TAPShield: Securing Trigger-Action Platforms against Strong Attackers

Mojtaba Moazen*, Nicolae Paladi†, Adnan Jamil Ahsan*, Musard Balliu*
*KTH Royal Institute of Technology
Email: moazen@kth.se, adnanja@kth.se, musard@kth.se
†CanaryBit AB and Lund University
Email: nicolae@canarybit.eu

***Abstract*—Automation apps enable seamless connection of IoT devices and services to provide useful functionality for end-users. Apps are typically executed on cloud-based Trigger-Action Platforms (TAPs) such as IFTTT and Node-RED, supporting both single- and multi-tenant models. Such models raise security and privacy concerns in the face of cloud attackers and malicious app makers, resulting in massive and uncontrolled exfiltration of sensitive user data.**

To address these concerns, we design TAPShield, an architecture that uses confidential computing and language-level sandboxing to protect user data against untrustworthy TAPs and malicious apps. TAPShield targets JavaScript-driven TAPs built on the Node.js environment and uses trusted execution environments implemented with Intel SGX to protect against cloud attackers. It further uses language-level sandboxes such as vm2 and SandTrap to protect against malicious apps. We implement TAPShield for two popular TAPs, Node-RED and IFTTT, and report on the security, performance, and compatibility trade-offs on a range of real-world apps. Our results show clear security benefits with acceptable performance overhead, while adhering to existing development practices of production-scale TAPs.

***Index Terms*—Trigger-Action Platform, Confidential Computing, Language-Level Sandboxing**

1. Introduction

Automation apps enable seamless integration of IoT devices and services to provide useful automated workflows for end-users. These apps are typically executed on cloud-based Trigger-Action Platforms (TAPs) such as IFTTT [1] and Node-RED [2] to connect trigger services with action services.

For example, the IFTTT app "If motion is detected by my Oco camera turns my Philips Hue bulb on" triggers whenever a smart security camera (Oco camera service) detects motion; it performs the action of turning on a smart lightbulb (Philips Hue service) [3]. To facilitate this integration, TAPs rely on OAuth-based delegation tokens that give them privileges to access trigger- and action-services on behalf of users, and execute reactive applications (IoT apps) in the cloud [4].

While IoT apps offer clear benefits to end-users in a variety of settings, they also pose serious security and privacy risks [5], [6]. IFTTT reports that 18 million users across 140 countries run 90 million apps connected to more than 900 services, ranging from baby monitors, surveillance cameras to cars and social networks, and to large-scale IoT systems like smart cities [7]. The multitude of IoT devices, services, and users makes these systems valuable targets.

Because TAPs rely on cloud back-ends to run IoT apps, an attacker that compromises the cloud environment may gain access to valuable data such as the security camera feed, thus breaching user privacy [8]. Another attack vector stems from the poor isolation between IoT apps from different app makers. Both IFTTT and Node-RED build on the Node.js runtime and use JavaScript to implement IoT apps, called *filter code* and *flows*, respectively. An app by a malicious maker may exfiltrate data from other users running only benign apps, whenever it executes on the same Node.js instance in the cloud. This problem is partially mitigated for Node-RED due to its single-tenant model, executing apps for each user on a separate Node.js instance. To reduce costs under the cloud's economic model, IFTTT instead adopts a multi-tenant model, executing apps from different users on the same Node.js instance. This model allows malicious apps to potentially exfiltrate large amounts of sensitive data from unsuspecting users [9]. Figure 2 and Figure 3 respectively depict the cloud attacker and app-level attacker on example IoT apps.

Recent research has already emphasized the above-mentioned issues under different threat models that affect user privacy in TAPs, including compromised TAPs [10]–[12] and malicious app makers [9], [13]–[15]. Existing solutions are however limited to proof-of-concept or clean-slate implementations, with shortcomings in terms of security, performance, and compatibility. We close this gap by designing, implementing, and evaluating TAPShield, a TAP architecture that protects user data against cloud attackers and malicious app makers.

TAPShield employs Trusted Execution Environments (TEEs) and JavaScript sandboxing mechanisms to automatically deploy and execute IoT workloads, while addressing the challenges of security, performance, and compatibility. TAPShield targets TAPs running on JavaScript runtimes and leverages attestation capabilities of TEEs to verify the integrity and confidentiality of workloads in the presence of cloud attackers. It further provides access control via sandboxing to ensure isolation in a multi-tenant setting, thus protecting against malicious apps.

A key goal of TAPShield is to ensure compatibility with production-scale TAPs with minimal changes to the current development lifecycle of IoT apps. To this end, we deploy TAPShield with popular TAPs - Node-RED and

IFTTT - which use a single- and a multi-tenant model, respectively. Our implementation relies on Gramine as a library OS to execute the underlying Node.js runtime in an Intel SGX TEE and on two language-level sandboxes, SandTrap and vm2, to isolate IoT apps. We conduct a set of experiments to evaluate IFTTT and Node-RED apps on TAPShield.

We evaluate the security and privacy benefits of TAPShield by collecting and executing a list of 60 IoT apps (30 secure apps and their 30 insecure versions), achieving full precision and recall. We further evaluate performance for the single-tenant setting of Node-RED and the multi-tenant setting of IFTTT. Our results show that TAPShield incurs an acceptable runtime (2.2x - 3.18x) and memory overhead (1.03x) for Node-RED. The use of sandboxing in IFTTT is more significant, with vm2 performing better than SandTrap (1.56x). Finally, we evaluate compatibility by running core, community-developed, and most popular Node-RED apps, as well as the 50 IFTTT apps, showing that TAPShield can be readily used with minimal changes to the current development practices. Both TAPShield and the related experiments are available on Github repository. [1]

In summary, the paper offers these contributions:

- We describe a new architecture for Trigger-Action Platforms with a main focus on security and privacy against strong attackers (Section 4).
- We implement TAPShield, a solution for securing production-scale JavaScript-driven TAPs such as IFTTT and Node-RED using Trusted Execution Environments and language-level sandboxing (Section 6).
- We evaluate TAPShield in a thorough experiment with real-world apps, reporting on security and privacy benefits, performance, and compatibility (Section 7).

2. Background

2.1. Trigger-Action Platforms

Trigger-Action Platforms (TAPs) are software solutions that automate actions based on predefined triggers or events. These platforms connect online services and devices, enabling users to create automated IoT workflows. When a trigger occurs, the TAP executes predefined actions. Examples of TAPs include IFTTT, Node-RED, and Microsoft Power Automate, each offering an array of interactions with IoT devices and services.

Node-RED is a single-tenant TAP developed in Node.js, offering a fully open-source framework that empowers users to customize and extend functionality according to their requirements. Node-RED follows the flow-based programming paradigm where users can create an app (flow) by connecting reusable code components (nodes). Each node represents an operation or a service, e.g. sending HTTP request, and flows represent the sequence of operations, e.g. sending a notification for each new email. Node-RED comes with a rich library of prebuilt (core) nodes that cover a wide range of functionalities including communication protocols (HTTP, MQTT, WebSocket), data processing (JSON, CSV), and many more including community-developed flows.

1. https://github.com/KTH-LangSec/TAPShield

IFTTT (If This then That) is another TAP that is developed in Node.js with a multi-tenant architecture. In contrast to Node-RED, which has single-tenant architecture, IFTTT allows multiple users to share a single TAP instance on Node.js. IFTTT apps (applets) provide the core functionality of IFTTT and implement simple conditional statements with ”If This Then That” structure. For example, the applet ”If new photo is posted to Instagram, save photo to Dropbox” connects two services (Instagram and Dropbox) by executing side-effect free JavaScript code called filter code. Filter code connects and customizes two main parts: 1) Trigger: which is an event that starts the applet, e.g. ’If new photo is posted to Instagram’. 2) Action: which specifies what happens when trigger occurs e.g. ’save photo to Dropbox’.

Compared to Node-RED, IFTTT has a simpler structure and lacks support for complex apps. IFTTT is ideal for users seeking easy, plug-and-play automation with minimal setup, while Node-RED offers greater flexibility and power for building and managing complex automation flows.

2.2. Trusted Execution Environments

Operating system kernels provide process memory isolation, preventing processes from accessing each other’s memory. However, this security feature relies on the assumption that the kernel, hypervisor, and operating system are inherently trustworthy. This assumption cannot be guaranteed, especially on a host operated by a Cloud Service Provider. Trusted Execution Environments (TEEs) provide isolation of processes from other software using hardware and firmware mechanisms, relying on a minimal Trusted Computing Base (TCB). Major enterprise platform vendors implemented TEEs, for example Intel SGX, AMD SEV, ARM CCA and NVIDIA Confidential Computing [16]. Moreover, academic research produced alternative approaches, for example Sanctum [17]. The main difference between these solutions is the composition of their TCB, defined by the hardware, firmware and software necessary to provide the required isolation.

Intel Software Guard Extensions (Intel SGX) is one of the widely deployed approaches to TEE implementation. It is a set of instruction set architecture extensions on Intel processors added to provide TEE functionality. Process execution in SGX is done within isolated TEE instances called enclaves, which consist of the Processor Reserved Memory (PRM) that contains the protected software’s code, data and stack. Intel SGX benefits from more mature software support due to its early introduction and widespread industry adoption since 2015. Intel continues to enhance Intel SGX for compatibility with cloud environments, supported by significant investments in development tools, SDKs, and documentation [18].

2.3. Enclave Attestation and Secret Provisioning

Attestation is a procedure ensuring the integrity and identity of an enclave to a remote user. To verify the enclave, Intel SGX employs cryptographic hashes which are stored within designated data structures. Specifically, the SGX hardware computes and securely stores two measurements for an enclave: *MRENCLAVE* and *MRSIGNER*. *MRENCLAVE* is a SHA-256 digest that contains information about the workload and enclave creation logs. These registers protect workload integrity because any modification to the workload or enclave image results in a different hash value. *MRSIGNER* is responsible for storing ”Sealing Identity”, and includes a sealing authority, product ID and version number which indicates the identity of TCB (Trusted Computing Base) and hardware running the enclave. Users can verify these registers to ensure that the workload has not been tampered with and is running in a secure enclave.

Intel SGX provides two types of attestation procedures, remote and local attestation [19]. During remote attestation, the enclave produces evidence checked by a remote verifier, while local attestation is used for two enclaves to check each other's evidence. Moreover, remote attestation has two main types: 1) Intel® Enhanced Privacy ID (EPID) is used for attesting enclaves on client machines running locally, and 2) Data Center Attestation Primitives protocol (DCAP) which can be used for attesting enclaves in a data center environment.

Intel SGX allows to provision encrypted data to applications deployed in an enclave. Secret provisioning is the process of providing decryption keys to a running enclave using a secure channel. Secret provisioning relies on attestation, and only successfully attested enclaves can access secrets to decrypt the application's data.

2.4. Library Operating System

Running arbitrary applications inside an SGX enclave is challenging, since the SGX implementation blocks some system calls. Each application must use the Intel SGX Software Development Kit (SDK) and specific system calls (*ECALL* and *OCALL*) to run in an SGX environment. Library Operating Systems (libOS) facilitate the transparent deployment of existing applications. A libOS is a software that provides the bare necessary functionalities (networking capabilities, I/O, and APIs) to run an application in a specific language runtime, such as Node.js for JavaScript. I/O operations outside the enclave, like file reading, are handled by system calls to the host OS, while the libOS abstracts *ECALL* and *OCALL* commands. There are various LibOSs for SGX like Gramine [20], Occlum [21], and Panoply [22].

Gramine is an open-source libOS, enabling translation of commands into specific system calls of the underlying operating system. This allows applications running on Gramine to interact with the operating system kernel through a lightweight and efficient interface provided by the libOS layer. Gramine provides additional granularity and control over resources, e.g. the mounted files, by giving the option to specify different trust levels. In our context, we will use Gramine for two reasons: first, we built our system on top of the Node.js JavaScript runtime, which requires a libOS to be executed on an enclave. Second, Gramine supports Intel SGX features such as attestation and secret provisioning and offers an interface for configuring enclave environments.

2.5. Sandboxing

Sandboxing in JavaScript can be used to run an application in a restricted environment, limiting access to the application host. The main goal of sandboxing an application is to prevent harmful access and isolate it at various levels. In the context of TAPs, a sandbox becomes crucial when working with a multi-tenant TAP, e.g. IFTTT, where multiple apps are deployed by different users on the same runtime. By executing the TAP's app in a sandbox, we can prevent malicious side effects and accesses at runtime, e.g. writing of values in the global scope, or restricting access to privileged operations, e.g. `require` different APIs [23]. There are two types of JavaScript sandboxes: process- and language-level [23]. Process-level sandboxes like BreakApp [24], Jailed [25], and Isolated-vm [26] use inter-process communication or the V8 engine's isolate interface to limit system process interactions. On the other hand, language-level sandboxing provides lightweight restriction of privileges to untrusted code. vm2 [27] is a sandbox that runs a JavaScript application in a single process. While vm2 prevents basic sandbox escaping by restricting module loading (through an allowlist), SandTrap is another approach that uses vm2 and adds fine-grained access control for enhanced sandbox protection, compatible with TAPs [9]. SandTrap can prevent exfiltration and tampering with values and APIs at the language level by integrating `vm` and structural proxy-based membranes to enforce security policies. Proxy-based membranes use JavaScript Proxies to act as a boundary between components, enabling control of the flow of data between trusted and untrusted components.

3. Motivation and Threat Model

We outline the security challenges in current cloud-based TAPs along with possible attacks which motivate us to develop TAPShield compatible with single- and multi-tenant environments. While TAPs enable users to connect their IoT services and devices, IoT apps require computational resources for data storage and execution. This is facilitated by cloud environments, which provide the infrastructure and resources to execute apps based on user data. Prior research shows that the cloud can be compromised [28]–[30], risking user data and compromising the integrity of the underlying applications. For instance, a compromised cloud environment may alter executable applications or expose sensitive user information to a range of attackers, from internal cloud operators to malicious software running in the same cloud environment.

Another attack vector is the execution of IoT apps in multi-tenant TAPs e.g. IFTTT, enabling malicious apps deployed in the same runtime environment to exfiltrate sensitive data of other benign users [9], [31]. Because multiple apps are executed on top of the same runtime instance, e.g. Node.js, a malicious app can affect the runtime environment via prototype poisoning or execute privileged operations to tamper with shared resources. We illustrate both attack vectors with code-level examples.

Consider the Node-RED app in Figure 1 which takes a string as input and converts it to lowercase, for example, by importing the community-developed node *node-red-contrib-lower*. We use this example to illustrate an attack vector in the single-tenant setting of a compromised cloud environment.

Figure 1. Execution flow of Node-RED app

Listing 1 shows the implementation of the lowercase function node. Node-RED registers the node (line 17) at the beginning of execution and processes the input message to convert it to lowercase (line 6) using the `ToLowerCase()` function. The new message is forwarded to the next node at runtime (line 15).

Let us consider the scenario of a strong cloud attacker that alters the functionality of *node-red-contrib-lower*, adding the highlighted lines of code to the original function.

Listing 1. Lower-Case node implementation (malicious code in orange).

```
1 module.exports = function(RED) {
2         function LowerCaseNode(config) {
3         RED.nodes.createNode(this,config);
4         var node = this;
5         this.on("input", function(msg) {
6         msg.payload = msg.payload.toLowerCase();
7                 const https= require("https")
8                 const options = {
9                 hostname: "attacker.com",
10                ...
11                }
12                const req = https.request(options)
13                req.write(msg.payload)
14                req.end()
15                node.send(msg);})
16                ;}
17 RED.nodes.registerType("lower-case", LowerCaseNode);};
```

This malicious code (lines 7-14) allows extracting sensitive user data such as the input string and sending it to an attacker-controlled endpoint. More broadly, unchecked trust in the cloud provider can lead to massive exfiltration of sensitive user data such as API keys, app configuration files or data pertaining to trigger and action services.

In multi-tenant TAPs like IFTTT, an attacker can compromise the user data despite the trust in the cloud environment. Consider two TAP users Alice (the victim) and Bob (the attacker) who have deployed their IFTTT apps in a trusted cloud environment. Listing 2 shows Alice's app which sets the email body to a private message and forwards it to a trusted email address during a predefined time slot (the Meta object retrieves the current time).

Listing 2. Filter Code deployed by Alice

```
var currentHour = Meta.currentUserTime.hour();
if (currentHour >= 22 || currentHour < 6 ) {
        Email.sendMeEmail.setBody("This is my secret message for you at time :" + currentHour );}
```

Bob deploys a malicious app (Listing 3) targeted to (i) poison the `Email` object by overriding the `Email.sendMeEmail.setBody` method and (ii) log the private message of Alice with `setBody` method. This applet triggers with a button and acts with the IFTTT notification service.

The malicious app exploits the lack of proper isolation of the underlying runtime, Node.js, to modify the `sendEmail` object. It first checks the `_isModified` property (line 1) to prevent the `sendEmail` object from being modified, and sets it back to true once the attack is executed (line 8). The attack stores a reference to the original `setBody` method to ensure the original functionality is maintained (lines 3). It then poisons the prototype of `Email` object by logging the email's body (lines 4-5), and finally, in line 6, calls the `apply` method to restore the original method. This malicious app stealthily logs the body of Alice's email whenever her IoT app is triggered.

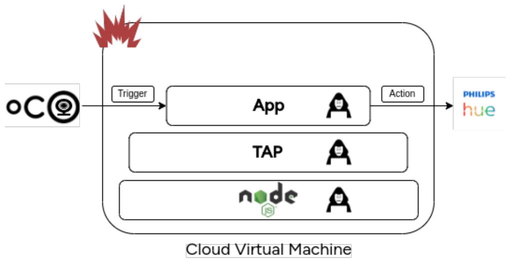

Figure 2. Cloud attacker in single-tenant setting

Listing 3. Filter Code deployed by Bob

```
1 if (!Email.sendMeEmail._isModified)
2     {
3     var StoredSetBody = Email.sendMeEmail.setBody;
4     Email.prototype.sendMeEmail.setBody = function(body) {
5         IfNotifications.sendNotification.setMessage(body)
6         return StoredSetBody.apply(this, arguments);
7     };
8     Email.sendMeEmail._isModified = true;
9     }
```

Alternatively, Alice may install a third-party app that aims to upload images taken from a Security Camera to Google Drive. The malicious app can set the input of the `GoogleDrive.uploadFileFromUrlGoogleDrive` API to `"https://attacker.com/log?"+ encodeURIComponent (PhotoURL)`, thus sending the photo URL to the attacker. Bastys et al. [13] show that value-level attacks are feasible, hence IFTTT does not provide protection against API- and value-level attacks, as illustrated in this section. Notably, IFTTT does not notify users when the app's code changes.

These examples motivate the need for a solution to protect users data against cloud attackers and app-level attackers (apps by malicious app makers).

3.1. Threat Model

Our threat model considers cloud attackers targeting the above-mentioned attack vectors in single- and multi-tenant architectures. We assume a strong attacker model that may tamper with the cloud infrastructure of the TAP, including the IoT app, runtime, OS, kernel, and any other entities operating outside of the Intel SGX enclave. We rely on the security guarantees of the TEE and its configuration, including the Intel SGX SDK, Gramine libOS, and the TAP runtime which is responsible for executing IoT apps. In the paper, we refer to Node.js, TAP and IoT apps as the *application stack* and assume a trusted user with the ability to deploy this stack on the cloud and retain complete control over the machine used for deployment. This machine - or process - executes outside of the cloud and is not affected by potential cloud compromise. We do not trust standard TLS to check the application stack

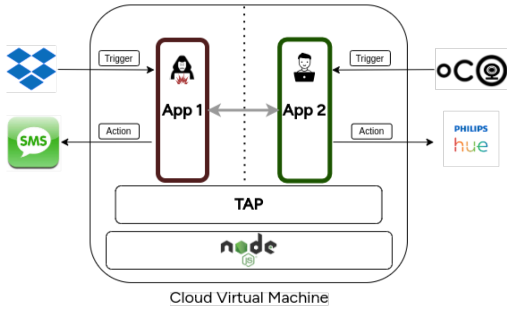

Figure 3. App-level attacker in multi-tenant setting

integrity. The reason is that it can be forged by an attacker within the cloud, potentially leading to data leaks. Instead, we trust the communication provided by Gramine to force the attestation on TLS.

Cloud attacker. Figure 2 illustrates the cloud attacker for a single-tenant architecture. Both the apps and the underlying TAP are deployed on an untrusted cloud and the attacker has full control on the app and its sensitive data (e.g. trigger inputs, API keys, app logs), as well as on the application stack. In the single-tenant model, all apps belong to the same user and are hence in the same trust domain. Here, a user deploys a benign app to turn Philips Hue bulbs on if Oco camera detects motion. A cloud attacker can therefore access the user's credentials and data on Oco camera service or can compromise the integrity of the application stack, including Node.js, TAP, and the app.

App-level attacker. Figure 3 illustrates the attack vector of a malicious app maker for a multi-tenant architecture. In this setup, we have multiple apps by different users which are deployed on trusted cloud and running on a multi-tenant TAP e.g. IFTTT. A malicious app can execute and gain control over the benign app of a different user running on the same TAP and runtime. In our example the malicious app uses SMS and Dropbox as trigger and action services; it can access the global scope of the runtime and access the Oco camera of a victim user.

Protection against side-channel attacks [32]–[34], and denial of the service attacks [35] fall outside the scope of this paper.

3.2. Research Questions

To address the mentioned challenges and threat models, this paper aims to address the following research questions:

- *How to design and implement a solution to secure TAPs against the cloud and app-level attackers?*

To address this question, we propose TAPShield and discuss its architecture and security guarantees in Section 4 and Section 5, respectively, along with the implementation details in Section 6.

- *How to evaluate the benefits of TAPShield in terms of security and privacy, performance, and compatibility?*

In Section 7 we report on a comprehensive evaluation of TAPShield with real-world IoT apps from popular TAPs such as Node-RED and IFTTT.

4. TAPShield Design and Protocol

Our objective is to protect against unauthorized modification of the application stack and prevent leakage of sensitive user data to the cloud attacker. We designed TAPShield to seamlessly integrate with existing sandboxing approaches, enhancing security by preventing unauthorized access by an app-level attacker.

We illustrate the design and architecture of TAPShield in Figure 4. The `Trusted Machine` refers to the deployer machine, potentially running on the end-user's own device. It is trusted and does not exhibit malicious behavior. The Trusted Machine deploys the secure application stack on an untrusted `Cloud` environment to protect it against the attackers we introduced in Section 3.

Data preparation. To deploy the application stack on the cloud, we prepare the data payload, which is divided into two parts: 1) *Trigger-Action Platform* is the bundled code of TAP runtime and its dependencies packaged into a single Node.js file. For Node-RED, the bundled Node.js file also includes community nodes that are included in the flow as a third-party libraries. 2) *Application* that includes `TAP configuration` and `Node.js` binary file, which will be passed to Gramine as the starting execution point inside the enclave. The application component includes `certificates` for TLS communication inside the TAP and `app` which can be either `filter code` or `flow` according to the TAP we are deploying. Following this procedure, we have a deployment-ready application for a Gramine environment (we further refer to such applications as "graminized"). The application consists of 4 main sub-components: a `Sandbox` configuration and generated policies for the app; an `Intel SGX manifest` configuration which defines the TEE specifications; an `Attestation Client` responsible for initiating the attestation process; and the `Encrypted TAP+App` that contains sensitive app data e.g. trigger and action API keys and TLS certificates, encrypted with a symmetric key generated by the `Trusted Machine`. Prior to deployment, the `Trusted Machine` stores a hash of the application stack in a local database signed with its private enclave key. A `Verifier`[2] later uses the signed hash of the application's files to perform integrity checks during the attestation process. This helps detect integrity attacks on the deployed application by a cloud attacker (see Section 3.1).

Enclave initialization and remote attestation. Once the graminized application is deployed, we instantiate and configure an Intel SGX enclave to run the apps. To initialize the enclave, Gramine first reads the provided TEE configuration in `SGX Manifest` to communicate with the `Verifier` endpoint that is already running in the `Trusted Machine`. Next, Gramine generates the application's `Evidence` containing both hardware and application specifications, and a hash of the graminized application (we explain this process in detail in the next section). The `Evidence` is then transferred to the `Attester`. Subsequently, the `Attester` initiates an encrypted communication on top of TLS library (RA-TLS) with the `Verifier`

2. We use terms `verifier`, `attester`, `evidence` and `claim` as defined in the Remote ATtestation procedureS (RATS) Architecture, RFC 9334

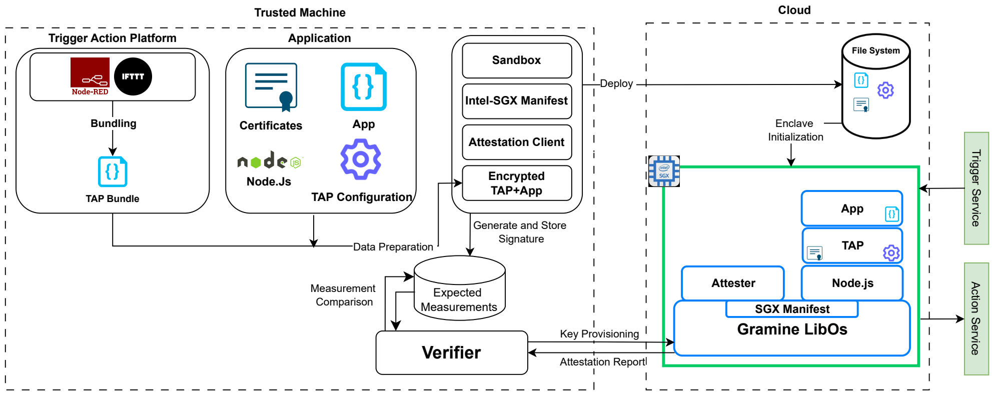

Figure 4. TAPShield architecture

endpoint using an `Attestation Request` and sends the `Evidence` to it for verification. The `Verifier` confirms the validity of the `Evidence` by comparing it against the measurement data stored earlier. This process ensures the integrity of the application within the cloud environment and confirms that it is indeed running on genuine Intel SGX hardware. Once the `Evidence` is verified, the `Verifier` sends the decryption key (stored in `Trusted Machine`) used for encryption through the same secure channel previously established by the `Attester`, enabling continued execution on the cloud by decrypting the necessary data and mounting into the enclave.

Enclave execution. Once the application receives the decryption key from the `Verifier`, execution proceeds by loading the payload files, as specified in the `SGX Manifest` configuration. Gramine is responsible for decrypting payload files and directories during the mounting process. Gramine initiates the mounting process by loading the `Node.js` runtime as the application's entry point, initiating the execution of the TAP runtime. Next, TAP initiates the app's execution by parsing the `filter code` (for IFTTT) or the `flow` file (for Node-RED). Moreover, the graminized application includes a `Sandbox` in the TAP runtime. The `Sandbox` prevents unauthorized apps from accessing each other's data and also shared global scope, thus protecting against the app-level attacker model. Finally, an app can be activated by various `trigger services`, performing corresponding `actions` during the execution by interacting with the enclave.

4.1. Protocol

We illustrate the execution protocol of TAPShield in Figure 5. `Trusted Machine` is used to prepare the graminized application. The `Verifier` running on the `Trusted Machine` verifies the `Evidence` and provisions decryption keys upon successful attestation. The `Cloud` hardware platform supports Intel SGX functionality; it uses a combination of two SGX enclaves – execution enclave (`Attester enclave`) and `Quoting enclave` – to deploy the application and collect a set of `claims` about the SGX enclave and its payload conveyed in the `Evidence` sent to the `Verifier`. We next describe the steps of the execution protocol.

Encryption of application. In Step 1 of Figure 5, we send the sensitive code and data to the `Verifier` that encrypts it with a specified symmetric key. Encryption ensures the confidentiality of code and data against the cloud attacker. This process includes the Node.js binary file, TLS configuration files which are used for communication between app and trigger or service endpoints, application bundle containing the app, and TAP configuration and encryption key. In Step 2, the `Verifier` sends the encrypted files to the trusted user directory for deployment.

Enclave initialization. Having received encrypted payload data from `Verifier`, the application deployment process forwards this data to the `Cloud` environment (Step 3.a). Furthermore, it forwards a set of manifest files to the `Cloud`: Node.manifest, containing TEE specifications; AttestLib with the `Attester` library; TLS certificate authority certificate (crt.ca), used for authenticated TLS communication in the attestation process; a Makefile for enclave building (Step 3.b).

The payload data, manifest and Makefile are used by the `Cloud` to build an Intel SGX enclave, as illustrated in Step 4 of Figure 5. The process of enclave signing requires the enclave to be digitally signed with a trusted certificate before it is loaded and executed within the secure environment.

Before signing the enclave, Intel SGX creates a data structure (*SIGSTRUCT*) to keep a measurement of the enclave's code (application stack). The measurement is a 256-bit hash that identifies the data inside the enclave. Since the enclave is an isolated region of memory (Enclave Page Memory), Intel SGX SDK calculates the enclave measurement again during the execution, and stores it in an *MRENCLAVE* register at Step 4. Next, Intel SGX compares the measurement of each enclave stored in *SIGSTRUCT* against *MRENCLAVE* and, if the measurements match, executes the enclave payload application. Concurrently at Step 5, the application and manifest files (application, TLS certificate authority certificate (crt.ca), and app flow or filter code) are sent to the `Attester` enclave to be signed by hardware. Upon successful completion of this procedure, the `Attester` enclave is initialized and ready to execute the attestation process.

Remote attestation and secret provisioning. Prior to application execution, the `Attester` attests its trustworthi-

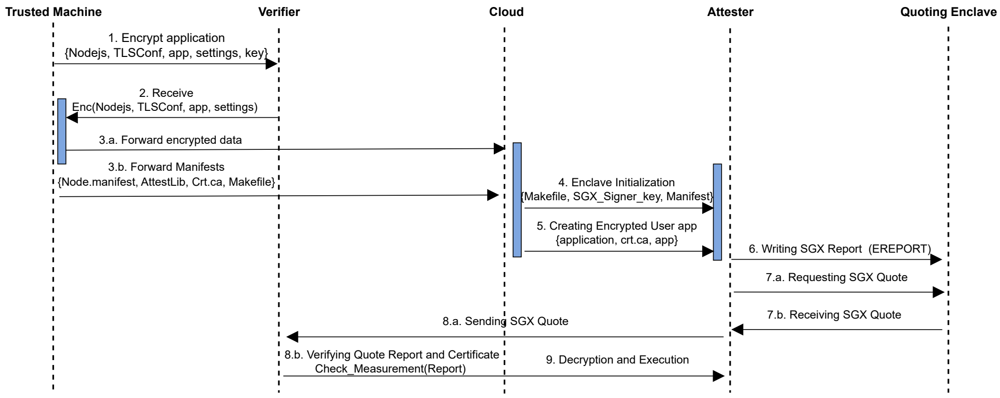

Figure 5. TAPShield protocol

ness to the `Verifier` (steps 6-9 in Figure 5). Attestation allows a remote user (trusted local machine) to verify that the application is running in a genuine SGX-supported environment with an initial enclave state. The data provided to the `Verifier` allows verification of the secure SGX-supported environment and checks the integrity of the application. DCAP remote attestation starts by opening Report file to write an SGX report using *EREPORT* hardware instruction at Step 6. The EREPORT function is used to create an attestation report within the enclave with respect to the manifest files generated before sending to the `Cloud`. This report contains the required measurements that allow the `Verifier` to confirm the hardware genuineness and check enclave (`Attester`) integrity. Once the SGX report is produced, the `Verifier` enclave communicates with the `Quoting` enclave to request SGX Quote (Step 7.a). The `Quoting` enclave authenticates the reports generated by the target enclave and compares them against the stored measurements from enclave initialization. The `Quoting enclave` generates a SGX Quote which can be verified outside of the cloud. SGX Quote uses SGX report which has been generated at Step 7 to create a Quote with embedded SGX report and then sends it to the `Attester` at Step 7.a. In order to generate a Quote with an embedded report, `Quoting` enclave communicates with Provisioning Certification Enclave (PCE) and then PCE sends a periodic request to Intel Provisioning Certification Service (PCS) to obtain the attestation collateral which contains attestation certificates and certificate revocation lists for the `Cloud` SGX machine. Next, the `Quoting` enclave sends a Quote to the target enclave (Step 7.b) that can be used by `Attester` for attestation. Upon receiving the SGX quote at Step 8.a, the `Verifier` checks the measurement embedded into the quote against the measurement stored before deployment to check the integrity of the payload application.

Once the `Verifier` successfully validates the report received from the enclave, it provisions the secret to the enclave (Step 8.b). It delivers the decryption key (key in Step 1) to the `Attester` to execute the application through a secure channel between `Attester` and `Verifier`. Once the `Attester` receives the decryption key from the `verifier` (Step 9) the enclave decrypts the application, data, TLS, and TAP configurations as specified in the SGX manifest, and executes the application.

5. Security Analysis

TAPShield provides protection against two attacker models introduced in Section 3.1. We now address the key research question and clarify how TAPShield protects against each model by referencing the TAPShield design and protocol discussed in Sections 4 and 4.1.

Cloud Attacker

The first concern related to this attacker model is the potential for malicious changes to the application. TAPShield offers protection against any modification by Cloud in the application stack, including Node.js, the TAP runtime, and the app configuration running on top of TAP. As noted in Sections 4 and 4.1, Remote Attestation compares the expected measurements with the evidence received from the cloud, which includes the application's signature (encompassing all configurations). The expected measurements of application stack (*MRENCLAVE*) are signed by a `Trusted Machine`'s private key and stored prior to the application's deployment. This ensures that the expected measurements are accurate and the comparison is reliable. Furthermore, any modification to the application stack will lead to a different measurement from the cloud, causing the attestation to fail.

The second concern regarding the cloud attacker is the potential to access data from the application stack, such as API keys for trigger and action services. To address this issue, we implement `Secret Provisioning`, as described in Section 4 and Step 9 of the protocol. This solution guarantees that decryption keys are provided only if the attestation process is successful. This approach prevents the cloud from accessing sensitive data and ensures that the data is loaded into a trusted enclave, hence the cloud attacker cannot access it.

The third concern is the security of communication during the remote attestation process. TAPShield uses Remote Attestation (`RA-TLS`) interface in order to create a secure communication channel between `Verifier` and `Attester`. RA-TLS allows to embed the `SGX Quote` (described in 4.1) in a X.509 certificate, ensuring that the endpoint (`Attester`) must attest the application to the `Verifier` before any other communication. The RA-TLS interface establishes a secure channel only if the attestation is successful, hence the decryption keys are provisioned through this secure channel.

The final concern relates to enclave communication with the outside world through system calls. IFTTT and Node-RED communicate with the OS and hypervisor in three ways: 1) interaction with file system; 2) communication over a network or through sockets; 3) interprocess communication (IPC). The latter is not supported by Gramine's IPC encryption and it is handled outside of the enclave environment [36]. In TAPShield, TAPs utilize system calls for read/write operations. TAPShield encrypts all input and output files during execution to ensure secure storage and access. When an app accesses files designated as encrypted in the Gramine manifest, Gramine itself manages encryption and decryption within the enclave. We enable this feature by specifying the files and directories in the Gramine manifest, for each use case. This ensures that all user data is encrypted before being written to untrusted host storage, effectively preventing data leakage. Additionally, data read from disk is subject to MAC verification, thus preventing tampering within the untrusted OS. For example, any modification of the output of the `read()` system call is detected during the reception and authentication process. To prevent encrypted content from being swapped between files, Gramine verifies the file's metadata, ensuring that its creation path matches the path specified during the open file operation [37].

The network communication with the outside world is secured by encrypting all network traffic to and from the enclave using TLS communication and certificates, which are instantiated at the beginning of TAP execution. This prevents the OS from accessing requests in plaintext. A compromised OS can introduce delays or drop packets, which falls outside our threat model.

The third communication channel in TAPShield is IPC. Neither Node-RED nor IFTTT instantiate an unencrypted pipe or shared memory outside of the enclave, yet the `Function` node in Node-RED may spawn a process using the `exec` API, which opens a pipe to the outside of the enclave. Since this IPC channel is not encrypted by design, this issue can be mitigated by establishing attestation between the two processes. Otherwise, all other shared memory and pipes in Node.js, such as objects and asynchronous functions, are executed within the same enclave as the Node.js process. Additionally, thread communication for `Worker_thread` APIs in Node.js is also handled within the same enclave by specifying the number of threads in Gramine Manifest file.

We validated the feasibility of attacks without TAPShield by accessing files and modifying the application stack. They are viable under the assumptions of Section 3.1, which require the attacker to have access to the target virtual machine in the cloud.

App-level Attacker

In the app-level attacker model, a malicious app can interact with a benign app, as described in Section 3. The attacker can manipulate APIs, modify values, or load unnecessary modules within Node.js. While the IFTTT engine and apps are not public, we developed a prototype of the IFTTT engine that implements the same functionalities based on prior work [9] and IFTTT documentation. Currently, IFTTT uses Amazon Lambda function [38] for process-level isolation and language-level sandboxing (vm2) to app isolation. While this approach prevents apps from loading unnecessary modules and mitigates prototype poisoning, it does not ensure the sandbox integrity, which a cloud attacker can compromise. In addition, the IFTTT sandbox does not protect apps subject to API- and value-level attacks, as neither Lambda nor the sandbox provides granular control at these levels. This is further exacerbated by the fact that IFTTT users are neither notified nor can observe themselves changes of the filter code. Prior works study the impact of potentially malicious apps in IFTTT. Bastys et al. [13] show that 30% of IFTTT apps can be subject to user's privacy risks and Cobb et al. [39] show that 32% of users install third-party apps based on a friend or a family member suggestion, which can be subject to API- or value-level attacks. We validated the feasibility of API- and value-level attacks on our own IFTTT account and apps, without affecting other users. To protect against these attack vectors, we utilize two sandboxing methods introduced in Section 2.5: vm2 and SandTrap. We integrate the vm2 module into the TAP runtime as an API (see Section 4) to execute filter code. In the vm2 configuration, we disable `eval` and `require` functions to prevent the filter code from loading external modules or executing remote commands at runtime. While vm2 offers basic isolation between apps, it does not fully prevent an app from API- and value-level attacks. To enhance protection, we leverage SandTrap, which protects against tampering with APIs and values during runtime. SandTrap enforces access control policies by generating them before app execution. We illustrate these policies in Figure 4 and explain them in Section 4. TAPShield uses two modes for communication between the IFTTT enclave and outside world. First, the enclave reads and executes the app configuration from file system and second, transmits objects with setter APIs to the IFTTT platform over the network. To protect confidentiality, TAPShield employs an encrypted file system and secure TLS-encrypted communication, as described in Section 5. Due to the simple nature of the apps, IFTTT does not spawn processes via IPC communication.

Integrating the solutions provided for both attacker models ensures sandbox integrity. In the case of SandTrap, we verify the integrity of the generated policies, while without this verification the cloud could still potentially alter the policies or remove the sandbox from the application stack.

6. TAPShield Implementation

We implement TAPShield to secure apps running on Node.js-driven TAPs such as Node-RED and IFTTT. We next provide implementation details for each of the TAPShield components. The system's architecture follows the workflow of Figure 4, which outlines the secure app execution process. Code deployment is automated through an agentless architecture with Ansible, as described in Appendix A.

6.1. Bundling of TAPs

In JavaScript, bundling is the process of combining multiple JavaScript modules into a single file, often referred to as a *bundle*. TAPShield implements this process in two steps. First, it bundles the TAP's application logic including all dependencies based on app features, second it ensures efficient code delivery to the Intel SGX enclave environment.

We use the `esbuild` library which is a fast and reliable Node.js bundler written in Go. Esbuild can minify code, removing unnecessary characters (like whitespaces) and renaming variables, to make the output file smaller and improve memory access speed, which leads to faster execution in the target environment [40]. This results in a single JavaScript file containing the necessary code to execute the application in the cloud. To deploy TAP in a cloud environment, we implement a wrapper designed to deploy the Node-RED and IFTTT runtime on the cloud, adhering to the cloud environment's requirements and TAP configuration and dependencies. This wrapper will serve Node-RED and IFTTT apps and runtime when the application is executed. Node-RED is split into two sub-packages *node-red* and *@node-red*, where the former consists of the Node-RED runtime and the latter provides API functionalities and nodes. A Node-RED flow may only use a subset of all available nodes, thus the bundle script can be configured to only include the required nodes. For IFTTT, we pass the SandTrap, vm2 and TAP scripts to the bundler.

6.2. Application

Depending on the TAP we are deploying on the cloud, the application contains different files and directories. For both Node-RED and IFTTT, the Node.js binary file will be provided by the `Trusted Machine`. To ensure platform compatibility, we use a 64-bit Linux binary file for both the the `Trusted Machine` and untrusted environment. Node-RED further requires signed X.509 certificates (including the entire certificate chain) to establish a secure communication channel using TLS.

IFTTT apps include the filter code which is the JavaScript code of the app, Action-Service and Trigger-Service properties are used to trigger the execution of filter code and execute the action. Node-RED app is a structured JSON file that defines the behavior of each flow. Node configurations are the main building block of the JSON configuration, which will be passed to the Node-RED runtime (*node-red* package) prior to the execution. In node configuration, we specify connections between nodes in workflow in a flow.

6.3. Gramine

In Gramine, the manifest file is a text-based configuration file designed for a specific application based on the application's purpose. It defines the necessary environment and resources required to execute the application within the Gramine framework. The manifest file consists of key-value pairs, as well as more complex structures like tables and arrays, formatted in the TOML syntax. The manifest file contains Secret Provisioning variables which are used for the DCAP attestation and mount points of the required file systems such as Node.js binary file, TLS files and TAP runtime. In each TAP deployment, the sensitive data of the user is encrypted by using `PF-crypt` library. Each TAP deployment needs a `.manifest.sgx` configuration. The application stack signature is generated using the `gramine-manifest` and `gramine-sgx-sign` modules. These modules generate and verify the signature of graminized applications. The signature is stored for later use in verifying the environment during enclave attestation. Next, the application is deployed to the cloud, where it is prepared for execution in the target enclave.

6.4. App Execution

After the application deployment process, we mount the main files of the application, including *main.js*, TAP configuration and app specifications. Since these files are encrypted, we need to decrypt them before the application execution process starts. To decrypt the necessary data and verify the TAP, Gramine starts the `RA-TLS` communication with a verifier service running on the `Trusted Machine`. During attestation, we send the Intel SGX `Quoting enclave` measurements to the `Trusted Machine` to verify the measurements and compare them to the one we stored. We developed the Attestation Server to compare measurements with the one we stored before deployment. After verification, the decryption key is provisioned to the enclave in order to execute the application with decrypted data.

6.5. Challenges

A key goal of TAPShield is to develop a ready-to-use solution against both attacker models of Section 3.1. Similar to other TEE-based approaches, Intel SGX enclaves operate in a restrictive environment that limits the visibility into execution for security reasons. While this restriction enhances security, it makes debugging particularly difficult during TAPShield's development. The lack of detailed runtime debugging information within the enclave convinced us to rely on external logging (e.g. system calls) to reduce potential errors and ensure we address the actual limitations of the TEE, rather than just the design. This limitation slowed down development and required multiple tests to ensure correct functionality.

Another challenge is the implementation of the TAP wrapper under TAPShield's constraints. This is more complicated with Node-RED, which contains multiple modules that should exist in the cloud runtime. In particular, one needs to ensure that all dependencies are appropriately bundled and optimized for a single-file deployment. This is challenging when an app in Node-RED uses third-party nodes that are not part of the Node-RED package.

7. Evaluation

We evaluate TAPShield[3] on a number of IoT apps running on our target TAPs, Node-RED and IFTTT, and assess the security and privacy benefits, performance, and compatibility. Specifically, we answer the following research questions:

- **RQ1:** What are the security and privacy benefits of TAPShield and how can it protect against cloud- and app-level attackers?

3. https://github.com/KTH-LangSec/TAPShield

TABLE 1. NODE-RED SECURITY-RELEVANT FLOWS

Flow	Specification	Included Community Package	Protected Cloud Attacks
Database operations	Contain a set of database operations in MySql	node-red-node-mysql	Change Operation Access sensitive data
Twitch API	Generate twitch bearer token and extract information about twitch channel	node-red-node-group	Leak bearer token Change target channel
Python executor	Execute a python script	Core nodes	Use sensitive Python API Read and Write on user directory
Uploader	Upload a file using Node-RED to the endpoint	Core nodes	Read uploaded file
Calendar bot	Interactive telegram calendar bot	node-red-contrib-telegrambot	Leak Telegram Bot key Change Chat_Id and responses
Google sheet Controller	Read and write on Google sheet	node-red-contrib-viseo-google-spreadsheet node-red-contrib-viseo-google-authentication	Change written data Leak Google AUTH API key
SMS message sender	Send a message to the specific phone number	Core nodes	Change content of message Leak Paid SMS API key
Email notifier	Get a notification when you have a new mail	node-red-contrib-email-out	Read user's emails Leak email's credentials
Object Detection	Machine learning object detection for input image	node-red-contrib-model-asset-exchange	Tamper with the image path Change the predicated result
Todoist	Todoist operations	@foxleigh81/node-red-contrib-todoist-api	Altering the ToDo table Leak Todoist app credentials

- **RQ2:** What is the performance overhead incurred by TAPShield ?
- **RQ3:** How can TAPShield support complex apps and what are its limitations with regards to compatibility?

Experimental setup: We ran our performance evaluation on a Dell Latitude 7440 with a 13th Gen Intel® Core™ i5-1345U CPU and 16GB memory for the trusted machine, and a DC2s-v2 Azure virtual machine with 2 Vcpu core and 8GB memory for the cloud. As execution environment, we use Node.js `V20` and Gramine `V1.7`.

Dataset: Because Node-RED is open-source, we have compiled a dataset of apps containing 4 categories of flows and nodes including core nodes, core flows, security-relevant community flows, and most dependent-upon flows. For the first two categories, we utilized the existing flows developed by Node-RED team, whereas for the other two, we performed an extensive analysis of 2,790 Node-RED flows created by the community of developers. The crawling of this dataset was conducted in June 2024. On the other hand, since IFTTT apps are not publicly available to users, we used a dataset of 208 apps from prior research [9], [13], [41] to evaluate TAPShield.

7.1. RQ1: Security and Privacy Evaluation

Node-RED. In this experiment, we consider security-relevant Node-RED apps (flows) to show the power of TAPShield in protecting sensitive data against the cloud attacker. To do this, we use the following methodology.

We first crawl Node-RED community-developed flows (sorted by download number) and identify flows that do not need a physical device to execute. Then, we specifically choose flows that either involve sensitive operations or use sensitive user information, e.g. read and write on file system. We also use Node-RED community nodes to implement three IFTTT use cases in Node-RED, thus demonstrating the capabilities of Node-RED with simpler apps developed by the IFTTT community. Table 1 illustrates the flows that we use in the experiment. We report the details of each flow specification, including the package names utilized by the flow (core vs community nodes), along with a number of potential attacks (exploitable under the assumptions in Section 3.1) that TAPShield can protect against. We refer to Appendix B for a detailed description of flows.

IFTTT. As we discussed in Section 5, TAPShield is compatible with the IFTTT TAP and two sandboxing approaches, SandTrap and vm2, to protect against app-level attacks (in addition to cloud attacks). Previous research indicates that 30% of IFTTT and 70.40% of Node-RED apps may violate user privacy through data exfiltration [9], [13]. Therefore, protecting against app-level attackers is crucial for TAP security. We evaluate the security benefits of TAPShield on a random selection of 20 apps. We analyzed 10 of the 25 IFTTT apps from SandTrap [9], and examined their benign and insecure versions, which we developed in our benchmark. We used the sensitivity of triggers and actions as primary criteria for app selection. Moreover, we randomly selected 10 apps from the 50 most popular apps, which we discuss later in our compatibility study [42]. Since these apps are benign, we also create their insecure variants, including 2 cases of exfiltration via prototype poisoning, 4 cases of API-level attack, and 4 cases of value tampering.

The resilience against app-level attacks is tested by generating and storing policies for each app, before deployment on the cloud. For exfiltration and API-level attacks, we leverage SandTrap's support for automatic policy generation. For value tampering attacks, we manually define the policies in SandTrap's configuration files, since this is not automatically supported. For example, the *CreateEvernoteWithFeed* app in Table 2 passes the `feedurl` value as input to the `setLinkUrl()` function to set the `evernote` link to `feedUrl`. To prevent an attacker from modifying this input, we add a policy to verify that the `arg` as input in the `setLinkUrl` function is always equal to `feedurl`. We log execution data for the two versions, benign and malicious, and validate TAPShield's effectiveness by analyzing the sandbox logs for app-level attacks and the SGX logs generated during the attestation process for cloud-level attacks.

Table 2 outlines the specification of each IFTTT app and two developed attacks that are prevented by TAPShield in combination with sandboxing. In the first 4 apps, the attacker attempts to exfiltrate data by poisoning the prototype of a shared object within the IFTTT environment. These 4 attacks are similar to the one described in Section 3, but with a different object being poisoned. They enable attackers to exfiltrate sensitive user data, e.g. email body in first use case, and can be mitigated by both vm2 and SandTrap, when integrated with TAPShield.

As shown in the table, *RedditAddSpotify*, *KasaTurnOff*, *ToggleMyLevition*, and *SetColorAllHue* aim to skip an action based on predefined conditions. The attacker modifies the action field with the setter functions in app code. This allows the attacker to change the functionalities of app while the benign user thinks the action was skipped. The next 4 cases are apps that maliciously attempt to skip the corresponding actions, thus tampering with action integrity. For example, *GetRainNotification* is designed to send a notification if rain is expected. This app is intended to set properties and does not include any conditions when Trigger APIs are called. However, in the insecure version, an attacker leverages the `Skip()` function to bypass the action's execution. TAPShield uses generated policies and its properties for each API prototype enforcing policies with SandTrap.

In the next 8 apps, the attacker modifies the value passed to the action service of each app, e.g. changing the input of the `SetMessage()` API in the *GoogleCalendar* use case. Since SandTrap is value-sensitive, TAPShield is secure against value tampering. We remark that vm2 does not provide any policy enforcement at API and value level.

For cloud attacker, we define a series of attacks on both confidentiality and integrity in Table 2. We divide attacks into two types. First, a series of attacks aim to compromise user privacy by accessing trigger and action data, such as uploaded Dropbox files in *DropBox-Email* use case. Second, we define integrity attacks that modify the app, such as removing the skip action in the *KasaTurnOff* use case. Finally, as we illustrated in Section 5, TAPShield protects against both attacks via secret provisioning and remote attestation.

7.2. RQ2: Performance Evaluation

Node-RED. We evaluate the performance of TAPShield with a focus on Node-RED, targeting Node-RED core nodes and core flows. These experiments additionally contribute to evaluate the compatibility of TAPShield with Node-RED, which we discuss in further detail in the next section.

Node-RED core nodes: Node-RED package provides a set of basic nodes with different functionalities and flows in its default version. To evaluate the performance of our tool, we execute a series of flows, each corresponding to a single core node along with the core flows provided by the Node-RED team. In addition to performance, our experiment assesses the compatibility of TAPShield with Node-RED: If a core node is not supported by TAPShield, this implies a limitation of our system. The same argument applies to core flows, which are essentially collections of core nodes.

We find that TAPShield can compile and run successfully all core nodes (and flows), except for node `./storage/23-watch.js`. This node uses `inotify` system call to monitor changes to the filesystem. As of August 2024, Gramine does not support this system call, leading to a failure of flows that use the watch node. In our large-scale analysis of the Node-RED community flows, we find that only 18 out of 2,790 flows utilized the `Watch` node, showing that this limitation impacts very few flows. In summary, our results find that TAPShield supports 34 out of 35 Node-RED core nodes with no changes to the original versions.

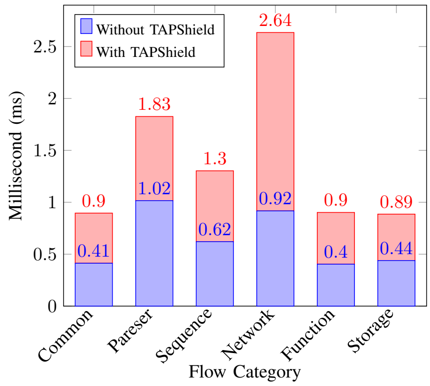

Figure 6. Performance evaluation on core flows

To evaluate the performance overhead of TAPShield on core nodes, we identified the most popular core nodes in community flows. We analyzed a total of 750 flows (sorted based on download number) and identified the top 12 most popular (frequently used) core nodes.

We then identified the trigger function of each node and used two Node.js methods (`console.time()` and `console.timeEnd()`) to measure their execution time with and without TAPShield. Figure 7 In Appendix C shows the average execution time of each node, which we run 20 times. We see an increased performance overhead for `WriteFile, HttpReq, Function` and `LinkCall` nodes, which we discuss later. Ultimately, we find that TAPShield incurs a performance overhead of 2.3x on average compared to execution with Node.js.

Node-RED core flows: The Node-RED repository includes a collection of sample flows designed to showcase the functionality of core nodes. Specifically, it provides 114 sample flows grouped into 6 distinct categories based on their functionality. We run the core flows with TAPShield to measure the execution time overhead. Because the sample flows demonstrate the basic functionality of each node, their performance overhead of all flows in a category is similar. For this reason, we selected 5 flows per category that offered different functionalities. As discussed previously, we excluded the `Watch` flows in `Storage` category. Then we evaluate the trigger execution time for the included nodes in a flow and run each flow 20 times. Then we calculate the average execution time of flows in the same category, since they provide similar functionality. Our results indicate an average increase of 2.2x, as shown in Figure 6. The y-axis represents the average execution time (in milliseconds) for flows within a specific category. This performance is still better than related approaches, such as Walnut [11], which have a best-case increase of 2.9x for a simpler TAP (IFTTT). Moreover, the most time-consuming category (Network nodes) incurs 2.64 ms overhead, which is negligible compared to the security benefits provided by TAPShield.

Discussion: Our experiments show an increase in performance overhead for flows and nodes that perform I/O operations outside the enclave. When comparing the results from core nodes and flows, we find that the bottleneck of using TAPShield occurs when we need to perform I/O operations outside of the enclave. As explained

TABLE 2. IFTTT SECURITY-RELEVANT FILTER CODES

IFTTT Filter Code	Specification	Trigger Service	Action Service	Protected App-level Attack	Protected Cloud-level Attack	Dataset
DropBox-Email	Send email when a new file is uploaded to DropBox	Dropbox	Email	Exfiltrate with email's content	Read uploaded files	SandTrap
MonzoDepositSetAmount	Set amount for Monzo depositing based on weekday	Time	Monzo	Exfiltrate with the amount number	Read deposited amount	SandTrap
StartiRobot	When I leave home, start a cleaning job	Location	iRobot	Exfiltrate the user location	Modify Trigger condition	Most popular
SyncNoteTodoist	Sync Evernote and Todoist	-	EverNote Todoist	Exfiltrate the task content	Modify Todo tasks	Most popular
RedditAddSpotify	Add top songs from Reddit to Spotify	Reddit	Spotify	Create an unwanted playlist	Read Reddit search	SandTrap
KasaTurnOff	Skip turning on Kasa during the day	Time	Kasa	Set speed level instead of skipping	Read Kasa API key	SandTrap
ToggleMyLeviton	Skip MyLeviton lights during the daylight	Time	MyLeviton	Set power instead of skipping	Remove skip event	SandTrap
SetColorAllHue	Skip Hue light coloring on non-rainy days	Weather	Hue	Turn on lights instead of skipping	Modify weather condition	SandTrap
GetRainNotification	Get Notification if tomorrow is rainy	Weather	IfNotifications	Skip notification	Read user location	Most popular
SetNasaWallpaper	Set NASA daily picture as Android wallpaper	NASA	AndroidDevice	Skip on weekdays	Modify sourceUrl	Most popular
WorkHoursTracker	Press a button to track work hours in Google Drive	Button	GoogleSheets	Skip working hours tracking	Modify Google Sheet format	Most popular
AddNasaNewstoReadingList	Add image of the day from NASA to iOS Reading List	NASA	iOS Reading List	Skip adding to reading list	Modify reading list	Most popular
GoogleCalendar	IFTTT notification when an event is not an all-day event	Google Calendar	IFTTT	Tamper with message	Remove title from message	SandTrap
createPhotoPost	Post a photo on Tumblr	Photos	Tumblr	Tamper with Tumblr URL	Replace title of Tumblr post	SandTrap
SkipEwelink	Skip Ewelink switch actions during daylight	Time	Ewelink Smartlife	Tamper with Ewelink's name	Tamper with the SmartLife device name	SandTrap
SkipSlackPost	Skip posting on Slack	Time	Slack	Tamper with URL	Modify a sensitive string	SandTrap
CreateEvernoteWithFeed	Create an Evernote based on a Feed URL	RSS Feed	Evernote	Tamper with Note URL	Read user's notes	Most popular
AddYoutubeLikesSongs	Add songs from videos you like to a Spotify playlist	YouTube	Spotify	Tamper with search query	Read user liked music	Most popular
IosReminder	Sync Quick Note with iOS Reminder	Quick Note	iOS Reminder	Tamper with reminder date	Read Note	Most popular
AddPhototoGoogleDrive	Upload any new photo taken by camera	Camera	Google Drive	Tamper with URL path	Read taken photo	Most popular

in Section 2.4, software isolation with Intel SGX uses the `OCALL` mechanism to secure the application. Each `OCALL` consists of three main commands, `EENTER, host processing` and `EEXIT`. For each system call processing, `EEXIT` executes first to leave the enclave and flush the CPU cache. Then after the processing of system call, `EENTER` performs several checks and requires hardware-interval synchronization of cores. Each `EEXIT` and `EENTER` needs 8000-12000 CPU cycles while the normal system call needs just about 100 cycles [43]. This implies that the CPU spends more cycles to execute applications in a secure environment. Beyond system calls, other factors affecting execution time are the swapping between encrypted (`enclave`) and unencrypted memory, and the usage of MACs for encrypted file system I/O.

In summary, our findings demonstrate that the performance overhead incurred during application execution is inevitable, particularly when the application requires communication with entities outside the enclave. On the other hand, TAPShield can improve the security and privacy by protecting against various attacks that would otherwise be exploited by a cloud attacker.

IFTTT. In this section, we evaluate the performance of TAPShield with sandboxing, aiming to protect against both attacker models described in Section 3.1. To achieve this, we evaluate the 10 IFTTT apps (filter code) of SandTrap dataset provided in Table 2. We execute each app 10 times in 4 different modes and calculate the execution time based on different setups, as shown in Table 3. We measure the overhead caused by executing an IFTTT filtre code within a sandbox. Specifically, for SandTrap, we conducted measurements under two scenarios: 1) each execution involved generating policies dynamically, and 2) policies were pre-generated, allowing execution without additional policy generation overhead. The table reports the average execution time of 10 filter codes in each mode.

Discussion: The performance evaluation reveals a significant overhead due to the use of SandTrap as compared to the original execution with and without the TAPShield (first and second row in Table 3. Importantly, because policies are generated once for each deployed app, the overhead decreases when pre-generating them. This approach removes the need for write operations, allowing execution to rely solely on read operations. Although the time overhead is considerable, 80.38 ms remains acceptable considering that the polling trigger time for IFTTT pro/pro+ users is 5 minutes (1 hour for regular users) [44]. This overhead arises because SandTrap implements policy enforcement by initially storing policies on the filesystem and subsequently reading them during the application's execution. This process requires enclave I/O operations, such as reading and writing on a file (in learning mode), which affects the performance of enclave execution substantially. SandTrap initially reads all policies and then loads them into the memory of the enclave. The execution time increases proportionally with the number of policies generated during the policy generation stage.

Additionally, given that SandTrap uses the Node.js `vm` module to establish isolation between the sandbox and the host [9], another factor contributing to the increased overhead is the virtual machine layer. Running code within a Node.js vm2 sandbox requires switching between the Node.js runtime environment and the TEE environment multiple times during the execution, which causes additional overhead in row 3 of Table 3. This explains the increased execution time when using TAPShield and the vm2 sandbox standalone.

TABLE 3. EVALUATION ON 10 IFTTT APPS IN 4 EXECUTION MODES

Environment	Execution Time (ms)
Node.js	0.17
TAPShield	2.45
TAPShield & vm2	51.51
TAPShield & SandTrap	138.29 (Policy learning: On)
	80.38 (Policy learning: Off)

7.3. RQ3: Compatibility

In this experiment, we consider the ability of TAPShield to execute complex apps, thus evaluating compatibility and answering **RQ3**.

Node-RED. We crawled the Node-RED community flows and sorted them based on the number of nodes they used. Then we manually checked the flows, removing those that needed a specific device or relied on unavailable APIs. We finally selected the 5 most dependent-upon flows from Node-RED community flows, as shown in Table 4. During the manual inspection, we found no restrictions due to TAPShield and thus successfully executed all selected flows. To assess the compatibility at scale, we analyzed the collected flows and discovered that out of 2,790 existing flows, 793 rely solely on core nodes without any community-developed ones. Based on **RQ2**, we know that all core nodes, except the `watch` node, can be executed,

TABLE 4. MOST DEPENDENT UPON FLOWS

Flow Name	Specification	# of Nodes	# of Unique Nodes
Monitoring URL	Web-based application to test URL and different endpoints	206	23
Weather Database	An application to store different weather utilities into MySql	100	10
OPCXML Service	Serve an OPC XML client in order to parse requests	74	15
Weather Quality Service	Weather and water quality MQTT server	70	12
IoT Devices Controller	Control different IoT devices using Telegram bot	53	18

implying these flows can run smoothly with TAPShield. Next, we explored how to automate the execution of the remaining apps that use community-developed nodes. However, automating this process was challenging for different reasons e.g. missing API configurations. As a result, we began manually selecting random apps and configuring them for execution with TAPShield. During this process, We found no limitations with TAPShield, provided the nodes were properly configured, a task that any TAPShield user can manage.

Another challenge was managing API keys, which are generated via the Node-RED user interface. To address this, we modified the library for community-developed nodes to read API keys from memory instead of the user interface.

We refer to Appendix D for a detailed description of the apps' functionality, and discuss here the time and memory overhead when executing them with TAPShield.

Performance Analysis: To assess TAPShield 's time overhead against real-world Node-RED flows, we perform a series of evaluations on selected flows. We execute each flow with TAPShield, followed by a series of predefined actions across 10 different paths. We then log the performance data, including the execution time of each path and the application's memory consumption by the Gramine process at regular intervals. To measure the execution time of each path, we log the execution time of the trigger for each node along the path. We then calculate the execution time of the path as the sum of the execution times of all its nodes, and finally compute the average execution time of the path. Our result is illustrated in Figure 8 in Appendix D, indicating an average increase of 3.18x compared to the application executed without TAPShield. We remark that even with TAPShield, the average execution time of each path is 2.44 ms, which is acceptable for executing real-world examples, especially when considering the security benefits provided by TAPShield.

For the memory overhead analysis, we repeated the same process used for measuring execution time, while concurrently monitoring memory consumption of the Gramine process. We specified 512MB memory for the `enclave_size` using Gramine configuration, yet our monitoring results show that the flows require less memory. We log memory usage during path execution and update the peak memory value whenever the process consumes more memory. The final result for memory consumption of each flow is shown in Figure 9 in Appendix D .

We find that some flows use less memory when executed with TAPShield. This is because the applications running inside enclaves are subject to SGX-specific memory management policies. SGX enclaves are allocated with a fixed enclave memory (EPC - Enclave Page Cache), which is managed differently from regular system memory [20]. EPC is an L3 cache for optimization, leading to multiple swapping when the application does not fit the EPC space. However, EPC allows the application to use processor reserved memory (PRM) instead of regular memory, ensuring that a portion of the application always fits inside the EPC and PRM.

IFTTT. To evaluate the compatibility on the IFTTT platform, we randomly selected 50 apps: 30 of a dataset of 133 apps of prior research [13] and 20 out of 50 most popular IFTTT applets of year 2024 [42]. The complete list of target apps is provided in Table 6 in Appendix D.

Our dataset comprises 27 unique triggers and 35 action services. We extracted the filter code using the method described in prior work [41]. Once the dataset was prepared, we execute the apps 10 times under 4 different execution modes: (1) using a Node.js runtime, (2) running TAPShield without a sandboxing mechanism, and executing with (3) vm2 and (4) SandTrap. For SandTrap, we first run each app in learning mode to generate the policies, which we then stored encrypted on the file system. Performance results are reported in Table 5 in Appendix D. The performance overhead of TAPShield with SandTrap is actually closer to vm2, as the randomly-chosen apps have a simpler structure. During the execution of these 50 apps, we did not encounter any compatibility issues.

8. Related Work

We discuss related works based on our attacker models and highlight how they compare to TAPShield. The key difference is our focus on protecting real-world TAPs, Node-RED and IFTTT, against strong attackers, along with the evaluation on metrics such as security, performance, and compatibility.

Cloud atacker. Recent parallel work by Jegan et al. [10] proposes a clean-slate architecture, TAPDance, to protect privacy of IFTTT apps by means of RISC-V keystone enclaves. TapDance uses attestation to protect the integrity of apps and finds that seamless execution of the TAP is not possible with Keystone because it needs other components e.g. TCP/TLS connection, for communication and a compiler to interpret the language. Instead, TAPShield uses a libOS to solve this problem and additionally isolates apps via sandboxing with no need to run one enclave per app. Zavalyshyn et al. [45] propose a private IoT platform using Intel SGX to secure apps from the cloud. It allows users to control data flows generated by IoT devices and minimizes unnecessary flows between IoT device and cloud. Oak et al. [46] study program partitioning techniques to identify sensitive code regions for enclave execution. Similar ideas can be explored to further improve the performance of TAPShield .

Chen et al. [12] and Schoettler et al. [11] explore secure multi-party computation to ensure confidentiality and integrity of apps. These techniques suffer from high overhead and require architectural changes to the TAPs. Hunt et al. [47] propose integrating Intel SGX with Google's Native Client sandboxing approach to secure distributed systems. They do not support JIT compilation, making it difficult to run Node.js runtime. AccTEE [48] and oak [49] represent an alternative method to run WebAssembly within an enclave, with the goal of offering two-way sandboxing for resource accounting in AccTEE. While this approach is similar to TAPShield, both AccTEE and oak focus more on WebAssembly modules, whereas TAPShield considers Node.js applications. Chiang et al. [50] propose OTAP, an end-to-end encryption protocol between the user and trigger-action services. OTAP keeps the data confidential to TAP while breaking away from the current practices. Moreover, the focus is only on confidentiality.

Other works implement data minimization techniques to limit the amount of private data exchanged with the cloud. minTap [41] uses program analysis to identify the minimal data that is needed for an app to function correctly. Ahmadpanah et al. [51] further optimize this approach via a pull-on-demand method. Xu et al. [52] develop instead a filtering technique for the same purpose. All these approaches are less helpful when private data sharing is needed as part of the app's functionality. PTAP [53] proposes adversarial machine learning to protect against sensitive inferences over public data, yet it does not work with apps that handle private data.

App-level attacker. Several works study the dangers of malicious apps in the context of a trusted cloud. Ahmadpanah et al. [9] design SandTrap, a JavaScript sandbox to isolate apps in multi-tenant settings. We evaluate SandTrap in combination with TAPShield, showing improved security with performance similar to vm2. Melara et al. [54] propose Pyronia, a fine-grained access control method that enforces rules through system calls and memory stack inspection. Pyronia can limit access for each app at function level, whereas TAPShield employs language-level sandboxes. Kang et al. [55] develop IoTBox, a sandboxing system designed for IoT environments. IoTBox runs the apps in a benign environment and records API usage, which is subsequently enforced within a sandbox In contrast, TAPShield does not rely on predefined activities and is immediately deployable. Birgersson et al. [56] use TEEs to secure computations over sensitive data in a multi-user setting, yet they only support simple side-effect free functions. Fernandes et al. [57] and Fan et al. [58] focus on integrity of rule execution in IFTTT platform. Other works propose program analysis and information flow control to identify malicious apps in a single-tenant setting [13]–[15], [59], [60].

9. Conclusion

We described the design and implementation of TAPShield, a toolchain for protecting security and privacy of cloud-based IoT apps against strong attackers. TAPShield builds on recent advances in workload isolation by using trusted execution environments and can optionally use language-level sandboxing to secure apps deployed on multi-tenant cloud environments. Drawing on the key metrics of security, performance, and compatibility, we conducted thorough experiments on two production-scale platforms, Node-RED and IFTTT, showing that TAPShield improves security and privacy with moderate performance overhead and no disruptions to current development practices. Future work includes studies with IoT app developers and users, as well as experiments with trusted execution environment implementations, beyond Intel SGX.

Acknowledgment

We thank the anonymous reviewers for their constructive feedback. We also express our gratitude to Mohammad Ahmadpanah, Javier Cabrera-Arteaga, and Andrei Sabelfeld for their insightful feedback. This work was supported by the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation, the Swedish Research Council (VR), and Digital Futures.

References

[1] IFTTT, "If This Then That," `https://ifttt.com`, 2024.

[2] Node-RED, "Node-RED," `https://nodered.org/`, 2024.

[3] IFTTT, "If motion is detected by my Oco camera turn my Philips Hue bulb on," `https://ifttt.com/applets/S8FuwsDd`, 2024.

[4] E. Fernandes, A. Rahmati, J. Jung, and A. Prakash, "Decentralized Action Integrity for Trigger-Action IoT Platforms," in *Network and Distributed System Security (NDSS) Symposium*, 2018.

[5] M. Balliu, I. Bastys, and A. Sabelfeld, "Securing IoT Apps," *IEEE Security and Privacy (S&P)*, 2019.

[6] Z. B. Celik, E. Fernandes, E. Pauley, G. Tan, and P. D. McDaniel, "Program Analysis of Commodity IoT Applications for Security and Privacy: Challenges and Opportunities," *ACM Computing Surveys (CSUR)*, 2019.

[7] IFTTT, "The Internet of Everything," `https://ifttt.com/explore/the-Internet-of-Everything`, 2024.

[8] E. Fernandes, J. Jung, and A. Prakash, "Security analysis of emerging smart home applications," in *IEEE Symposium on Security and Privacy, S&P*, 2016.

[9] M. M. Ahmadpanah, D. Hedin, M. Balliu, L. E. Olsson, and A. Sabelfeld, "SandTrap: Securing JavaScript-driven Trigger-Action platforms," in *USENIX Security Symposium*, 2021.

[10] D. S. Jegan, M. Swift, and E. Fernandes, "Architecting trigger-action platforms for security, performance and functionality," in *Network and Distributed System Security (NDSS) Symposium*, 2024.

[11] S.Schoettler, A.Thompson, R.Gopalakrishna, and T.Gupta, "Walnut: A low-trust trigger-action platform," *ArXiv*, 2020.

[12] Y. Chen, A. R. Chowdhury, R. Wang, A. Sabelfeld, R. Chatterjee, and E. Fernandes, "Data privacy in trigger-action systems," in *IEEE Symposium on Security and Privacy (SP)*, 2021.

[13] I. Bastys, M. Balliu, and A. Sabelfeld, "If this then what? controlling flows in iot apps," in *ACM SIGSAC Conference on Computer and Communications Security*, 2018.

[14] Z. B. Celik, L. Babun, A. K. Sikder, H. Aksu, G. Tan, P. McDaniel, and A. S. Uluagac, "Sensitive Information Tracking in Commodity IoT," in *USENIX Security Symposium*, 2018.

[15] Z. Celik, G. Tan, and P. D. McDaniel, "IoTGuard: Dynamic Enforcement of Security and Safety Policy in Commodity IoT," in *Network and Distributed System Security (NDSS) Symposium*, 2019.

[16] R. Guanciale, N. Paladi, and A. Vahidi, "SoK: Confidential Quartet - Comparison of Platforms for Virtualization-Based Confidential Computing," in *IEEE International Symposium on Secure and Private Execution Environment Design (SEED)*, 2022.

[17] V. Costan, I. Lebedev, and S. Devadas, "Sanctum: Minimal hardware extensions for strong software isolation," in *USENIX Security Symposium*, 2016.

[18] Intel Corporation, "Rising to the challenge: Data security with intel confidential computing," https://community.intel.com/t5/blogs/blogarticleprintpage/blog-id/blog-security/article-id/47, 2022, [Accessed: 2024-08-28].

[19] Intel, "Attestation Services for Intel® Software Guard Extensions," https://www.intel.com/content/www/us/en/developer/tools/software-guard-extensions/attestation-services.html, n.d., [Accessed 20-08-2024].

[20] C. che Tsai, D. E. Porter, and M. Vij, "Graphene-SGX: A practical library OS for unmodified applications on SGX," in *USENIX Annual Technical Conference*, 2017.

[21] Y. Shen, H. Tian, Y. Chen, K. Chen, R. Wang, Y. Xu, Y. Xia, and S. Yan, "Occlum: Secure and efficient multitasking inside a single enclave of intel sgx," in *International Conference on Architectural Support for Programming Languages and Operating Systems*, 2020.

[22] S. Shinde, D. Le, S. Tople, and P. Saxena, "Panoply: Low-tcb linux applications with sgx enclaves," in *Network and Distributed System Security (NDSS) Symposium*, 2017.

[23] A. AlHamdan and C.-A. Staicu, "SandDriller: A Fully-Automated approach for testing Language-Based JavaScript sandboxes," in *USENIX Security Symposium*, 2023.

[24] N. Vasilakis, B. Karel, N. Roessler, N. Dautenhahn, A. DeHon, and J. M. Smith, "BreakApp: Automated, flexible application compartmentalization." in *NDSS*, 2018.

[25] Jailed, "jailed: execute untrusted code with custom permissions," https://github.com/asvd/jailed, n.d., [Accessed 07-08-2024].

[26] Isolated VM, "Isolated JS environments for node.js," https://github.com/laverdet/isolated-vm, 2024.

[27] vm2, "vm2 npm package," https://www.npmjs.com/package/vm2, n.d., [Accessed 20-08-2024].

[28] T. Jones, "The Cognyte Cloud Data Breach 5 Billion Records Leaked," https://medium.com/nerd-for-tech/86a7114cef06, 2022.

[29] "What happened in the Raychat data breach?" https://www.twingate.com/blog/tips/raychat-data-breach, 2024.

[30] Y. Jie, "Alibaba Falls Victim to Chinese Web Crawler in Large Data Leak," https://on.wsj.com/3cJz9ED.

[31] M. Abbadini, D. Facchinetti, G. Oldani, M. Rossi, and S. Paraboschi, "Natisand: Native code sandboxing for javascript runtimes," in *International Symposium on Research in Attacks, Intrusions and Defenses*, 2023.

[32] S. Carré, A. Facon, S. Guilley, S. Takarabt, A. Schaub, and Y. Souissi, "Cache-timing attack detection and prevention: Application to crypto libs and pqc," in *Constructive Side-Channel Analysis and Secure Design (COSADE)*, 2019.

[33] J. Wang, Y. Cheng, Q. Li, and Y. Jiang, "Interface-based side channel attack against intel sgx," *ArXiv*, 2018.

[34] W. Lee, T. Kim, and Y. Shin, "Poster: On the feasibility of inferring sgx execution through pmu," in *ACM Asia Conference on Computer and Communications Security*, 2024.

[35] S. Dong, K. Abbas, and R. Jain, "A survey on distributed denial of service (ddos) attacks in sdn and cloud computing environments," *IEEE Access*, 2019.

[36] Gramine, "Gramine features," https://gramine.readthedocs.io/en/latest/devel/features.html, [Accessed 05-03-2025].

[37] Gramine, "Encrypted Files in Gramine Gramine documentation," https://gramine.readthedocs.io/en/latest/devel/encfiles.html, [Accessed 06-03-2025].

[38] "sns2ifttt: An AWS Lambda function that acts as an in-between from SNS to IFTTT," https://github.com/ehhva/sns2ifttt, [Accessed 10-03-2025].

[39] C. Cobb, M. Surbatovich, A. Kawakami, M. Sharif, L. Bauer, A. Das, and L. Jia, "How risky are real users' ifttt applets?" in *USENIX Conference on Usable Privacy and Security (SOUPS)*, 2020.

[40] esbuild, "esbuild FAQ," https://esbuild.github.io/faq/, n.d, [Accessed 07-08-2024].

[41] Y. Chen, M. Alhanahnah, A. Sabelfeld, R. Chatterjee, and E. Fernandes, "Practical Data Access Minimization in Trigger-Action Platforms," in *USENIX Security Symposium*, 2022.

[42] IFTTT, "Top Applets of 2024," https://ifttt.com/explore/top-applets-2024, [Accessed 05-03-2025].

[43] Gramine, "Performance tuning and analysis," https://gramine.readthedocs.io/en/latest/performance.html, n.d., [Accessed 07-06-2024].

[44] IFTTT, "Polling trigger," https://help.ifttt.com/hc/en-us/articles/4411016949403-Glossary#Pollingtrigger, [Accessed: 2024-10-16].

[45] I. Zavalyshyn, N. Santos, R. Sadre, and A. Legay, "My house, my rules: A private-by-design smart home platform," in *International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services (MobiQuitous)*, 2021.

[46] A. Oak, A. M. Ahmadian, M. Balliu, and G. Salvaneschi, "Language support for secure software development with enclaves," in *34th IEEE Computer Security Foundations Symposium*, 2021.

[47] T. Hunt, Z. Zhu, Y. Xu, S. Peter, and E. Witchel, "Ryoan: A distributed sandbox for untrusted computation on secret data," in *USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2016.

[48] D. Goltzsche, M. Nieke, T. Knauth, and R. Kapitza, "AccTEE: A webassembly-based two-way sandbox for trusted resource accounting," in *International Middleware Conference (MIDDLEWARE)*, 2019.

[49] OAK, "OAK Project," https://github.com/project-oak/oak, [Accessed: 2024-10-17].

[50] Y.-H. Chiang, H.-C. Hsiao, C.-M. Yu, and T. H.-J. Kim, "On the privacy risks of compromised trigger-action platforms," in *European Symposium on Research in Computer Security (ESORICS)*, 2020.

[51] M. M. Ahmadpanah, D. Hedin, and A. Sabelfeld, "Lazytap: On-demand data minimization for trigger-action applications," in *IEEE Symposium on Security and Privacy (SP)*, 2023.

[52] R. Xu, Q. Zeng, L. Zhu, H. Chi, X. Du, and M. Guizani, "Privacy leakage in smart homes and its mitigation: Ifttt as a case study," *IEEE Access*, 2019.

[53] M. Aghvamipanah, M. Amini, C. Artho, and M. Balliu, "Activity recognition protection for iot trigger-action platforms," in *IEEE European Symposium on Security and Privacy (EuroS&P)*, 2024.

[54] M. S. Melara, D. H. Liu, and M. J. Freedman, "Pyronia: Intra-Process Access Control for IoT Applications," *ArXiv*, 2019.

[55] H. Jin Kang, S. Qin Sim, and D. Lo, "Iotbox: Sandbox mining to prevent interaction threats in iot systems," in *IEEE Conference on Software Testing, Verification and Validation (ICST)*, 2021.

[56] M. Birgersson, C. Artho, and M. Balliu, "Sharing without showing: Secure cloud analytics with trusted execution environments," in *IEEE Secure Development Conference*, 2024.

[57] E. Fernandes, A. Rahmati, J. Jung, and A. Prakash, "Decentralized Action Integrity for Trigger-Action IoT Platforms," in *Network and Distributed Security (NDSS) Symposium*, 2018.

[58] J. Fan, Y. He, B. Tang, Q. Li, and R. Sandhu, "Ruledger: Ensuring execution integrity in trigger-action iot platforms," in *IEEE Conference on Computer Communications (INFOCOM)*, 2021.

[59] M. Surbatovich, J. Aljuraidan, L. Bauer, A. Das, and L. Jia, "Some recipes can do more than spoil your appetite: Analyzing the security and privacy risks of ifttt recipes," *International Conference on World Wide Web*, 2017.

[60] E. Fernandes, J. Paupore, A. Rahmati, D. Simionato, M. Conti, and A. Prakash, "FlowFence: Practical Data Protection for Emerging IoT Application Frameworks," in *USENIX Security Symposium*, 2016.

[61] Ansible, "Ansible," https://www.ansible.com/, 2024, [Accessed: 2024-08-28].

[62] Jenkins, "Jenkins," https://www.jenkins.io/, 2024, [Accessed: 2024-08-28].

Appendix A. Infrastructure as Code

Infrastructure as Code (IaC) is essential in modern IT operations, streamlining the deployment and management of infrastructure and applications. These tools enable developers to automate the software deployment process and minimize the need for manual involvement. Ansible [61] and Jenkins [62] are examples of widely-used IaCs in the software development process. In this paper, we use Ansible as an IaC tool to provide a simple, ready-to-use process for deploying IoT apps on TAPs. Ansible is an open-source IaC software (developed by Red Hat) that provides functionalities for software provisioning, system configuration, and application deployment. Ansible uses the Secure Shell Protocol (SSH) protocol to execute tasks remotely and does not require additional software on the target machine.

Appendix B. Security-relevant Node-RED flows

In the following, we illustrate each use case that we utilized in the security analysis of TAPShield, as detailed in Section 7.1.

Database Operation flow aims to communicate with a running database (DB) using the `node-red-node-mysql` community library and the `Inject` node. The flow begins by connecting to a specific DB. It then proceeds with various operations such as Create, Insert, Delete, and more.

Twitch API flow allows users to use the Twitch API with Node-RED. It first generates the Bearer token for the user using an `Http-Request`. Then, this flow continues by requesting Twitch API endpoints to retrieve information about a specific channel, such as stream state, viewer counts, and follower usernames. One piece of sensitive information in this flow is the Bearer token, which could be leaked to the cloud. Furthermore, modifying the endpoints and responses of the APIs represents another potential attack scenario. In both cases, the cloud attacker would need to alter the application, which is not possible due to the attestation verification enforced by TAPShield. Additionally, the application is encrypted during execution using Intel SGX cryptographic functions (encrypted mounting), hence the cloud is not permitted to read enclave memory.

Python Executor injects the Python script using the `Template` node and then executes it on the target machine using the `WriteFile` and `Exec` nodes. One potential attack involves altering the script to redirect user data to a specific attacker endpoint, which is not allowed due to the attestation process. Secondly, any additional file used by the script is mounted using the encryption, which protects against the leakage of sensitive information from the application to the cloud.

Uploader is an interface that enables the user to upload a file to a specific endpoint. This flow consists of a set of `Template` and `Network` nodes designed to provide the user interface and handle upload requests. One potential scenario in a compromised cloud involves either reading the uploaded file or altering the flow to send the file to an attacker's endpoint.

Calendar Bot flow use `telegrambot` library to create a calendar bot capable of communicating with users via Telegram. In order to deploy this flow in the cloud we use `BotFather` telegram bot which is a manager bot. Given that the flow communicates with the Telegram bot using a key, one potential attack is the leakage of this key which is protected by application encryption. Furthermore, the compromised cloud could manipulate the `Chat-ID` and Bot configuration to redirect the user's conversation to a different, specified bot which causes different expected `Measurement` in the attestation verification step.

Google Sheet Controller enables a user to retrieve Google Sheet records or write `JSON` data to the sheet using `spreadsheet` and `authentication` community nodes. To authenticate with Google, it is necessary to inject a Service Account Key, which is sensitive information. If this key is leaked, it grants access to authorized files through Google APIs. We protect against this attack through encrypted mounting of the application containing the node libraries. Another potential scenario is the modification of Google Sheet data by altering the application, which is prevented by the `Attester` verification.

SMS Message Sender flow aims to provide a service for each user to send a specific text message to a phone number using Vonage service APIs. This service offers a collection of endpoints that allow users to both receive and send text messages. We use a free subscription of this service to evaluate the functionalities of this flow. Exposing the API keys in this instance could result in users being billed for additional expenses, which is mitigated through application encryption. Furthermore, any modification of the text message would cause the verification to fail during the attestation process.

Email Norifier is a Node-RED flow designed to provide notifications whenever a user receives a new email in a specific mailbox. We re-implemented the same flow in Node-RED using `node-red-contrib-email-out` nodes. `node-red-contrib-email-out` uses email credentials to trigger the flow. As a result, the deployed application includes credentials that are sensitive to the user and are decrypted during the key provisioning step. Furthermore, if the cloud is compromised, it could allow attackers to read the user's new emails if the application is executed in an insecure execution environment.

Object Detection flow aims to detect different types of objects using the `node-red-contrib-model-asset-exchange` library, which contains nodes for deep learning microservices from Model Asset Exchange. This library supports a wide variety of application domains in deep learning, such as object detectors, image segmenters, and even audio classifiers. The flow is triggered by an image input and passes the image to the service with appropriate hyperparameters to execute the algorithm. In an insecure execution environment, the cloud could alter hyperparameters, manipulate the final detection results,

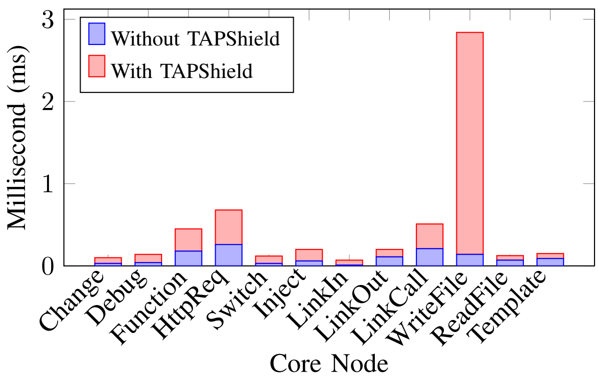

Figure 7. Performance evaluation of core nodes

or even supply a modified image to the algorithm.

Todoist flow is the third flow that we implement in Node-RED. The `node-red-contrib-todoist-api` library in the Node-RED community enables communication with the Todoist application using its REST APIs. We need to pass multiple pieces of data, such as API keys and `task-IDs`, to the flow, and leakage of this data could be dangerous. Furthermore, running the flow in an insecure environment could alter the application's functionality—such as deleting all tasks—which we protect against through attestation.

Appendix C. RQ2: Performance Evaluation

Figure 7 shows the average execution time of each node for Node-RED.

Appendix D. RQ3: Compatibility

TABLE 5. Evaluation on 50 IFTTT Apps in 4 Execution Modes

Environment	Execution Time (ms)
Node.js	0.15
TAPShield	1.32
TAPShield & vm2	45.85
TAPShield & SandTrap (Learning OFF)	36.57

In the following, we present the most dependent-upon Node-RED flows used in TAPShield evaluation. It is important to note that each application is specified with a name and is illustrated with this name in Figures 8 and 9. In addition, Table 6 illustrate the IFTTT filter codes for compatibility evaluation.

MURL: *Monitoring URL* is a web-based application used to monitor different endpoints and their accessibility via a user interface. After running the application, users can access it at `https://ServerAddress:8443/api/home` and begin adding various endpoints. Furthermore, through the application, users can configure different requests and specify the `Request-Header` and `Status-Code`. The application offers a user-friendly interface that enables users to schedule requests according to their needs using `CronJob`. Altogether, the flow developer utilized 206 nodes from 23 distinct node types in Node-RED.

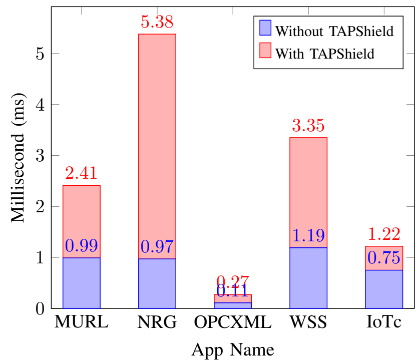

Figure 8. Performance evaluation of most dependent upon Flows

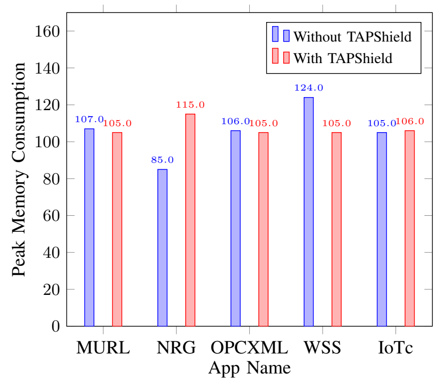

Figure 9. Average memory consumption of Node-RED flows

NRG: *Node-RED Gateway (NRG)* is a Node-RED web server that allows users to serve static and dynamic web pages using Node-RED. NRG employs the Model-View-Controller (MVC) architectural pattern to facilitate the delivery of logic files in the `.NRG` format within a user interface, which includes built-in support for `HTML` and `CSS`. The NRG web server provides support for server error handling, logical action declaration, and server-side scripting, aiming to offer users functionalities similar to those of a traditional web server. NRG contains a total of 65 nodes, including 15 distinct node types. One important aspect of this flow is reading the web server configuration from pre-defined files, which introduces additional overhead when running inside the enclave.

OPCXML: *OPCXML Client* is an application that requests data from an `OPCXML` server. OPC (OLE for Process Control) is a set of standards for connecting industrial automation and control systems. OPCXML is a part of the OPC feature set and is designed for exchanging data between OPC-compliant systems using `XML` (eXtensible Markup Language). The *OPCXML Client* flow supports various data processing actions such as `Browse`, `GetProperties`, `GetStatus`, and `Subscribe`. To implement this flow, the developer used the `SubFlow` feature of Node-RED, which handles different requests using 74 nodes categorized into 15 types. Ultimately, users are able to write the request responses and read them from their own directory.

WSS: *Weather Status Service using MQTT Broker* is a flow implemented in Node-RED that provides a message broker for sharing weather status information. By using Node-RED user interface nodes, the flow offers a dashboard that displays weather status using

TABLE 6. IFTTT APPS FOR COMPATIBILITY EVALUATION

IFTTT Filter Code	Specification	Trigger Service	Action Service
Close Garage Door via Google Assistant	Close MyQ garage door with Google Assistant voice command	Google Assistant V2	MyQ Devices
Close Garage Door via Siri	Close MyQ garage door using iOS Shortcuts	iOS Shortcuts	MyQ Devices
Automatically Arm Blink System When Leaving Home	Arm Blink security system upon exiting a location	Location	Blink
Liked YouTube Songs to Spotify Playlist	Add liked YouTube video songs to a Spotify playlist	YouTube	Spotify
Sync Evernote and Todoist Tasks	Create Todoist tasks from Evernote notes	Trigger	Todoist, Evernote
Track Work Hours with Button Press	Log button presses as work hours in Google Sheets	Do Button	Google Sheets
Quick Note to iOS Reminders	Save a quick note to iOS Reminders with a priority	Do Note	iOS Reminders
Generate a Draft Blog Post	Create a draft Google Doc for blog ideas from Do Note	Do Note	Google Docs
Save NASA Image to iOS Reading List	Save NASA's image of the day to the iOS Reading List	Space	iOS Reading List
Upload Camera Photos to Google Drive	Automatically upload photos taken with Do Camera to Google Drive	Do Camera	Google Drive
RSS to Weebly	Publish a blog post on Weebly from RSS feeds	Feed	Weebly
RSS Feeds to Nimbus Note	Add RSS feed entries as notes in Nimbus	Feed	Nimbus Note
RSS to Trello	Create Trello cards from RSS feed items	Feed	Trello
RSS to Evernote	Create Evernote notes with RSS feed links	Feed	Evernote
RSS to Toodledo	Create Toodledo tasks from RSS feeds	Feed	Toodledo
Rain Tomorrow? Get a Notification	Receive a mobile notification if rain is forecasted	Weather	IfNotifications
Call Ends? Show Caller's Location	Show the caller's location when a call ends	Android Phone	Android Device
NASA Image as Android Wallpaper	Set NASA's daily image as Android wallpaper	Space	Android Device
Save Liked Songs to Spotify Playlist	Add newly liked Spotify songs to a playlist	Spotify	Spotify
Start Roomba Cleaning When Leaving Home	Start an iRobot Roomba when exiting a region	Location	iRobot
Tweet Instagram Photos as Native Twitter Images	Post an Instagram photo as a tweet with an image	Instagram	Twitter
Sync iOS Contacts to Google Spreadsheet	Append new iOS contacts to a Google Sheet	IosContacts	GoogleSheets
Save Tagged Facebook Photos to Dropbox	Save Facebook-tagged photos to a Dropbox folder	Facebook	Dropbox
Backup Tagged Facebook Photos to iOS Photos Album	Save tagged Facebook photos to an iOS Photos album	Facebook	IosPhotos
Track Work Hours in Google Calendar	Log office entry/exit times in Google Calendar	Location	GoogleCalendar
Track Work Hours in Google Drive with Button Press	Log button press events to Google Sheets	DoButton	GoogleSheets
Upload Instagram Photos to Facebook Page Album	Upload new Instagram photos to a Facebook page album	Instagram	FacebookPages
Log Time Spent at Locations in a Spreadsheet	Record time spent at specific locations in Google Sheets	Location	GoogleSheets
Tweet Facebook Status Updates	Post Facebook status updates as tweets	Facebook	Twitter
Call Phone When Arlo Detects Motion	Call phone when Arlo security camera detects motion	Arlo	PhoneCall
Call Phone When Blink Camera Detects Motion	Call phone when Blink security camera detects motion	Blink	PhoneCall
Receive Intrusion Alerts via SMS	Send an SMS alert when intrusion is detected	AnywareServices	Sms
Post New Instagram Photos to WordPress	Create a WordPress post with a new Instagram photo	Instagram	Wordpress
Dictate a Voice Memo and Email MP3 File	Send an email with an MP3 of a dictated voicemail	PhoneCall	Email
Quickly Email Yourself a Note	Send an email with a note written in DoNote	DoNote	Email
Activate LightwaveRF Socket Based on Time	Control a LightwaveRF socket based on the time	TS	LightwaverfPower
Modify RSS Feed Image Size and Post to Facebook	Adjust image size in RSS feed before posting to Facebook	Feed	FacebookPages
Send Slack and Email Notifications for Trello Cards	Notify Slack and Email when a Trello card is added	Trello	Slack, Email
Change Hue/Nanoleaf Light Color Based on Weather	Adjust Hue light colors depending on the weather	Weather	Hue
Create a Blogger Entry from a Reddit Post	Publish top Reddit posts as Blogger entries	Reddit	Blogger
Calculate Event Duration and Create iOS Calendar Entry	Calculate duration of Google Calendar event	GoogleCalendar	IosCalendar
Notify When All-Day Calendar Event is Added	Send a notification when an all-day event is added	GoogleCalendar	IfNotifications
Send Rich Notification for Gmail Messages	Notify about new Gmail messages matching a search query	Gmail	IfNotifications
Send Google Search Results as Notification	Send a Google search result link as a rich notification	DoNote	IfNotifications
Control Gogogate Door Based on Time Range	Close a Gogogate door within a specific time range	TriggerService	Gogogate
Set Random LIFX Color via Button Press	Change LIFX light color randomly on button press	DoButton	Lifx
Adjust LIFX Light Colors as It Gets Darker	Change LIFX light colors based on time of day	DateAndTime	Lifx
Filter Twitter Deals and Send Email Digest	Send daily email digest based on Twitter search results	Twitter	EmailDigest
Send Daily Motivational Quote and Weekly Digest	Notify with a daily quote and send a weekly digest	DateAndTime	IfNotifications, EmailDigest

`OpenWeather` APIs. This process involves utilizing distinct endpoints to communicate with the message broker, which then transfers data to the user interface. Once the process is completed, users can access the dashboard at the `https://ServerAddress:8443/api/ui` endpoint. The `Weather Status` flow comprises 70 nodes, categorized into 12 different types.

IoTc: *IoT Devices Controller* is an application based on the Telegram Bot. Using this flow, users are able to manage their home IoT devices via a Telegram Bot. We configured the bot with predefined commands and defined a set of devices such as `Light` and `Switch`. The flow offers an interactive environment for users to manage each IoT device. As physical connectivity is not directly provided in this flow, commands are passed to the endpoint, and the `Debug` node is used to print them to the user interface. The flow primarily consists of 53 nodes from 18 different types.