Towards Automated Policy Training in Android Software-Defined Networking

Shuwen Liu
Worcester Polytechnic Institute
Worcester, MA, USA
sliu9@wpi.edu

Craig A. Shue
Worcester Polytechnic Institute
Worcester, MA, USA
cshue@wpi.edu

***Abstract*—Bring-your-own-device (BYOD) deployments often rely on mobile endpoint controls to protect enterprise resources. However, policy construction largely remains manual and difficult to maintain as applications evolve. This paper presents an automated policy training framework for Android software-defined networking (SDN) enforcement that derives filtering policies from observed application behavior. Our approach correlates user interface (UI) context with network activity to automatically generate enforcement rules from task-driven application workflows. The framework supports multiple policy abstractions, including allow-list policies based on observable metadata, attribute-based access control (ABAC) rules derived from behavioral patterns, and transparency-oriented rules that approximate whether network activity is attributable to user-visible actions. By leveraging automated workflow replay and behavioral trace collection, the system enables policy generation without requiring manual rule authoring or prior policy corpora. Experimental evaluation demonstrates that the resulting policies effectively distinguish legitimate application traffic from covert background activity, denying all malicious traffic in covert traffic tests while maintaining up to 99.8% acceptance of legitimate traffic when incorporating decrypted HTTPS information. Additional experiments with a trojan horse spyware show that even when the malware introduces limited user interaction, the vast majority of its traffic remains non-transparent and detectable. These results suggest that automated behavioral policy training can substantially reduce deployment complexity while improving the accuracy and maintainability of security policies in BYOD environments, while also raising the bar for adversaries by constraining stealthy attack strategies.**

I. Introduction

The bring your own device (BYOD) scenario introduces potential security risks to enterprise networks. Security tools in mobile devices, such as mobile device management (MDM) solutions [1], have been proposed for companies to ensure data protection compliance. Such tools provide centralized management of a company's policy for mobile phones. Ineffective policy can risk the loss of company data and the harmful installations by users [2], such as viruses or trojans on a company's mobile device. Liu et al. [3] introduces AppJudicator for securing mobile devices in BYOD systems via software-defined networking (SDN) enforcement and use allow-list policies as a proof-of-concept to validate its effectiveness. Such allow-list policies are critical to the access control system. However, generating and maintaining fine-grained policies is challenging for deployers due to the dynamic, context-dependent nature of mobile application behaviors and the lack of transparent, structured mappings between user actions and network activities. In this work, we explore tools and strategies to simplify the creation and maintenance of filtering policies.

AppJudicator's integration of user interface (UI) context and network flows introduces new research possibilities in developing efficient, UI-driven security policies. We explore the inclusion of various types of information, such as plaintext network traffic and UI data, to construct security policies. Moreover, we explore the practical aspects of policy creation and validation. We propose tools and mechanisms to analyze individual applications by replaying representative workflows and automatically deriving multiple policy abstractions (e.g., allow-list, ABAC, and transparency-based rules) from the resulting behavioral traces. We further evaluate the proposed mechanism in terms of the efficiency of policy generating and the effectiveness of policy enforcement.

A prior study [4] used the Sikuli automation framework to generate application traces in popular Windows programs, such as Microsoft Word, PowerPoint, and Excel. These traces identify typical application workflows by capturing sequences of mouse clicks and keystrokes, some of which triggered network activity. Since mobile application developers rely on specific APIs to operate UI actions, we can use automation test tools, such as `Appium` [5], to trigger UI events across various applications. However, existing tools fall short in generating complete application workflows because they cannot automatically enumerate all application UI elements comprehensively [6]. To address this gap, we enhance these tools with capabilities of automation by proposing an automated framework for data collection and policy construction.

This work focuses on supporting security policy deployment within organizational environments. We propose an automated policy training framework for mobile devices that includes several automated tools and strategies to generate and validate security policies. We leverage the Android-in-the-Wild (AITW) data set [7], which is a large-scale dataset for mobile device control that contains natural language instructions, UI screens, and actions for a variety of human tasks. We create a UI control tool that replays the AITW data set to generate workflows for mobile applications. While executing these workflows, we use an automated data capture tool to

collect various types of information related to the workflows, including network traffic, UI data, and application data. We leverage this data to construct various types of policies, including allow-list policies and attribute-based access control (ABAC) policies.

Beyond simple allow-listing, we investigate how different categories of observable data can serve as features to improve policy accuracy. In particular, we compare policies trained using only plaintext metadata, such as domains and UI context, against policies that additionally incorporate decrypted HTTP header features obtained through trusted interception. We further explore ABAC policy extraction and user transparency rules derived from UI and network correlations. These policy mechanisms are easily generated and enable automated policy construction with minimal manual configuration while preserving high enforcement accuracy.

In exploring these directions, we ask the following research questions (RQs):

RQ1: Role of decrypted vs. plaintext network data in training: When policies are trained from application-specific traces, does incorporating both plaintext-observable metadata and decrypted HTTP data improve deployable enforcement accuracy compared to training on plaintext-only features?

RQ2: Role of transparency in security goals: Can we apply "disclosure to the end user" as a measurable attribute—derived from UI events and network evidence—to enable policy decisions to block covert traffic?

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

- We design an automated policy training pipeline that replays natural, task-driven mobile workflows and produces synchronized UI–network traces on physical devices.
- We show that incorporating a restricted set of stable decrypted HTTP header features (via trusted interception) yields a large reduction in collateral blocking, improving benign acceptance from 95.1% to 99.8% while preventing all malicious traffic in our covert traffic tests.
- We demonstrate two complementary policy abstractions beyond allow-lists: automatically extracted ABAC rules and transparency rules that approximate whether a connection is disclosed to the user, achieving strong separation between benign and covert background traffic, with 97.2% of legitimate traffic satisfying transparency rules compared to only 1.9% of MacroDroid traffic and 2.7% of Trojan-generated traffic.
- We provide practical deployment guidance on which features are stable enough for enforcement and how to validate generalization under workflow variation.

II. Related Work

This section reviews prior research in mobile device management, UI trace datasets, replay-based automation, LLM-driven mobile agents, and automated access-control policy construction. Although substantial progress has been made in each domain independently, their integration for policy construction in realistic BYOD settings remains underexplored and that fusion is instrumental to our work.

A. BYOD Policies

Enterprise BYOD enforcement is commonly implemented through MDM and conditional access systems. Major vendors such as Apple [8] and Microsoft [1] publish detailed management schemes and attribute definitions that describe device posture requirements, application identifiers, and contextual enforcement primitives. Microsoft's conditional access framework further exposes identity-driven policy attributes and environmental conditions [9]. These specifications clarify what can be enforced, but they do not provide systematic guidance on how deployers should derive application-specific rules from empirical behavioral traces.

In practice, administrators often manually construct allow-lists or conditional rules based on coarse application identifiers or destination domains. Such policies may drift from real application behavior as applications evolve. Prior mobile security research has incorporated contextual signals, including application identity and UI state, to improve enforcement precision [3]. However, the data collection process in these systems remains largely manual, and we found no publicly available dataset that pairs real-world BYOD policies with synchronized UI–network traces. This absence motivates automated policy training: rather than assuming a pre-existing policy corpus, we generate task-driven traces and infer enforceable rules directly from observed UI and network behavior.

B. UI Datasets and Replay

Large-scale mobile UI datasets provide structured interaction traces, but most omit network information. Rico [10] captures UI hierarchies and gesture traces across thousands of Android applications, enabling large-scale UI modeling. Android-in-the-Wild (AITW) [7] extends this direction by collecting task-driven demonstrations with natural-language instructions and detailed action sequences. Other datasets focus on specific aspects of UI representation or task execution. For example, MoTIF [11] emphasizes mobile task workflows, while UIBert [12] provides UI element grounding corpora. UGIF [13] concentrates on instructional demonstrations within mobile environments, and AITZ [14] augments traces with structured reasoning chains to support UI automation research. AMEX [15] further enriches interaction traces with multi-level annotations and executable sequences.

Despite these advances, none of the above datasets record synchronized network packets alongside UI traces. This limits their direct applicability to network-aware policy extraction. A notable exception is MIRAGE-AppAct-2024 [16], which reports a dataset that correlates UI actions with network flows labeled by package name and activity type. While this dataset demonstrates that joint UI–network instrumentation is feasible, it has not collected the detailed information in UI actions, such as UI input text. Consequently, research relying on open datasets must augment UI traces by replaying them and collecting network evidence independently.

Replay-based approaches attempt to bridge this gap by converting recorded traces into executable workflows. One line of work translates Rico videos into action sequences and reports partial replay success on physical devices [17]. Another extracts replayable macros from Rico traces to support automation pipelines [18]. However, reproducibility remains challenging. Studies reviewing Android record-and-replay pipelines report substantial replay instability across user scenarios and failure types [19]. Flaky UI tests caused by asynchronous timing, environmental drift, and evolving layouts further illustrate the fragility of deterministic replay [20]. More broadly, the "replication crisis" has been characterized as a setting where experimental results cannot be reproduced and published findings become difficult to trust [21]. These observations align with prior discussions of replication challenges in HCI and empirical systems research [22], [23]. Rather than requiring perfect fidelity reproduction, our approach targets moderate fidelity that is sufficient to generate realistic traffic diversity for policy inference.

C. Mobile Agents and Reproducibility

Recent LLM-based mobile agents aim to execute complex tasks through iterative reasoning over screenshots and GUI trees. AutoDroid [24] integrates domain-specific application knowledge with language model reasoning to guide task execution. AppAgent [25] demonstrates that multimodal LLMs can operate smartphones through a simple tap-and-scroll action space without backend access. ClickAgent [26] separates reasoning from UI element localization and reports competitive replay success on AITW benchmarks. More recent systems such as MobileUse [27] introduce hierarchical reflection mechanisms to improve robustness, while GUI-explorer [28] performs pre-execution exploration to map screen transitions and mitigate knowledge staleness.

These agents share a common architecture: they extract GUI structure and screenshots, query an LLM for the next action, locate the corresponding UI element, execute the action, and iterate. While this paradigm can repair broken workflows and adapt to UI drift, it introduces practical challenges. Many reported results rely on large vision-language models or multi-GPU hosting environments. Latency, infrastructure dependencies, and quota constraints may limit their feasibility in real BYOD deployments where lightweight automation is preferred.

D. Automated Policy Construction

Automated policy extraction has been studied extensively in distributed and server-side systems. Early work explored deriving security policies directly from program behavior and data flows. For example, data-flow assertions have been proposed to enforce application security by specifying constraints on how information propagates through program execution [29]. In microservice environments, static analysis has been used to derive inter-service access rules by extracting invocation logic [30]. Log-based approaches such as Log2Policy [31] transform HTTP request logs into ABAC rules through clustering and attribute mining. These systems demonstrate that access-control policies can be inferred from observed interactions rather than manually authored.

Beyond service-level policies, recent work extracts database access-control constraints by analyzing SQL queries under varying HTTP request conditions [32]. By correlating request attributes with query predicates, such systems synthesize concise, human-readable rules describing permissible access patterns. Other systems research has explored mechanisms for enforcing high-level policies across complex distributed infrastructures. For example, Sesame [33] introduces policy containers and privacy regions to support end-to-end privacy compliance in distributed services. These approaches share a common objective with our work: deriving or enforcing security policies based on observable system behavior.

Our setting differs in two important dimensions. First, we observe both network flows and UI context, enabling policy features that connect user actions to network behavior. Second, BYOD enforcement may operate under partial visibility, requiring careful selection of stable and deployable attributes from plaintext metadata and selected HTTP data. We adopt an ABAC-style representation aligned with NIST guidance [34]. We extend policy construction with transparency-oriented predicates that approximate whether a connection is attributable to user-visible actions. This moves policy derivation beyond coarse destination filtering toward disclosure-aware enforcement tailored to mobile environments.

III. Threat Model and Trusted Computing Base

We consider an organizational BYOD deployment in which an Android device enforces network filtering policies through APPJUDICATOR. As in other endpoint security systems (e.g., MDM clients), defenders have limited ability to reduce the trusted computing base (TCB): the Android OS, the APPJUDICATOR application, and the enforcement path that observes flows and applies policy decisions must be trusted to execute correctly. If the OS or the security application is compromised, an adversary could bypass filtering, suppress or forge UI/network logs used for training, or misreport enforcement outcomes, undermining the guarantees of any data-driven policy construction.

Our automated policy training pipeline expands the TCB in two ways. First, we add components for on-device trace collection (e.g., UI instrumentation via accessibility APIs and network observation via the SDN enforcement path). Second, for experiments that incorporate decrypted HTTP information, we assume a trusted interception capability (e.g., a locally trusted certificate and interception service) that can expose selected HTTP header features during training and, if enabled, during enforcement. This assumption is common in enterprise environments where devices are provisioned under centralized administrative control, but it is optional in our design: when interception is unavailable or undesirable, APPJUDICATOR can still train and enforce policies using plaintext metadata (e.g., DNS domains, flow endpoints) and UI context.

We model the primary attacker as an installed malicious application that attempts to exfiltrate data or generate covert background traffic while remaining within the device's normal permission model. Concretely, the adversary may issue network requests without corresponding user-visible workflows (e.g., MacroDroid-style [35] automation or background services), attempt to blend into benign destinations, and exploit workflow variation and UI drift to evade rules derived from limited traces. We further consider more capable adversaries, such as the Android Trojan Horse Spyware [36], that generate network traffic concurrently with user interaction and may leverage social engineering to obtain permissions.

We assume that the adversary does not fully compromise the OS or the trusted enforcement components, but may attempt to evade detection by increasing apparent transparency (e.g., triggering UI events). However, such strategies require maintaining consistent and meaningful correlations between UI activity and network behavior over time. This introduces a practical constraint: to remain effective, the malware must either operate covertly with minimal user interaction, or actively engage the user in a manner that risks raising suspicion. Our design targets this tension, aiming to detect covert activity and increase the difficulty of constructing attacks that are both stealthy and behaviorally consistent.

IV. Approach

In this section, we present an automated framework for training Android SDN filtering policies from synchronized UI–network traces. We describe (i) automated data collection via workflow replay, and (ii) automated construction of allow-list, ABAC, and transparency-driven policies.

A. Automated Data Collection

Liu et al. [37] used `Appium` [5] to execute representative workflows of popular Android applications (e.g., Chrome, Gmail, YouTube) and leveraged AppJudicator to collect correlated network traffic and UI events as training data. They validated policies in a separate testing phase with injected malicious activity to measure accuracy in distinguishing legitimate vs. malicious behavior. This process required significant manual workflow authoring and repeated execution, limiting deployability in organizational BYOD settings.

To scale and diversify training traces, we develop an automated replay tool based on Appium [5] to reproduce web-shopping interaction traces from the AITW dataset [7] on a physical Android device (Moto G Power 2022). Each AITW episode specifies a natural-language goal and a sequence of UI actions with screenshots. To ensure deterministic browser initialization, we mitigate Chrome's First Run Experience (FRE) by clearing open tabs and enforcing a fresh start state at each replay. During execution, Appium may intermittently fail to identify UI elements due to dynamic content changes, focus loss, or application crashes. We incorporate error-handling mechanisms that detect failures and automatically restart the application so that replay can proceed without manual intervention. In practice, replay achieves moderate fidelity: it reproduces textual inputs (e.g., search queries) reliably; however, occasional deviations occur in selecting the exact websites specified by the original trace due to dynamic page layout and content drift. We download 1,122 web-shopping episodes from AITW and parse them into structured JSON files with corresponding screenshots, demonstrating scalability of the replay pipeline.

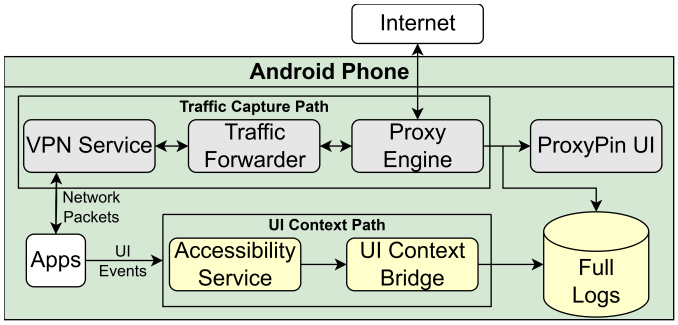

Fig. 1. The green shaded area shows an Android phone. The original Proxypin [38] functionality is shown in gray shapes. The yellow shapes depict the user-transparent components we add in this work, including an Accessibility service model, a UI context bridge, and a logging system. Our TCB includes all components in the shaded area, while the unshaded shapes show the applications and remote servers that could be malicious.

As shown in Figure 1, we develop a mobile traffic and UI capture tool that integrates a machine-in-the-middle (MITM) proxy with the Android VPN interface to observe network activity during replay. The system is deployed directly on a physical Android device and builds on key components from the open-source Proxypin project [38]. Application traffic generated by mobile apps is first intercepted by a local VPN Service, which creates a virtual network interface using the Android `VpnService` API. Captured packets are then passed to a Traffic Forwarder, which reconstructs TCP streams and routes them to a local Proxy Engine. The proxy engine performs HTTPS interception using a user-trusted certificate and extracts plaintext HTTP data from decrypted connections. During this process, we instrument the decrypted traffic pipeline to record structured request and response data, including IP/TCP metadata, selected HTTP headers, and bounded message bodies. The ProxyPin UI displays the processed traffic and exports it to a persistent *Full Logs* store for later analysis.

To correlate network activity with user interactions, we additionally introduce a parallel UI Context Path. This path leverages the Android `AccessibilityService` API to capture UI events generated by applications, including the foreground package name, UI class, event type, window title, visible text elements, and timestamps. These events are processed through a UI Context Bridge, which aggregates and exposes the captured UI context to the logging system. The resulting UI evidence is recorded alongside the network logs in the *Full Logs* repository, allowing network packets and UI events to be aligned. This combined logging pipeline enables joint analysis of application behavior at both the network and UI levels.

TABLE I
FIELDS CAPTURED FOR EACH NETWORK TRANSACTION RECORD.

Field Type	Visibility	Fields
Network data	Not encrypted	AppName
	Not encrypted	Remote_IP
	Not encrypted	Remote_Port
UI information	Not encrypted	PackageName
	Not encrypted	ClassName
	Not encrypted	EventType
	Not encrypted	WindowTitle
	Not encrypted	NodeText
	Not encrypted	Timestamp
HTTP data	Not encrypted	Domain
	Decrypted	URI
	Decrypted	HTTP_Method
	Decrypted	HTTP_StatusCode
	Decrypted	HTTP_Request_Header
	Decrypted	HTTP_Request_Body
	Decrypted	HTTP_Response_Header
	Decrypted	HTTP_Response_Body

Each captured record corresponds to a network transaction annotated with nearby UI events. The collected data includes network metadata, UI context information, and HTTP-level information. Table I summarizes the fields collected by our instrumentation and categorizes them based on whether they remain observable without HTTPS interception. In particular, we distinguish plaintext features that can be obtained directly from network flows or UI instrumentation (e.g., destination domains and UI context) from decrypted HTTP features that require trusted interception to access HTTP headers or message bodies. Notably, the HTTP domain name (hostname) is typically visible in plaintext during the initial DNS lookup and the Server Name Indication (SNI) phase of the HTTPS handshake, allowing the destination domain to be observed even when payload contents remain encrypted. This separation reflects realistic BYOD deployments in which HTTPS interception may be unavailable, restricted by policy, or selectively enabled only for specific applications.

B. Automated Policy Construction

Using the synchronized traces, we automatically construct several types of security policies. Each policy maps an observed record to an allow/deny decision at enforcement time.

We first construct allow-list policies using only plaintext fields (i.e. destination domain and UI text context). These policies capture stable communication patterns while remaining deployable in environments that cannot perform HTTPS interception. Concretely, the plaintext allow-list allows a record if its destination domain is observed during training and it is accompanied by admissible UI evidence (e.g., window title and/or on-screen text consistent with browsing activity). For example, when a user interacts with an e-commerce website `example.com` and the UI contains text such as "Example", network requests to `https://www.example.com` are consistently observed alongside events such as `TYPE_VIEW_CLICKED` and `TYPE_VIEW_FOCUSED`. Such recurring domain–UI relationships allow the policy to permit requests to `example.com` when similar UI context is present.

We then extend allow-lists by incorporating a restricted set of decrypted HTTP header attributes obtained through trusted interception. Each HTTP transaction is mapped to a compact fingerprint consisting of stable fields: `Host`, `Origin`, `Accept-Encoding`, `Accept-Language`, `Content-Type`, `User-Agent`, `Sec-Fetch-*`, and `sec-ch-ua-*`. We explicitly exclude volatile identifiers such as `Cookie`, `Authorization`, and other per-session validation headers, as well as size-dependent fields (e.g., `Content-Length`), to reduce brittleness across sessions. The resulting hybrid policy allows a record if and only if its fingerprint (optionally scoped by application identity and UI context) appears in the training-derived allow-set. For example, a typical request generated by Chrome when loading a web page may contain a distinctive header pattern such as `sec-ch-ua: "Chromium";v="120"`, `sec-ch-ua-platform: "Android"`, `Sec-Fetch-Site: same-origin`, `Sec-Fetch-Mode: navigate`, and `Sec-Fetch-Dest: document`. These headers describe the browser environment and request context and remain stable across browsing sessions, unlike session-specific identifiers such as cookies which vary. By constructing fingerprints from these contextual headers, the system can reliably recognize legitimate browser-originated traffic without relying on volatile identifiers.

Beyond allow-lists, we automatically extract ABAC policies from training traces. Each record is transformed into a tuple of subject attributes (application identity and package name), action attributes (UI event type and/or HTTP method), and resource attributes (destination domain and URI). These tuples yield structured rules that generalize beyond exact fingerprints while remaining interpretable and app-scoped. For example, a rule such as `(App = Chrome, EventType = TYPE_VIEW_CLICKED, Domain = example.com)` captures the common pattern where a user clicks on a product listing in the browser and the application subsequently issues network requests to retrieve page content from `example.com`.

Finally, we derive user transparency rules that correlate network traffic with user-visible actions. These rules estimate whether network connections correspond to explicit user interaction, previously visited content, or known system background behavior. We define a set of rules that capture different forms of observable evidence. The rule `rule_exact_domain_in_text` checks whether the destination domain appears directly in UI input fields (e.g., the browser address bar). The rule `rule_domain_related_text` matches domains with previously observed UI text associated with earlier visits to the same site. Interaction-based rules include `rule_edittext`, which identifies whether the user interacted with an editable text field prior to the connection, `rule_clicked`,

which detects a recent user click on a link or button, and `rule_focused`, which checks whether the user interface contained an active or focused element when the connection occurred.

For example, when a user clicks on a hyperlink or product listing in the browser, the UI typically generates an event such as `TYPE_VIEW_CLICKED` on elements like `android.view.View` or `android.widget.ImageButton`. This interaction often corresponds to selecting a link (e.g., an `<a href>` element) that navigates to a new page. Shortly afterward, the browser issues HTTP requests whose responses include structured content such as `text/html`, `application/javascript`, or `application/json` describing the destination page and its embedded resources (e.g., images, scripts, or iframes). This temporal correlation between UI interaction events, link targets, and resulting network responses provides observable evidence that the network activity is consistent with user-driven browsing behavior.

Additional rules capture evidence derived from network responses or previously observed behavior. The rule `rule_dom_destination` checks whether the HTTP response payload references the destination domain through explicit external resources such as `<a href>`, `<img src>`, or `<iframe src>` elements. The rule `rule_system_background` captures domains that consistently appear during application idle periods and are therefore considered benign background activity learned during training. These rules collectively contribute evidence to a transparency predicate, denoted as `isDisclosedToUser()`, which evaluates whether a network connection can be attributed to observable user activity.

Beyond viewing enforcement purely as destination filtering, we interpret covert traffic as a lack of user-visible justification. We therefore evaluate `isDisclosedToUser()` by aggregating the above rules: the predicate returns true if any rule provides evidence linking the connection to user-visible behavior, and false otherwise. At enforcement time, a connection is allowed only if `isDisclosedToUser()` evaluates to true; otherwise, it is denied. This formulation aligns policy decisions with end-user expectations and targets stealthy behaviors that attempt to blend into legitimate application traffic.

V. Evaluation

We aim to assess how our approach supports IT operators in generating and managing security policies within AppJudicator. In particular, we evaluate the usability and effectiveness of the resulting policies. Using the automated data collection tool described in the previous section IV-A, we replay 1,122 AITW episodes over approximately seven hours on a physical Moto G Power Android phone with eight 2300 MHz cores and 4 GB of RAM, collecting 65,771 decrypted HTTP request–response pairs and 47,353 UI interaction records. These traces serve as the training data for policy generation. We further generate separate legitimate and malicious test datasets to evaluate the effectiveness of the resulting policies.

A. Decrypted vs. Plaintext Training Policy Evaluation

In this subsection, we explore the research question: *When policies are trained from application-specific traces, does incorporating both plaintext-observable metadata and decrypted HTTP data improve deployable enforcement accuracy compared to training on plaintext-only features?* We evaluate enforcement accuracy using 1,122 training episodes and a disjoint test set consisting of benign traffic from newly replayed episodes, which complete natural-language goals that are distinct from the training episodes, and covert malicious traffic generated by MacroDroid [35]. We compare two allow-list policies constructed during training: (i) a *plaintext* policy that relies only on features observable without HTTPS interception, and (ii) a *plaintext+decrypted* policy that additionally incorporates selected HTTP header attributes obtained through trusted interception. During enforcement, observed connections are matched against the policy rules derived from the training traces to determine whether the connection should be allowed or denied.

To evaluate robustness against covert activity, we generate background HTTP traffic using MacroDroid [35], which can issue network requests without an associated user interaction. We create five macro scripts populated with URLs drawn from the abuse.ch malicious URL list [39] and include common probing patterns in the URI (e.g., rapid GET requests to `/admin`, `/login`, `/api/v1/status`, and randomized endpoints such as `/track`, `/ping`, and `/keepalive`). Each script produces thousands of requests, forming a covert malicious test data set.

Table II reports the resulting confusion matrices. Using only plaintext features yields strong security properties with zero false allows but lower benign acceptance, correctly allowing 95.1% of legitimate records (1,520 out of 1,598). Incorporating decrypted HTTP header fingerprints significantly reduces false denials from 78 to 3, improving benign acceptance to 99.8% (1,595 out of 1,598) while maintaining perfect blocking of covert malicious traffic (1,378 out of 1,378). Both allow-list policies therefore achieve 100% correct denies on the MacroDroid-generated traffic. Intuitively, these covert requests lack the expected UI context and fail to match the learned policy rules. Overall, the results indicate that automated policy training effectively blocks covert background traffic, while the inclusion of decrypted HTTP header information primarily improves benign acceptance by reducing collateral blocking.

We next evaluate automatically extracted ABAC policies derived solely from plaintext fields. Each training record is transformed into an ABAC tuple consisting of subject, action, and resource attributes. The subject attributes capture application identity and UI context (e.g., `AppName`, `PackageName`, and `ClassName`); the action attributes correspond to user interface events (e.g., `EventType` such as `TYPE_VIEW_FOCUSED` or `TYPE_VIEW_CLICKED`); and

TABLE II
CONFUSION MATRICES FOR EVALUATED POLICIES.

Decision type	Allow-list (Plaintext)	Allow-list (+ Decrypted)	ABAC (Plaintext)
Correct allow	1,520	1,595	1,470
Correct deny	1,378	1,378	1,378
Incorrect allow	0	0	0
Incorrect deny	78	3	128

the resource attributes describe network endpoints observed in the traffic (e.g., `Remote_IP`, `Remote_Port`, and `Domain`). Each record therefore produces an ABAC tuple containing at least one attribute from each category, such as (`AppName`, `EventType`, `Domain`) or (`PackageName`, `EventType`, `Domain`). We then evaluate these policies using both benign test traces and the MacroDroid-generated malicious traffic described earlier.

Table II summarizes performance. The ABAC policy blocks all covert malicious records (1,378 out of 1,378) but correctly allows 92.0% of legitimate records (1,470 out of 1,598), while incurring 128 false denials. Compared to the fingerprinted allow-lists, ABAC improves interpretability but is more conservative on benign traffic in our current formulation.

B. User Transparency Rule Evaluation

In this subsection, we investigate the research question: *can we apply "disclosure to the end user" as a measurable attribute—derived from UI events and network evidence—to enable policy decisions to block covert traffic?* We evaluate user transparency rules that justify network activity based on observable user interaction and browsing context. These rules incorporate several sources of evidence, including UI-derived signals (e.g., recent click events, focused input fields, and destination domains appearing in on-screen text), web-page structure evidence derived from the Document Object Model (DOM), and learned background behavior observed during application idle periods. A network connection is considered transparent if there exists observable evidence linking it to user-visible activity or to previously learned benign system behavior. We therefore measure how often these rules fire on legitimate traffic generated by AITW workflow replays and on covert malicious traffic generated by MacroDroid.

To extend this evaluation to more realistic adversaries that combine background activity with user interaction, we further implement and deploy an Android trojan horse spyware application based on prior work [36]. This malware requests sensitive permissions (e.g., SMS, contacts, and call logs) and exfiltrates collected data to a remote command-and-control server through persistent background tasks. It also presents a user interface and leverages social engineering techniques to justify permission requests, thereby introducing limited but realistic user interaction.

Table III summarizes the occurrence of each transparency rule in legitimate and malicious traffic. Overall, 97.2% of legitimate records satisfy at least one transparency rule, while only 1.9% of MacroDroid traffic and 2.7% of the Trojan-generated traffic do so. This large separation indicates that the transparency predicate effectively captures user-visible or expected application behavior, while covert background activity rarely produces the observable signals required to satisfy these rules.

One source of legitimate activity arises from background system behavior, since modern applications often generate network traffic without explicit user interaction (e.g., update checks or synchronization). The `rule_system_background` rule captures domains that consistently appear during application idle periods, accounting for 18.3% of legitimate records and none of the malicious ones (Table III). The remaining transparency signals arise primarily from user interactions and page structure. UI-driven rules explain a large fraction of legitimate traffic: `rule_clicked` fires on 42.5% of records, while `rule_focused` (31.1%) and `rule_edittext` (11.5%) capture requests associated with active user input. Text–domain correlation rules further identify cases where the destination domain appears in visible UI content, with `rule_domain_related_text` firing on 20.1% of legitimate records and `rule_exact_domain_in_text` on 16.5%. In addition, the `rule_dom_destination` rule detects references to destination domains embedded in HTML page markup (e.g., hyperlinks or embedded resources), firing on 23.0% of legitimate records and only 1.9% of MacroDroid traffic and none of the Trojan traffic.

Our experiment follows a representative user scenario indicated in that prior work [36]: the user installs the application, grants requested permissions, interacts with the Trojan interface for a short period, and then leaves the device idle while the malware continues operating in the background. It captures both interaction-driven and covert phases of Trojan behavior.

The results show that Trojan-generated traffic remains largely non-transparent despite the presence of user interaction. Only 2.7% of records satisfy any transparency rule, with minimal contributions from UI-based signals: `rule_clicked` (2.3%) and `rule_focused` (0.4%). No matches are observed for text-based or domain-correlation rules, indicating that the spyware's network activity is not meaningfully tied to user-visible content. Compared to MacroDroid, which produces purely background traffic, the Trojan introduces a small number of UI-correlated events; however, the vast majority of its traffic remains covert and unexplainable by user behavior.

While a sufficiently sophisticated Trojan could attempt to evade these policies by generating extensive user interactions or synthesizing realistic UI events, doing so introduces a fundamental trade-off. To appear transparent, the malware must meaningfully engage the user and maintain consistent and aligned UI–network correlations over time. Such behavior increases the risk of user suspicion, as unnatural interaction patterns are difficult to disguise without affecting usability. In contrast, covert attacks rely on minimizing user awareness and interaction. Our results therefore suggest that transparency-

TABLE III
OCCURRENCE RATES OF TRANSPARENCY RULES IN LEGITIMATE AND COVERT MALICIOUS TRAFFIC. LEGITIMATE DATASET: 63,922 RECORDS. MALICIOUS DATASET: 63,408 (MACRODROID [35]) + 3,210 (ANDROID TROJAN HORSE SPYWARE [36]) RECORDS.

Rule	Legitimate	Malicious	
		MacroDroid	Trojan
Any rule	62,130 (97.2%)	1,200 (1.9%)	86 (2.7%)
rule_exact_domain_in_text	10,563 (16.5%)	0 (0.0%)	0 (0.0%)
rule_domain_related_text	12,871 (20.1%)	0 (0.0%)	0 (0.0%)
rule_edittext	7,355 (11.5%)	0 (0.0%)	0 (0.0%)
rule_clicked	27,145 (42.5%)	0 (0.0%)	74 (2.3%)
rule_focused	19,876 (31.1%)	0 (0.0%)	12 (0.4%)
rule_dom_destination	14,693 (23.0%)	1,200 (1.9%)	0 (0.0%)
rule_system_background	11,679 (18.3%)	0 (0.0%)	0 (0.0%)

based policies raise the bar for attackers: evasion requires not only technical capabilities but also convincing user-facing behavior that is difficult to achieve without exposing the attack.

VI. DISCUSSION

We further analyze the implications of our results for adversarial behavior, user interaction, and persistent threats in mobile environments.

A. Implications for Adversarial Behavior and User Interaction

In our Trojan experiments, the malware presents a user interface and generates limited user interaction before continuing its background exfiltration behavior. This setting is stronger than purely background MacroDroid traffic because it gives the adversary some opportunity to appear user-driven. Even under this condition, only 2.7% of Trojan-generated records satisfy any transparency rule, compared with 97.2% of legitimate traffic. It suggests that limited user interaction is not sufficient to make persistent malicious networking appear transparent.

One possible extension is to enhance the Trojan so that it generates more user interaction activities. However, this would also change the nature of the attack. A Trojan that continuously asks for Android permissions, engages the user, displays meaningful content, and aligns each network request with plausible UI events is no longer purely covert. It must trade stealth for interaction. Prior work on Android permissions shows that users do not always understand permission warnings, but permission prompts and visible behaviors still represent points at which users may notice suspicious activity or abandon an application [40]. Thus, forcing malware to become more interactive increases its exposure surface.

The transparency results also have implications for the advanced persistent threat (APT) on mobile devices. Persistent network activity is easier to justify when an application is persistently used. However, many mobile applications have short active lifetimes after installation; industry retention reports consistently show substantial early abandonment and uninstall rates for mobile apps [41]. This observation matters for BYOD enforcement: if a rarely used application continues producing network traffic long after the user's last visible interaction, that traffic becomes increasingly difficult to justify through UI-derived transparency evidence. Our framework does not claim that persistent threats are impossible. Rather, it constrains the attacker's strategy and increases the requirements for success. To maintain persistent networking without being denied, a malicious application must warrant recurring user engagement with legitimate-looking activity.

B. Limitations and Future Directions

Our current Trojan experiment demonstrates that an application with limited user interaction does not substantially weaken transparency-based enforcement. However, adversaries have additional evasion strategies. A more adaptive Trojan could attempt to synthesize UI events, delay exfiltration until immediately after user actions, or embed malicious communication inside apparently useful functionality. Evaluating such adversaries is an important direction for future work. However, to evade transparency rules, malware must create stronger and more persistent correlations between user-visible behavior and network activity. This requirement raises the bar beyond simple background exfiltration.

Prior work has shown that Android permission warnings alone are often insufficient for correct security decisions [40]. A more recent study [42] further indicates that even modern privacy disclosures are frequently overlooked or misunderstood by users. If security decisions are exposed to users in an understandable way, they may further increase the cost of stealthy behavior by making suspicious applications more visible. Future work may wish to study user-facing notifications related to security decisions.

VII. CONCLUSION

This work presents an automated policy training framework for Android SDN enforcement using UI contexts and network traces. Our results demonstrate that the inclusion of decrypted HTTPS information increases the accuracy of automatically generated allow-list policies to 99.8%, while maintaining complete blocking of covert malicious traffic. We further show that attribute-based and transparency-based policies provide interpretability by linking user actions to network behavior. Experiments with a trojan horse spyware indicate that even when malware introduces limited user interaction, most of its traffic remains non-transparent and detectable. While more sophisticated adversaries could attempt to evade detection by generating extensive or realistic UI activity, doing so creates a trade-off between stealth and user engagement, increasing the risk of user suspicion. Overall, automated policy training enables deployable, accurate, and maintainable security policies for BYOD environments while raising the bar for adversaries.

REFERENCES

[1] 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

[2] K. Glowinski, C. Gossmann, and D. Strümpf, "Analysis of a cloud-based mobile device management solution on android phones: technological and organizational aspects," *SN Applied Sciences*, vol. 2, pp. 1–18, 2020.

[3] S. Liu, C. A. Shue, J. P. Petitti, Y. Lei, and Y. Liu, "Mobile SDNs: Associating end-user commands with network flows in android devices," *IET Communications*, vol. 19, no. 1, p. e70047, 2025. [Online]. Available: https://ietresearch.onlinelibrary.wiley.com/doi/abs/10.1049/cmu2.70047

[4] Z. Chuluundorj, C. R. Taylor, R. J. Walls, and C. A. Shue, "Can the user help? Leveraging user actions for network profiling," in *International Conference on Software Defined Systems (SDS)*. IEEE, 2021, pp. 1–8.

[5] Appium Developers, "Introduction to Appium," https://appium.io/docs/en/latest/, 2025.

[6] R. Coppola, M. Morisio, and M. Torchiano, "Mobile gui testing fragility: A study on open-source android applications," *IEEE Transactions on Reliability*, vol. 68, no. 1, pp. 67–90, 2019.

[7] C. Rawles, A. Li, D. Rodriguez, O. Riva, and T. Lillicrap, "Androidinthewild: A large-scale dataset for android device control," *Advances in Neural Information Processing Systems*, vol. 36, pp. 59 708–59 728, 2023.

[8] Apple Inc., "Device management client schema," 2023, available at https://github.com/apple/device-management.

[9] Microsoft Corporation, "Conditional access overview," 2023, available at https://learn.microsoft.com/en-us/entra/identity/conditional-access/overview.

[10] B. Deka, Z. Huang, C. Franzen, J. Hibschman, D. Afergan, Y. Li, J. Nichols, and R. Kumar, "Rico: A mobile app dataset for building data-driven design applications," in *Proceedings of ACM Symposium on User Interface Software and Technology*, New York, NY, USA, 2017, p. 845–854.

[11] A. Burns, D. Arsan, S. Agrawal, R. Kumar, K. Saenko, and B. A. Plummer, "Mobile app tasks with iterative feedback (MoTIF): Addressing task feasibility in interactive visual environments," 2021. [Online]. Available: https://arxiv.org/abs/2104.08560

[12] C. Bai, X. Zang, Y. Xu, S. Sunkara, A. Rastogi, J. Chen, and B. A. y Arcas, "Uibert: Learning generic multimodal representations for ui understanding," 2021. [Online]. Available: https://arxiv.org/abs/2107.13731

[13] S. G. Venkatesh, P. Talukdar, and S. Narayanan, "UGIF: Ui grounded instruction following," 2023. [Online]. Available: https://arxiv.org/abs/2211.07615

[14] J. Zhang, J. Wu, T. Yihua, M. Liao, N. Xu, X. Xiao, Z. Wei, and D. Tang, "Android in the zoo: Chain-of-action-thought for GUI agents," in *Findings of the Association for Computational Linguistics: EMNLP 2024*, Nov. 2024, pp. 12 016–12 031.

[15] Y. Chai, S. Huang, Y. Niu, H. Xiao, L. Liu, G. Wang, D. Zhang, S. Ren, and H. Li, "AMEX: Android multi-annotation expo dataset for mobile GUI agents," in *Findings of the Association for Computational Linguistics: ACL 2025*, Jul. 2025, pp. 2138–2156.

[16] I. Guarino, D. Ciuonzo, A. Montieri, and A. Pescapè, "Mirage-appact-2024: A novel dataset for mobile app and activity traffic analysis," in *International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*, 2024, pp. 663–666.

[17] J. Chen, A. Swearngin, J. Wu, T. Barik, J. Nichols, and X. Zhang, "Extracting replayable interactions from videos of mobile app usage," 2022. [Online]. Available: https://arxiv.org/abs/2207.04165

[18] F. Huang, G. Li, T. Li, and Y. Li, "Automatic macro mining from interaction traces at scale," in *Proceedings of the CHI Conference on Human Factors in Computing Systems*, ser. CHI '24, New York, NY, USA, 2024.

[19] Z. Song, S. M. H. Mansur, R. Rathnasuriya, Y. Fatima, W. Yang, K. Moran, and W. Lam, "Can you mimic me? exploring the use of android record & replay tools in debugging," in *IEEE/ACM International Conference on Mobile Software Engineering and Systems (MOBILESoft)*, 2025, pp. 32–43.

[20] S. Thorve, C. Sreshtha, and N. Meng, "An empirical study of flaky tests in android apps," in *IEEE International Conference on Software Maintenance and Evolution (ICSME)*, 2018, pp. 534–538.

[21] A. Cockburn, P. Dragicevic, L. Besançon, and C. Gutwin, "Threats of a replication crisis in empirical computer science," *Commun. ACM*, vol. 63, no. 8, p. 70–79, Jul. 2020.

[22] K. Hornbæk, S. S. Sander, J. A. Bargas-Avila, and J. Grue Simonsen, "Is once enough? on the extent and content of replications in human-computer interaction," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ser. CHI '14, New York, NY, USA, 2014, p. 3523–3532.

[23] F. Echtler and M. Häußler, "Open source, open science, and the replication crisis in hci," in *Extended Abstracts of the 2018 CHI Conference on Human Factors in Computing Systems*, ser. CHI EA '18, New York, NY, USA, 2018, p. 1–8.

[24] H. Wen, Y. Li, G. Liu, S. Zhao, T. Yu, T. J.-J. Li, S. Jiang, Y. Liu, Y. Zhang, and Y. Liu, "Autodroid: Llm-powered task automation in android," in *Proceedings of Annual International Conference on Mobile Computing and Networking*, ser. ACM MobiCom '24, New York, NY, USA, 2024, p. 543–557.

[25] C. Zhang, Z. Yang, J. Liu, Y. Li, Y. Han, X. Chen, Z. Huang, B. Fu, and G. Yu, "Appagent: Multimodal agents as smartphone users," in *Proceedings of the CHI Conference on Human Factors in Computing Systems*, ser. CHI '25, New York, NY, USA, 2025.

[26] J. Hoscilowicz and A. Janicki, "ClickAgent: Enhancing UI location capabilities of autonomous agents," in *Proceedings of the 26th Annual Meeting of the Special Interest Group on Discourse and Dialogue*, Aug. 2025, pp. 471–476.

[27] N. Li, X. Qu, J. Zhou, J. Wang, M. Wen, K. Du, X. Lou, Q. Peng, J. Wang, and W. Zhang, "Mobileuse: A gui agent with hierarchical reflection for autonomous mobile operation," 2025. [Online]. Available: https://arxiv.org/abs/2507.16853

[28] B. Xie, R. Shao, G. Chen, K. Zhou, Y. Li, J. Liu, M. Zhang, and L. Nie, "GUI-explorer: Autonomous exploration and mining of transition-aware knowledge for GUI agent," in *Proceedings of the Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Jul. 2025, pp. 5650–5667.

[29] A. Yip, X. Wang, N. Zeldovich, and M. F. Kaashoek, "Improving application security with data flow assertions," in *Proceedings of the ACM SIGOPS Symposium on Operating Systems Principles*, ser. SOSP '09, New York, NY, USA, 2009, p. 291–304.

[30] X. Li, Y. Chen, Z. Lin, X. Wang, and J. H. Chen, "Automatic policy generation for Inter-Service access control of microservices," in *USENIX Security Symposium (USENIX Security 21)*. USENIX Association, Aug. 2021, pp. 3971–3988. [Online]. Available: https://www.usenix.org/conference/usenixsecurity21/presentation/li-xing

[31] S. Xu, Q. Zhou, H. Huang, X. Jia, H. Du, Y. Chen, and Y. Xie, "Log2policy: An approach to generate fine-grained access control rules for microservices from scratch," in *Proceedings of the 39th Annual Computer Security Applications Conference*, ser. ACSAC '23, New York, NY, USA, 2023, p. 229–240.

[32] W. Zhang, D. Bali, J. Kerney, A. Panda, and S. Shenker, "Extracting database access-control policies from web applications," 2024. [Online]. Available: https://arxiv.org/abs/2411.11380

[33] K. Dak Albab, A. Agvanian, A. Aby, C. Tiffany, A. Portland, S. Ridley, and M. Schwarzkopf, "Sesame: Practical end-to-end privacy compliance with policy containers and privacy regions," in *Proceedings of the ACM SIGOPS Symposium on Operating Systems Principles*, ser. SOSP '24, New York, NY, USA, 2024, p. 709–725.

[34] V. C. Hu, D. Ferraiolo, R. Kuhn, A. R. Friedman, A. J. Lang, M. M. Cogdell, A. Schnitzer, K. Sandlin, R. Miller, K. Scarfone *et al.*, "Guide to attribute based access control (abac) definition and considerations (draft)," *NIST special publication*, vol. 800, no. 162, pp. 1–54, 2013.

[35] MacroDroid Developers, "Macrodroid device automation," https://www.macrodroid.com/, 2025.

[36] C. M. Mwange and E. C. Cankaya, "Android trojan horse spyware attack: A practical implementation," in *2024 12th International Symposium on Digital Forensics and Security (ISDFS)*, 2024, pp. 1–5.

[37] 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)*. IEEE, 2023, pp. 118–122.

[38] wanghongenpin, "Proxypin," https://github.com/wanghongenpin/proxypin, 2025.

[39] abuse.ch AG, "Urlhaus," https://urlhaus.abuse.ch/, 2025.

[40] A. P. Felt, E. Ha, S. Egelman, A. Haney, E. Chin, and D. Wagner, "Android permissions: User attention, comprehension, and behavior," in *Proceedings of the eighth symposium on usable privacy and security*, 2012, pp. 1–14.

[41] Business of Apps, "Mobile app retention," https://www.businessofapps.com/guide/mobile-app-retention/, 2025.

[42] Y. Lin, J. Juneja, E. Birrell, and L. F. Cranor, "Data safety vs. app privacy: Comparing the usability of android and ios privacy labels," *Proceedings on Privacy Enhancing Technologies*, vol. 2, pp. 182–210, 2024.