By Your Command: Extracting the User Actions that Create Network Flows in Android

Shuwen Liu, Joseph P. Petitti, Yunsen Lei, Yu Liu, Craig A. Shue
Worcester Polytechnic Institute
{sliu9, jppetitti, ylei3, yliu25, cshue}@wpi.edu

***Abstract*—Given the complexity of modern systems, it can be difficult for device defenders to pinpoint the user action that precipitates a network connection. Mobile devices, such as smartphones, further complicate analysis since they may have diverse and ephemeral network connectivity and support users in both personal and professional capacities. There are multiple stakeholders associated with mobile devices, such as the end-user, device owner, and each organization whose assets are accessed via the device; however, none may be able to fully manage, troubleshoot, or defend the device on their own.**

In this work, we explore a set of techniques to determine the root cause of each new network flow, such the button press or gesture for user-initiated flows, associated with a mobile device. We fuse the User Interface (UI) context with network flow data to enhance network profiling on the Android operating system. In doing so, we find that we can improve network profiling by clearly linking user actions with network behavior. When exploring effectiveness, the system enables allow-lists to reach over 99% accuracy, even when user-specified destinations are used.

I. Introduction

Mobile devices have varied and opportunistic network usage patterns, they are often personally-owned and used for personal and professional purposes, and they are often vital in modern organizations. These intertwined patterns complicate efforts to manage the devices since no individual stakeholder may have all the necessary technical knowledge or visibility into the device to troubleshoot issues or enhance security.

IT system and network managers need visibility into endpoints to expedite problem diagnosis and reduce resource usage and infrastructure costs [1]. However, even with detailed logging infrastructure, the link between user actions and the resulting behavior of a device may be unclear. When errors occur, organizations need ways to localize the defect, which can be more difficult when they do not own or have direct access to the device. Further, proper security of mobile devices is a concern since organizations need ways to ensure mobile devices are not allowing adversaries a foothold into their systems. Device owners and end users need to ensure a device is protected without sacrificing their privacy. Further, solutions in this space must avoid impractical requirements (e.g., recompiling the Android kernel or gaining root access to their device, which may incur its own security risks [2]), which was a requirement in prior work [3], [4].

Since the network communication plays a key role in troubleshooting performance or security concerns, an accurate network sensor could help secure the devices in the network. However, since mobile devices can connect to a wide range of networks, it may be infeasible to ensure every network has in-network sensors. Accordingly, we explore options to improve network sensing on the mobile device end-point itself. Prior work [5] shows it is applicable to implement software-defined networking (SDN) tools as an effective network sensor on endpoints running the Windows operating system. However, while that prior work could leverage kernel device drivers to perform its work, 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 exploration leads to the research question: *Can UI interaction and network activity be used to successfully predict and associate network flows with user actions on Android devices?* Mobile devices operating systems have isolation techniques that may hinder such an association.

In the exploration, we make the following contributions:

- **Create a UI-aware Endpoint Network Sensing System:** We create a new Android application, called AppJudicator, which leverages UI interaction and SDN principles to determine whether network flows are legitimately user-initiated (Section III).
- **Characterize Network Profiling Potential for UI-aware Systems:** We explore AppJudicator's ability to perform real-time network profiling and management on Android devices. We enable dynamic allow-lists that can leverage user actions to permit user-driven events. We found that our system helps increase the accuracy of allow lists for user-supplied destinations from less than 7% to over 99% (Section IV).

II. Background and Related Work

Our approach is related to prior work in profiling Android system and network characteristics, graphic user interfaces and accessibility services, and end-point SDN techniques. We now provide context and background in each of these areas.

A. Profiling Systems and Networks on Android

Prior work has shown the value of profiling communication patterns on Android. ProfileDroid [6] characterized application traffic and privacy concerns. They linked coarse-grained user-level information with network flows to identify traffic associated with advertising from primary application traffic. Netsight [7] helped localize the root causes associated with

network failures. Other work demonstrated the effectiveness of endpoint network logging by using browser data to explore the end-to-end network path to distinguish on-path failures from attacks [8].

While powerful, some network instrumentation on Android devices may require root privileges to devices, raising security concerns [9] and possibly complicating deployment. To avoid such challenges, NoRoot [10] and NetGuard [11] configure the built-in Android VPN client to route traffic to their own local applications in order to enable profiling and firewalling services. Meddle [12] enables inspection of network traffic remotely using the VPN interface. Our work likewise uses the VPN interface to obtain network traffic without requiring root privileges; however, unlike prior work, we fuse this information with the user's activities in the UI to distinguish user-driven traffic from automated processes.

Prior work has explored network logging on mobile devices for network troubleshooting [13] and digital forensics [13]. Further, Sipola et al. [14] used deep learning methods on network logs to identify malicious network activities. Our approach augments traditional network logs with user activity to help provide context for operations.

The instrumentation of Android devices can look at internal interactions as well. Prior work has explored system calls and their ability to reconstruct users' intended actions [15], [16]. While Android maintains a system log, analysts typically need a debugger or root access to get the full logs. Using static analysis, AppContext [17] found patterns that distinguish malicious apps and benign apps. However, such approaches are not amenable for real-time network profiling.

B. User Interface (UI) and Accessibility Sensors

End users control programs on mobile devices via the user interface. That interface can provide context for different operations. AppIntent [18] uses static analysis to trace UI activities to data transmission events; however, this technique cannot be used in real-time traffic classification. The GUILeak [19] tool traces UI actions and records user input, which is manually compared with contractual and privacy policies described by the vendor to detect privacy violations. Given the manual steps involved, these processes are not amenable to real-time traffic evaluation. SmartDroid [20] proposes to enforce a sequence of system calls associated with UI elements. However, the method is difficult to implement on average users' mobile phones since it usually requires rooting a device.

Tools to automate UI interactions have also been developed to help with software testing. The Android `Monkey` [21] tool is an official mechanism that allows developers to send sequential events to the device via its `adb` debugger interface. `Appium` [22] is a tool commonly used for software test automation. We use `Appium` in our experiments.

Android provides an accessibility API [23] that allows accessibility services developers to cross the traditional boundary of sandboxed applications. The services can send commands and receive UI events invoked by a human user to other applications. Chuluundorj et al. [24] apply this service to leverage UI event information in securing IoT devices in home networks, which shows UI events can be effectively used in network profiling.

C. Software-Defined Networking (SDN) on Endpoints

SDN separates the data plane that forwards traffic from the control plane that determines the appropriate data plane table entries. SDN has been well studied in enterprise networks [25]–[28] and residential networks [29]–[31]. However, these SDN agents are typically implemented in switches, either in hardware or in virtual hypervisors. SDN can also use agents on endpoints. Taylor et al. [27], [32] provide a prototype to shift an SDN agent from a switch to an end-host. Chuluundorj et al. [5] develop and implement a practical SDN agent in the Windows operating system.

SDN has also been implemented in Android: meSDN [3] and PBS-Droid [4] propose to deploy SDN agents in the Linux kernel on Android. However, this method requires recompiling the OS, which is an obstacle for many users. HanGuard [33] proposes to install SDN technology on both Android endpoints and home routers to enforce fine-grained access control. Unlike these works, we focus on deployability and enabling greater context from endpoints to enhance access control decisions.

III. Approach: UI and Network Sensor Fusion

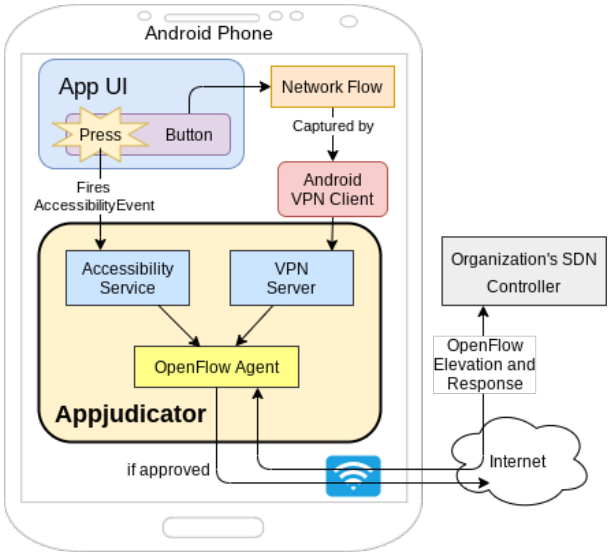

Fig. 1. AppJudicator sends UI and network data to an OpenFlow agent.

A. Instrumenting the Network via the Android VPN API

As described in Section II-A, prior work has established that capturing network traffic through Android's built-in VPN client allows an application to intercept traffic device-wide without requiring root access to the device. Such functionality has also been used in Apple's iOS for similar purposes. We use this established approach to enable our novel data fusion efforts.

In our configuration, the built-in VPN API provides an interface that can tunnel all traffic to APPJUDICATOR's local VPN server component. We further use the interface to build two streams: 1) a VPNInputStream that intercepts packets from applications and 2) a VPNOutputStream that allows the module to transmit reply packets to the application. We depict the sequence of the module's actions in Figure 2. The VPN service allows APPJUDICATOR to control the intercepted packets and implement an SDN agent. If the SDN agent has a table entry for the flow, it will deliver the packet as directed by that table entry (step m in Figure 2). Otherwise, the agent will initiate the OpenFlow elevation process (steps d through i in Figure 2).

Since TCP and UDP packets need different processing, APPJUDICATOR divides packets and routes them to either the `TCPVPNService` or the `UDPVPNService`. When a connection is received in the `TCPVPNService`, it initiates a connection to the remote system through a *protected socket*. The protected socket designation ensures that the communication is not itself intercepted by our VPN service (and thus creating a loop). Upon establishing the protected channel, the `TCPVPNService` works as a translation device, stitching together the connections between the local application and the VPN service and the VPN service to the remote server (shown as step n and o in Figure 2).

Finally, we need a mechanism to link packets back to their associated application. We use the Android `ConnectivityManager` API to identify the user ID (UID) associated with the flow's fields. Since each Android application has a different UID, we can use the Android `PackageManager` and the `QUERY_ALL_PACKAGES` permissions to determine the package name associated with the UID. Accordingly, we can aggregate packets into flows using the standard flow tuple (IP_{src}, IP_{dest}, protocol, $Port_{src}$, $Port_{dest}$). To link the flow with a package, we first use `ConnectivityManager` Android API to look up the UID of the flow owner by (IP_{src}, protocol, $Port_{dest}$) information. The `PackageManager` Android API further allows us to obtain the package name and context for each UID. To support SDN functionality, we use the flow tuple as the key in a hash table combined with a queue data structure for packets awaiting a verdict from the SDN controller.

B. Instrumenting the UI with Accessibility Services

The accessibility services and APIs on the Android platform provide a way to gather data about UI events. These accessibility services are designed to support alternative user interactions, such as screen readers for individuals with impaired vision. We leverage these services to correlate UI events with networks flows to establish the origin for each network action.

We enable this service in our application by declaring it in our application's `Manifest` [23] file and specifying the `BIND_ACCESSIBILITY_SERVICE` and the `canRetrieveWindowContent` permissions. Using the `intent-filter`, we specify the scope of applications that we monitor. In our prototype, we include all applications in our scope but note that production deployment may tailor the scope to achieve privacy goals. Given the potential privacy and security implications of these permissions, the Android OS itself provides a clear statement about the service's capabilities and the associated risks.

The accessibility services represent UI events as a UI state change. Our application listens for all types of UI events, including button presses, swipes, long presses, and focus changes. For each event, we can obtain a context description from the event object. With the `AccessibilityEvent.getSource()` function, we can extract more information about the layout hierarchy, including information associated with parent and child widgets. This function allows our system to identify text and labels most closely associated with an action, allowing us to create detailed context for an action. For example, for the `TYPE_VIEW_TEXT_SELECTION_CHANGED` UI event type, we can obtain the contents of an `EditText` UI object, which often contains a user's input. When these inputs contain an IP address, DNS host name, or URL, it can aid correlation of network flows; otherwise, the data can be ignored to preserve privacy. We further use the `FLAG_INCLUDE_NOT_IMPORTANT_VIEWS` flag with accessibility services to obtain full visibility into the application's UI [23].

APPJUDICATOR obtains and stores all UI events in the scope of monitoring on a per-application basis. When queried by the SDN agent, the service provides the most recent events.

C. Fusing UI and Network Sensor Data via SDN

APPJUDICATOR implements a subset of the OpenFlow v1.0 specification. Since our software only operates on a single device, it does not implement the full set of OpenFlow options. We thus omit data about the data link layer headers, the physical ports, or VLANs. As with traditional OpenFlow agents, we allow wildcards in the rule match fields. Our SDN agent finds the most specific matching rule and returns the associated action to the VPN service to apply to any queued or subsequent packets associated with the flow.

If the OpenFlow agent does not find a table match on the packet received from `VPNService`, it elevates the packet to the SDN controller, allowing the controller to profile the traffic and provide direction on how to handle the packet. In our prototype, we only implement the *drop* and *forward* actions, which will discard or transmit packets, respectively. Future work may support additional OpenFlow actions.

Unlike the traditional OpenFlow `PACKET_IN` message sent to an SDN controller, our agent provides additional UI context. Our SDN agent queries the accessibility module for UI information associated with the flow (Section III-B). The UI events are attached in reverse-chronological order. However, the agent filters some UI events, like `TYPE_VIEW_CONTENT_CHANGED`, that are very frequent and have limited utility to the controller. Accordingly, APPJUDICATOR elevates the types of UI events that are most likely to reveal user intentions to controller. As reported in Section IV,

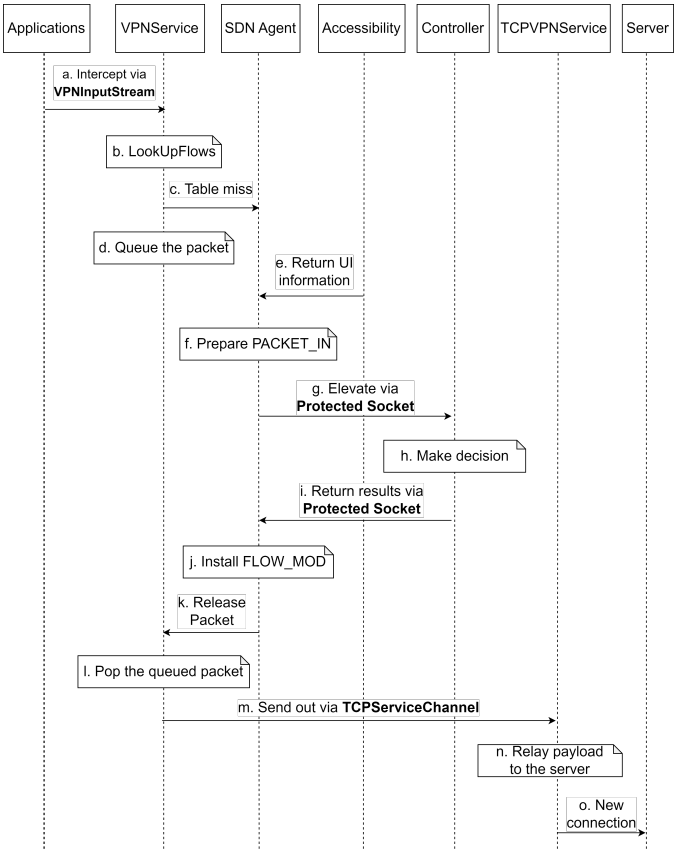

Fig. 2. Depiction of APPJUDICATOR outbound packet processing as a sequence diagram

we empirically found that limiting reports to UI events from the last four seconds is most effective.

IV. EFFECTIVENESS EVALUATION: NETWORK PROFILING

In this Section, we follow a similar evaluation methodology as Chuluundorj et al. [5]. Specifically, we compare ourselves against traditional network-based sensors used in enterprise networks, which only permit matched network flows. By testing the sensors with legitimate and malicious data, we generate the accuracy rates of different sensors. These sensors can be categorized into three broad classes:

- **IP Header Sensor**: This sensor only considers a network flow as a match if it has the same remote system's IP addresses and port numbers as in the idle or training data.
- **DNS-aware Sensor**: This sensor incorporates recent DNS queries with the IP Header Sensor data. In addition to matched network flows reported by the IP sensor, it considers a network flow as a match if it has the same host name as in the idle or training data.
- **UI-aware Sensor**: The UI sensor includes the matches from the DNS sensor as a baseline. The UI sensor would further consider a network flow as a match if the remote server appears in the UI context data.

We start our exploration by analyzing the background activity associated with an application without any end-user activity. To do so, we record all the network activities for three

Application	Data Samples Count			Allow-List Match Rate		
	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%

TABLE I
ALLOW-LIST MATCH RATE WITH IP, DNS, AND UI SENSORS FOR LEGITIMATE, APPIUM-SUPPLIED DESTINATIONS. THE DNS ALLOW-LISTS CAN SUCCEED WITH LIMITED-DESTINATION APPLICATIONS (E.G., AS WITH GMAIL AND YOUTUBE). WHEN DESTINATIONS ARE DYNAMICALLY SPECIFIED (E.G., TERMIUS), ONLY THE UI SENSOR IS EFFECTIVE AT ALLOWING TRAFFIC.

different sensor types while each tested application is idle. We create rules permitting all such network traffic to construct an `Idle Period` allow-list that approves background activities that do not need user intervention.

We next train our research set-up to construct an association between end-user actions and network activity. We use `Appium` [22] to automatically invoke a scripted sequence of UI elements. Instead of traversing all possible UI elements, we focus on the events that only occur when a user interacts with the device. When `Appium` creates these inputs by simulating users' actions, every action is presented on the phone's screen. In this training phase, we record the UI events and network behaviors observed at our sensors. We label this data as the `Training Data`, which records what UI information and network activity appears together in a non-compromised application. During our later testing phase, each sensor can construct allow-list rules using the training data to determine if it can correctly classify traffic.

For our evaluation, we selected popular applications from the Google Play Store that are widely used in people's daily lives, including Gmail and YouTube. We add Termius, an SSH client, to cover a broad range of usage scenarios. We note that Gmail and YouTube interact with a limited set of remote systems such as the servers associated with the application developer or advertisers. However, Termius allow users to specify arbitrary destinations, as URLs or destination IP addresses or host names, providing significant flexibility but making profiling efforts more complex. Nonetheless, the proposed UI sensor can provide useful information containing URLs, IP addresses, and DNS host names, which can help improve profiling efforts. In this experiment, we focused our analysis on user inputs in the web browsers and SSH client.

During the `Idle Period` training phase, we activate each application and leave the program running without user input for two hours. For the `Training Data` stage, we develop thirty work flow scripts for Gmail and YouTube. For Termius, we create a specific work flow that randomly visits different destination servers. We use the websites in the top 500 websites list from `SimilarWeb` [34] and divide them equally into separate, non-overlapping groups for training and testing. We collect the data set of idle period and training phase at the SDN controller. During the testing phase, we execute the same work flow scripts, while the controller employs three rule matching sensors with rules constructed from the previous training data.

In Table I, we show the allow-list match rates for Termius, Gmail, and YouTube. The accuracy rate stands for how much legitimate traffic has been matched in the sensors. In these scenarios, all traffic is legitimate: it is initiated by `Appium` to replicate end-user behavior. Focusing on Gmail and YouTube, we see that the IP Sensor has the lowest accuracy for these applications. The result appears to be influenced by DNS load balancing or the effects of content distribution network (CDN) deployments. The DNS sensor's accuracy is higher than 98.8% for Gmail and YouTube, demonstrating the value of host name matching rules. When examining the SSH client, we see that the IP Sensor and the DNS sensor have lower accuracy, matching around 6.6% to 6.9% of traffic. In contrast, our UI sensor enables a match rate that is 99.1% at the controller. This highlights the importance of understanding the user-specified destinations when profiling traffic.

V. Conclusion

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 allow lists for user-supplied destinations from less than 7% to over 99%. We believe the work holds promise for further exploration.

Acknowledgements

This material is based upon work supported by the National Science Foundation under Grant No. 1651540.

References

[1] Sematext, "Log analysis tutorial: What it is, why, and when devops use it," https://sematext.com/blog/log-analysis/, Sep 2020.

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

[3] 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.

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

[5] 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.

[6] 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.

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

[8] 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.

[9] 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, pp. 1093–1104.

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

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

[12] 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.

[13] A. J. Slagell, Y. Li, and K. Luo, "Sharing network logs for computer forensics: A new tool for the anonymization of netflow records," in *Workshop of the IEEE International Conference on Security and Privacy for Emerging Areas in Communication Networks*, 2005.

[14] 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.

[15] S. A. Hofmeyr, S. Forrest, and A. Somayaji, "Intrusion detection using sequences of system calls," *Journal of computer security*, vol. 6, no. 3, pp. 151–180, 1998.

[16] A. Lanzi, D. Balzarotti, C. Kruegel, M. Christodorescu, and E. Kirda, "Accessminer: using system-centric models for malware protection," in *ACM Conference on Computer and Communications Security*, 2010.

[17] 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.

[18] 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 *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, 2013, pp. 1043–1054.

[19] 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.

[20] 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, pp. 93–104.

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

[22] J. Foundation, "Appium: Automation for apps," https://appium.io/docs/en/2.0/, 2021.

[23] Android Studio Devs., "Create your own accessibility service," https://developer.android.com/guide/topics/ui/accessibility/service, 2019.

[24] 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.

[25] 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.

[26] 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.

[27] 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.

[28] 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.

[29] N. Feamster, "Outsourcing home network security," in *ACM SIGCOMM Workshop on Home Networks*, 2010, pp. 37–42.

[30] 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.

[31] 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.

[32] 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.

[33] 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.

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