABLE IV
SENSOR CLASSIFICATION ACCURACY FOR CHROME AND FIREFOX WITH MIX OF LEGITIMATE ("GOOD") AND MALICIOUS ("BAD") TEST FLOWS.

	Number of Data Samples				Correct Decision		Incorrect Decision	
App.	Idle	Train	Testing	Sensor	Deny	Allow	Deny	Allow
Chrome	207	1,658	3,593 (1,811 good, 1,782 bad)	IP	13.1%	4.8%	95.2%	86.9%
				DNS	19.8%	8.7%	91.3%	80.2%
				UI	100.0%	98.6%	1.4%	0.0%
Firefox	280	1,428	3,140 (1,506 good, 1,634 bad)	IP	18.9%	5.3%	94.7%	81.1%
				DNS	22.4%	9.6%	90.4%	77.6%
				UI	100.0%	99.4%	0.6%	0.0%

The IP sensor and DNS sensor have low overall accuracy, correctly allowing less than 10% of the legitimate traffic that includes user-supplied destinations. They correctly deny only up to 22.4% of the malicious traffic, since the malicious URLs often used IP addresses and host names in the training data as the malicious destinations were co-hosted on popular cloud servers. The UI sensor correctly denies 100% of the malicious traffic since that traffic does not have the associated UI activity required by the UI sensor's matching rules. Moreover, the UI sensor correctly allows over 98.5% of legitimate traffic. The legitimate traffic analyzed in Table IV only includes the initially requested destination. These results suggest that a UI sensor can aid network profiling and allow-list systems.

V. Performance Evaluation

We explore the performance implications of APPJUDICATOR via the Android Studio Android emulator of a Google Pixel 4 smartphone with four cores, 2 GBytes of memory, and Android API level 30. The emulator is hosted by Android Studio Bumblebee 2021.1.1 Patch 2 installed on a laptop with six 2.6 GHz cores and 16 GBytes of memory connecting via WiFi to a router that provides residential Internet access. We evaluate the performance in three metrics: (a) networking round trip time measured by using the emulator to access a remote server running Ubuntu 20.04 hosted on a public network server, (b) web traffic performance examined by using Firefox 117.1.0 installed on the emulator, and (c) computational resources collected by the Android Studio Profiler.

A. Network Round Trip Time (RTT)

Since our local VPN service intercepts all outgoing and incoming packets as they traverse the VPN service, it necessarily adds networking delay. To examine how much delay APPJUDICATOR adds, we record the overall end-to-end RTT between an application running in the Android emulator and a remote server on a public network. We perform the experiments on a simulated Google Pixel 4 phone with Android API level 30 and an Ubuntu 20.04 server. We run our SDN controller on a separate virtual machine on the same local network as the Android emulator. Such virtual machine is hosted by a separate physical device from the machine running the Android emulator. We use the remote server to periodically send packets with tailored payloads to the application on the emulator and have the application simply echo it back. We compare the scenarios when APPJUDICATOR is enabled and disabled. Since the variable in these cases is on emulator side, we introduce a timer on the remote server side to record the overall end-to-end delay.

We measure the overall end-to-end RTT by recording two timestamps at the remote server. The first timestamp is the time when a packet with a specific payload is sent from the remote server (t_1). The second is the time when the remote server receives the packet with the same payload that the application echos back (t_2). We continuously repeat this process with a packet arrival rate that is around 50 packets per second. When APPJUDICATOR is enabled, the delay between these two timestamps includes any delays resulting from the consultation with the SDN agent, the UI Monitor and the VPN service (as described in Section III). Accordingly, we consider the difference between t_2 and t_1 as the overall end-to-end RTT between the application on our emulator and the remote server. To compare between APPJUDICATOR being enabled and disabled, we can evaluate the total additional delay added by APPJUDICATOR.

In our experiments, we collect the timestamps over 1,000 trials. As shown by the blue (left) line in Figure 3, the average overall end-to-end RTT of packets with APPJUDICATOR disabled is 19.06 ms and the median is 15.86 ms, which accounts for the networking delays between our local network and the remote server. The green (right) line shows the overall end-to-end RTT with APPJUDICATOR enabled. The average

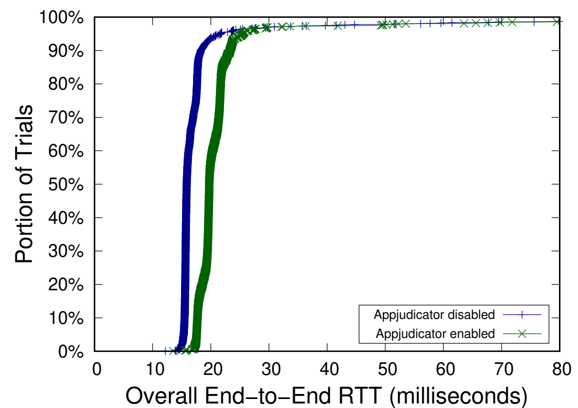

Fig. 3. CDF of overall end-to-end round-trip time (RTT) when APPJUDICATOR is enabled and disabled.

RTT of packets is 23.23 ms and the median is 19.73 ms. For transferring network packets between the application on the emulator and the remote server on a public network, APPJUDICATOR adds 3.87 ms to the median overall end-to-end RTT.

B. Web Traffic Performance

In addition to measuring the delay added to a connected TCP flow, we also examine how much delay APPJUDICATOR adds to the web page load time when the user tries to access a new website. As stated in Section III, the first packet of each new outgoing flow must wait for context from the UI Monitor, elevation to the SDN controller, and installation of the rule by the local Android SDN agent. We explore the web page load time to evaluate the delay caused by these operations.

We conduct these experiments in the same Android emulator and network configuration as in Section V-A. We use `Appium` to operate `Firefox` to access the Top 100 domains listed on `Cloudflare Radar` [63]. The `Appium` script has been repeatedly running to collect over 1,000 data points each for the cases where APPJUDICATOR is enabled and disabled. We implement `Firefox Profiler` [64] to capture performance files directly from Android Firefox browser. After parsing the performance files, we measure the web page load time by analyzing the timestamps of network events.

Figure 4 shows that when we use the browser without APPJUDICATOR running, the average page load time for the Top 100 domains is 448.07 ms. After we enable APPJUDICATOR, the average page load time increases to 765.92 ms. We note that APPJUDICATOR would add about 317.85 ms delay to the complete loading of new web pages that require an SDN consultation for each new network request. Web page retrievals often incorporate a large number of individual web requests; these results represent the overheads of each new flow review in aggregate.

We also conduct experiments to evaluate the time cost of packet elevation to the SDN controller. The controller runs a Python program that responds to each `PACKET_IN`. It extracts any included UI information from the `PACKET_IN` messages

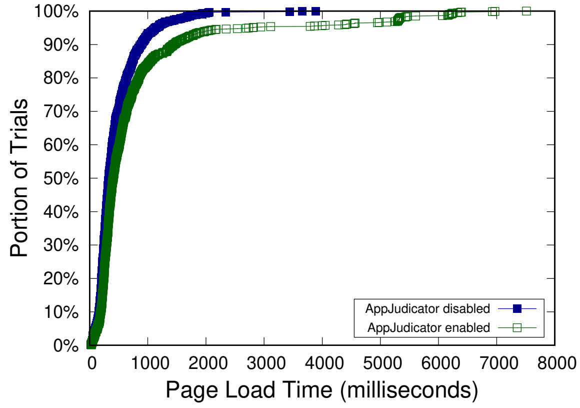

Fig. 4. CDF of web page load time with APPJUDICATOR enabled and disabled

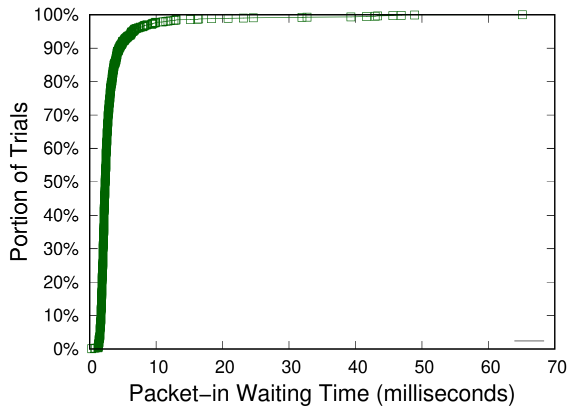

Fig. 5. CDF of waiting time between the SDN agent sending out a packet-in message and receiving a response from the SDN controller

and sends its decision back to the SDN agent. We measure the total waiting time as the time difference between the SDN agent sending out a PACKET_IN and receiving a response, which includes the networking delays in the local network and the processing time of the SDN controller. During the experiments, we record over 1,000 data points of PACKET_IN waiting time when the Appium script manipulates Firefox to access the Top 100 websites. In Figure 5, the results show that 90% of waiting time data points are less than 4.50 ms and the average waiting time is 3.24 ms.

C. Computational Resources

Energy consumption is a significant consideration for mobile devices. Accordingly, we evaluate battery consumption, memory usage and CPU usage of APPJUDICATOR on the Android emulator. Our tests use Android Studio's Profiler [65] and the top tool to gather data. We emulate a Google Pixel 4 device with an Android API level of 30 in an Android emulator with 4 cores and 2 GBytes of memory.

During the experiments, we first start APPJUDICATOR and then play music in the YouTube Music application to simulate a common use case in the device. The music application is set to stream music from the Internet, requiring network transmissions to obtain the music. We use the monitoring tools to record the CPU usage of APPJUDICATOR and YouTube Music. The monitoring tools record a data point for each second with an experiment duration of 1,000 seconds.

TABLE V
CPU AND RAM USAGE OF APPJUDICATOR AND YOUTUBE MUSIC

Application	Resource	Percentile of Trials		
		10th	50th	90th
APPJUDICATOR	CPU usage (%)	11.0	12.0	14.0
	RAM usage (MBytes)	130.7	132.7	132.7
YouTube Music	CPU usage (%)	29.0	33.0	36.0
	RAM usage (MBytes)	255.5	263.4	273.3

As shown in Table V, when APPJUDICATOR runs on the virtual Android device, it consumes a relatively constant amount of memory, with an average of 132.7 MBytes. It uses less memory than the YouTube Music application, which uses an average of 264.5 MBytes memory. The CPU consumption of APPJUDICATOR averages around 12.5%, which is lower than the 32.7% average CPU usage of YouTube Music. Android Studio's Profiler classifies the energy consumption of APPJUDICATOR as "light". Given these results, APPJUDICATOR seems to have performance characteristics that are significantly less demanding than a popular Android application.

TABLE VI
COMPARISON BETWEEN APPJUDICATOR AND OTHER ACCESS CONTROL SYSTEM USING SDN

	HARBINGER [6]	APPJUDICATOR
Accuracy rate	99.1%	98.6%
Network connection	Ethernet	WiFi
Added web page load time	Around 600 ms	317.85 ms
Added end-to-end RTT	Around 6 ms	3.87 ms

To provide context for the performance of APPJUDICATOR, we compare the approach to HARBINGER [6], which is an SDN access control system built for the Microsoft Windows OS. We compare the approaches across evaluation metrics reported for both tools. Table VI shows APPJUDICATOR has slightly lower matching accuracy than HARBINGER and APPJUDICATOR introduces less delays to both web page load time and end-to-end RTT.

VI. DISCUSSION

We now describe how our approach relates to two key concepts: web page connection dependencies and privacy.

A. Support for Web Page Dependencies

Web browsers have different characteristics than most other mobile applications. A website visit often results in connections to multiple servers to load resources specified in a web document. Since websites often employ end-to-end encryption, a middlebox typically cannot predict what resources are needed to complete a specific page load in advance. When a user initiates a web access by clicking the screen, only the initial web request is elevated to the controller along with the UI interaction. As a result, the profiling we describe in Section IV is limited to profiling the destination of the initial

web request. To fully achieve the goal of profiling dependent web requests, our proposed approach needs additional sensing functionality.

Some web browsers have symmetric key export functionality [66] that allows external applications (like Wireshark) to decrypt TLS-protected communication. However, this functionality typically exists on traditional desktop operating systems and it not widespread on mobile devices. Techniques to hook specific library calls, such as the `libssl.so` library's `SSL_read` and `SSL_write` functions, with tools such as Frida [67], enable access to the plaintext communication associated with the application. However, these tools require root access to the phone's operating system, which introduces challenges for deployment and security.

Alternatively, the use of a trusted device-wide root certificate can enable interception of TLS communication in the VPN service module, enabling examination of unencrypted communication. The use of root certificates are typically discouraged since they break the end-to-end encryption model; however, positioning the interception and decryption on the endpoint device itself may offset some concerns. With such an approach, the web browser would no longer be able to see the TLS certificate associated with the remote server, since it would only see the certificate the VPN service module presents. Accordingly, the VPN service module would need to perform appropriate certificate validation of the remote server's certificate, much like the prior work in TLS-Deputy [68].

B. Privacy Implications

The use of an SDN controller with the smartphone grants a third-party service the ability to perform network profiling and management. For corporate-owned devices in which the user agrees to monitoring, privacy may not be an issue. However, for other use cases, this monitoring may significantly affect the privacy of the device user while running APPJUDICATOR. We now explore these concerns and mechanisms that can help manage these risks.

As engineering efforts with APPJUDICATOR continue, we will explore creating a controller on the phone that synchronizes policy with an organization's main controllers. When the controller is unreachable from the phone, the phone-based controller can then make access control decisions with the local cached policy. To grant visibility to the organization, the phone's controller could report connectivity information to the organization controller, indicating what policy it applied but without supplying the UI sensor's detailed records. This would allow an organization controller to know that a flow was related to a user's action, without visibility into the details.

Another option would be to consult different controllers for different applications on a phone. For applications associated with an organization, the phone may consult the organization's controllers. For all other applications, the phone may contact a controller associated with an end-user-designated service provider. This model would allow users to split their phone automatically into roles associated with their profession and their personal life.

C. Concluding Remarks

In this work, we propose and evaluate APPJUDICATOR, an SDN system for mobile devices that associates UI elements with network flows. With the ability to consult external SDN controllers for assistance, the APPJUDICATOR tool can respond to evolving threats while providing sufficient context for access control decisions. This mechanism helps increase the accuracy of network profiling from 22.4% to 98.6%. In our evaluation, we find that APPJUDICATOR consumes acceptable computation resources while introducing modest network delay.

REFERENCES

[1] C. Rotsos, D. King, A. Farshad, J. Bird, L. Fawcett, N. Georgalas, M. Gunkel, K. Shiomoto, A. Wang, A. Mauthe *et al.*, "Network service orchestration standardization: A technology survey," *Computer Standards & Interfaces*, vol. 54, pp. 203–215, 2017.

[2] Q. Wang, W. U. Hassan, A. Bates, and C. Gunter, "Fear and logging in the internet of things," in *Network and Distributed Systems Symposium*, 2018.

[3] Google, "Security risks with modified (rooted) Android versions," 2020. [Online]. Available: https://support.google.com/accounts/answer/9211246?hl=en

[4] J. Lee, M. Uddin, J. Tourrilhes, S. Sen, S. Banerjee, M. Arndt, K.-H. Kim, and T. Nadeem, "meSDN: Mobile extension of SDN," in *Workshop on Mobile Cloud Computing & Services*, 2014.

[5] S. Hong, R. Baykov, L. Xu, S. Nadimpalli, and G. Gu, "Towards sdn-defined programmable byod (bring your own device) security." in *NDSS*, 2016.

[6] Z. Chuluundorj, C. R. Taylor, R. J. Walls, and C. A. Shue, "Can the user help? leveraging user actions for network profiling," in *IEEE International Conference on Software Defined Systems (SDS)*, 2021.

[7] S. Liu, J. P. Petitti, Y. Lei, Y. Liu, and C. A. Shue, "By your command: Extracting the user actions that create network flows in android," in *2023 14th International Conference on Network of the Future (NoF)*, 2023, pp. 118–122.

[8] W. Enck, P. Gilbert, S. Han, V. Tendulkar, B.-G. Chun, L. P. Cox, J. Jung, P. McDaniel, and A. N. Sheth, "Taintdroid: an information-flow tracking system for realtime privacy monitoring on smartphones," *ACM Transactions on Computer Systems (TOCS)*, vol. 32, no. 2, pp. 1–29, 2014.

[9] M. Kan, "Android malware that can infiltrate corporate networks is spreading," 2016. [Online]. Available: https://www.computerworld.com/article/3126390/android-malware-that-can-infiltrate-corporate-networks-is-spreading.html

[10] D. Palmer, "Over 400 instances of dresscode malware found on google play store, say researchers," Oct. 2016. [Online]. Available: https://www.zdnet.com/article/over-400-instances-of-dresscode-malware-found-on-google-play-store-say-researchers/

[11] I. Golovin, "Unkillable xhelper and a trojan matryoshka," Apr. 2020. [Online]. Available: https://securelist.com/unkillable-xhelper-and-a-trojan-matryoshka/96487/

[12] W. Wong, "New malware-as-a-service threat targets Android phones," Sep. 2018. [Online]. Available: https://securityintelligence.com/news/new-malware-as-a-service-threat-targets-android-phones/

[13] A. Hazum and F. He, "Agent smith: A new species of mobile malware," 2019. [Online]. Available: https://research.checkpoint.com/2019/agent-smith-a-new-species-of-mobile-malware/

[14] M. Benson, "Bank 2fa codes and android malware breaches online banking security," 2020. [Online]. Available: https://www.cybernewsgroup.co.uk/bank-2fa-codes-android-malware-breaches-online-banking-security/

[15] Palo Alto Networks, "The world's first ml-powered ngfw," 2020. [Online]. Available: https://www.paloaltonetworks.com/network-security/next-generation-firewall

[16] ——, "Palo alto networks education services," 2020. [Online]. Available: https://www.paloaltonetworks.com/services/education

[17] I. Rosenberg, A. Shabtai, Y. Elovici, and L. Rokach, "Adversarial machine learning attacks and defense methods in the cyber security domain," *ACM Computing Surveys (CSUR)*, vol. 54, no. 5, pp. 1–36, 2021.

[18] W. U. Hassan, A. Bates, and D. Marino, "Tactical provenance analysis for endpoint detection and response systems," in *2020 IEEE Symposium on Security and Privacy (SP)*, 2020, pp. 1172–1189.

[19] Microsoft Intune, "Microsoft intune securely manages identities, manages apps, and manages devices," 2021. [Online]. Available: https://learn.microsoft.com/en-us/mem/intune/fundamentals/what-is-intune

[20] IBM, "Mobile device management (mdm) solutions," 2024. [Online]. Available: https://www.ibm.com/products/maas360/mobile-device-management

[21] Z. Chuluundorj, S. Liu, and C. A. Shue, "Generating stateful policies for iot device security with cross-device sensors," in *IEEE International Conference on Network of the Future (NoF)*, 2022.

[22] S. Burnett, L. Chen, D. A. Creager, M. Efimov, I. Grigorik, B. Jones, H. V. Madhyastha, P. Papageorge, B. Rogan, C. Stahl *et al.*, "Network error logging: Client-side measurement of end-to-end web service reliability," in *USENIX Symposium on Networked Systems Design and Implementation*, 2020, pp. 985–998.

[23] N. A. Handigol, *Using packet histories to troubleshoot networks*. Stanford University, 2013.

[24] X. Wei, L. Gomez, I. Neamtiu, and M. Faloutsos, "Profiledroid: Multi-layer profiling of android applications," in *International Conference on Mobile Computing and Networking*, 2012.

[25] H. Zhang, D. She, and Z. Qian, "Android root and its providers: A double-edged sword," in *ACM Conference on Computer and Communications Security*, 2015.

[26] M. Bokhorst, "Netguard - no-root firewall," 2021. [Online]. Available: https://play.google.com/store/apps/details?id=eu.faircode.netguard

[27] Grey Shirts, "Noroot firewall," 2020. [Online]. Available: https://play.google.com/store/apps/details?id=app.greyshirts.firewall&hl=en_US&gl=US

[28] Android Studio, "Vpnservice," 2019. [Online]. Available: https://developer.android.com/reference/android/net/VpnService

[29] A. Rao, J. Sherry, A. Legout, A. Krishnamurthy, W. Dabbous, and D. Choffnes, "Meddle: middleboxes for increased transparency and control of mobile traffic," in *Proceedings of the 2012 ACM conference on CoNEXT student workshop*, 2012, pp. 65–66.

[30] A. Le, J. Varmarken, S. Langhoff, A. Shuba, M. Gjoka, and A. Markopoulou, "Antmonitor: A system for monitoring from mobile devices," in *Proceedings of the 2015 ACM SIGCOMM Workshop on Crowdsourcing and Crowdsharing of Big (Internet) Data*, 2015, pp. 15–20.

[31] T. Sipola, A. Juvonen, and J. Lehtonen, "Anomaly detection from network logs using diffusion maps," in *Engineering Applications of Neural Networks*. Springer, 2011, pp. 172–181.

[32] A. Lanzi, D. Balzarotti, C. Kruegel, M. Christodorescu, and E. Kirda, "Accessminer: using system-centric models for malware protection," in *Proceedings of the 17th ACM conference on Computer and communications security*, 2010, pp. 399–412.

[33] W. Yang, X. Xiao, B. Andow, S. Li, T. Xie, and W. Enck, "Appcontext: Differentiating malicious and benign mobile app behaviors using context," in *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, vol. 1. IEEE, 2015, pp. 303–313.

[34] X. Wang, X. Qin, M. B. Hosseini, R. Slavin, T. D. Breaux, and J. Niu, "Guileak: Tracing privacy policy claims on user input data for android applications," in *Proceedings of the 40th International Conference on Software Engineering*, 2018, pp. 37–47.

[35] Z. Yang, M. Yang, Y. Zhang, G. Gu, P. Ning, and X. S. Wang, "Appintent: Analyzing sensitive data transmission in android for privacy leakage detection," in *ACM SIGSAC Conference on Computer & Communications Security*, 2013.

[36] C. Zheng, S. Zhu, S. Dai, G. Gu, X. Gong, X. Han, and W. Zou, "Smartdroid: an automatic system for revealing ui-based trigger conditions in android applications," in *ACM Workshop on Security and Privacy in Smartphones and Mobile Devices*, 2012.

[37] C. Hu and I. Neamtiu, "Automating gui testing for android applications," in *Proceedings of the 6th International Workshop on Automation of Software Test*, 2011, pp. 77–83.

[38] Y. Sui, Y. Zhang, W. Zheng, M. Zhang, and J. Xue, "Event trace reduction for effective bug replay of android apps via differential gui state analysis," in *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2019, pp. 1095–1099.

[39] W. Choi, G. Necula, and K. Sen, "Guided gui testing of android apps with minimal restart and approximate learning," *Acm Sigplan Notices*, vol. 48, no. 10, pp. 623–640, 2013.

[40] L. Gomez, I. Neamtiu, T. Azim, and T. Millstein, "Reran: Timing-and touch-sensitive record and replay for android," in *2013 35th International Conference on Software Engineering (ICSE)*. IEEE, 2013, pp. 72–81.

[41] OpenJS Foundation, "Appium: Automation for apps," 2021. [Online]. Available: https://appium.io/docs/en/2.0/

[42] Android Studio, "Ui/application exerciser monkey," 2020. [Online]. Available: https://developer.android.com/studio/test/monkey

[43] ——, "Create your own accessibility service," 2019. [Online]. Available: https://developer.android.com/guide/topics/ui/accessibility/service

[44] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "Openflow: enabling innovation in campus networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, pp. 69–74, 2008.

[45] D. Levin, M. Canini, S. Schmid, and A. Feldmann, "Incremental sdn deployment in enterprise networks," *ACM SIGCOMM Computer Communication Review*, vol. 43, no. 4, pp. 473–474, 2013.

[46] C. Lorenz, D. Hock, J. Scherer, R. Durner, W. Kellerer, S. Gebert, N. Gray, T. Zinner, and P. Tran-Gia, "An sdn/nfv-enabled enterprise network architecture offering fine-grained security policy enforcement," *IEEE communications magazine*, vol. 55, no. 3, pp. 217–223, 2017.

[47] J.-L. Chen, Y.-W. Ma, H.-Y. Kuo, C.-S. Yang, and W.-C. Hung, "Software-defined network virtualization platform for enterprise network resource management," *IEEE Transactions on Emerging Topics in Computing*, vol. 4, no. 2, pp. 179–186, 2015.

[48] J. Bailey and S. Stuart, "Faucet: Deploying sdn in the enterprise," *Communications of the ACM*, vol. 60, no. 1, pp. 45–49, 2016.

[49] N. Feamster, "Outsourcing home network security," in *ACM SIGCOMM Workshop on Home Networks*, 2010.

[50] C. R. Taylor, C. A. Shue, and M. E. Najd, "Whole home proxies: Bringing enterprise-grade security to residential networks," in *IEEE International Conference on Communications*, 2016.

[51] Y. Liu, C. R. Taylor, and C. A. Shue, "Authenticating endpoints and vetting connections in residential networks," in *2019 International Conference on Computing, Networking and Communications (ICNC)*. IEEE, 2019, pp. 136–140.

[52] C. R. Taylor, D. C. MacFarland, D. R. Smestad, and C. A. Shue, "Contextual, flow-based access control with scalable host-based sdn techniques," in *IEEE INFOCOM*, 2016.

[53] M. E. Najd and C. A. Shue, "Deepcontext: An openflow-compatible, host-based sdn for enterprise networks," in *2017 IEEE 42nd Conference on Local Computer Networks (LCN)*. IEEE, 2017, pp. 112–119.

[54] Y. Lei and C. A. Shue, "Detecting root-level endpoint sensor compromises with correlated activity," in *International Conference on Security and Privacy in Communication Systems*. Springer, 2019, pp. 273–286.

[55] S. Demetriou, N. Zhang, Y. Lee, X. Wang, C. A. Gunter, X. Zhou, and M. Grace, "Hanguard: Sdn-driven protection of smart home wifi devices from malicious mobile apps," in *ACM Conference on Security and Privacy in Wireless and Mobile Networks*, 2017.

[56] B. Yuan, D. Zou, S. Yu, H. Jin, W. Qiang, and J. Shen, "Defending against flow table overloading attack in software-defined networks," *IEEE Transactions on Services Computing*, vol. 12, no. 2, pp. 231–246, 2016.

[57] LabCIF, "How to intercept network trafic on android," 2024. [Online]. Available: https://github.com/LabCIF-Tutorials/Tutorial-AndroidNetworkInterception

[58] K. Vishnubhotla, "Every industry's battle: The threat of mobile malware on the enterprise," 2024. [Online]. Available: https://www.zimperium.com/blog/every-industrys-battle-the-threat-of-mobile-malware-on-the-enterprise/

[59] C. R. Taylor, T. Guo, C. A. Shue, and M. E. Najd, "On the feasibility of cloud-based sdn controllers for residential networks," in *2017 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*. IEEE, 2017, pp. 1–6.

[60] SimilarWeb, "Top websites ranking," 2023. [Online]. Available: https://www.similarweb.com/top-websites/

[61] E. Rejthar, "Searching for malicious code among add-ons," 2020. [Online]. Available: https://blog-nic-cz.translate.goog/2020/11/19/hledani-skodliveho-kodu-mezi-doplnky/?_x_tr_sl=cs&_x_tr_tl=en&_x_tr_hl=en-US

[62] Avast Researchers, "Third-party extensions for facebook, instagram, and others have infected millions," 2020. [Online]. Available: https://blog.avast.com/malicious-browser-extensions-avast

[63] Cloudflare Radar, "Domain rankings for united states," 2023. [Online]. Available: https://radar.cloudflare.com/domains/us

[64] Firefox Developers, "Firefox profiler," 2023. [Online]. Available: https://profiler.firefox.com/

[65] Android Stuido, "Measure app performance with Android Profiler," Oct. 2020. [Online]. Available: https://developer.android.com/studio/profile/android-profiler
[66] AskF5, "K50557518: Decrypt ssl traffic with the sslkeylogfile environment variable on firefox or google chrome using wireshark," 2021. [Online]. Available: https://support.f5.com/csp/article/K50557518
[67] NowSecure, "Frida, a world-class dynamic instrumentation framework," 2021. [Online]. Available: https://frida.re/docs/home/
[68] C. R. Taylor and C. A. Shue, "Validating security protocols with cloud-based middleboxes," in *2016 IEEE Conference on Communications and Network Security (CNS)*. IEEE, 2016, pp. 261–269.