applied sciences

MDPI

Article

A Taxonomy for Security Flaws in Event-Based Systems

Youn Kyu Lee [1] **and Dohoon Kim** [2,*]

1 Department of Information Security, Seoul Women's University, Seoul 01797, Korea; younkyul@swu.ac.kr
2 Department of Computer Science, Kyonggi University, Suwon-si 16227, Korea
* Correspondence: karmy01@kgu.ac.kr

Received: 1 September 2020; Accepted: 14 October 2020; Published: 20 October 2020

Abstract: Event-based system (EBS) is prevalent in various systems including mobile cyber physical systems (MCPSs), Internet of Things (IoT) applications, mobile applications, and web applications, because of its particular communication model that uses implicit invocation and concurrency between components. However, an EBS's non-determinism in event processing can introduce inherent security vulnerabilities into the system. Multiple types of attacks can incapacitate and damage a target EBS by exploiting this event-based communication model. To minimize the risk of security threats in EBSs, security efforts are required by determining the types of security flaws in the system, the relationship between the flaws, and feasible techniques for dealing with each flaw. However, existing security flaw taxonomies do not appropriately reflect the security issues that originate from an EBS's characteristics. In this paper, we introduce a new taxonomy that defines and classifies the particular types of inherent security flaws in an EBS, which can serve as a basis for resolving its specific security problems. We also correlate our taxonomy with security attacks that can exploit each flaw and identify existing solutions that can be applied to preventing such attacks. We demonstrate that our taxonomy handles particular aspects of EBSs not covered by existing taxonomies.

Keywords: security taxonomies; event-based systems; mobile cyber physical systems; security flaws

1. Introduction

Event-based systems (EBSs) developed by using message-oriented middleware (MOM) platforms [1] have been widely used in mobile cyber physical systems (MCPSs) as well as a wide range of applications including Internet of Things (IoT) [2–5], financial markets, logistics, and web apps [6], including those that directly interfaced with users (e.g., Android apps [7]). In the case of MCPSs, for example, since they integrate distributed entities including computational, communication, and physical components [8], event-based architecture has been considered as an appropriate mechanism for their implementation [8–11]. MCPSs' inherent heterogeneity and integration of multiple processes make event-based architecture as a relevant approach for their modeling and application [12–15]. Specifically, EBSs are highly scalable, easily evolvable, and have a low coupling that makes them especially suitable for highly heterogeneous distributed systems [16–21].

EBSs' popular attributes are led by their communication model. For example, in EBSs, components (interchangeably referred to as "event-clients" or "event-agents") invoke each other implicitly by publishing event messages (simply referred to as "events") instead of directly calling other components via explicit references. Accordingly, the components may not know the consumers of the events they publish, and may not necessarily know the producers of the events they consume as well. Although this communication mechanism provides several advantages, as its operation is based on non-determinism in event processing, it exposes EBSs to security threats such as event spoofing, interception, and eavesdropping [22–24] (called event attacks). To minimize the risk of such threats on EBSs, security efforts are required.

When working on software security efforts, developers or administrators are required to determine the types of security flaws that exist in the system, the relative importance of each flaw, and the types of techniques that can be employed to handle each flaw. A security flaw taxonomy (an ordered system that indicates the natural relationships of security flaws [25,26]) can provide a basis for developers to make better decisions in securing their target software system. For the past three decades, many such lists and taxonomies of security problems have been studied [25–38]. However, despite the prevalence of EBSs, systematic identification and classification of EBSs' security flaws have not been extensively studied yet. Existing security flaw taxonomies do not adequately reflect the security issues that originate from the EBSs' characteristics or have been found in recent types of EBSs such as Android (Android is a mobile operating system (OS), but it also has been studied as a particular type of EBS because it supports event-based communication model. In this research, we consider Android not only as an OS, but also as a software system encompassing from middleware to applications. We will discuss the details in Section 2.2). Because EBSs have particular attributes that general software systems do not bear (e.g., implicit invocation in event communication), the existing lists or taxonomies are not directly applicable for securing EBSs. Therefore, it is inherently necessary to first systematically identify and classify EBSs' fundamental security flaws to negate any vulnerabilities in the system.

In this paper, we introduce a new taxonomy that classifies the security flaws within EBSs [22,39–47]. Built upon previously identified security flaws present in general software systems [25], our taxonomy classifies particular types of inherent flaws in EBSs, and is distinguished from the existing taxonomies because (1) it clarifies and classifies the inherently present security flaws in EBSs, (2) it covers all types of security flaws in the EBSs domain that have been identified so far, and (3) it considers different types of EBSs configurations (e.g., commercial or open-source MOM platforms). We also correlate our taxonomy with security attacks that can exploit these flaws and existing solutions that are applicable to preventing corresponding attacks. We evaluated our taxonomy in terms of its coverage by comparing it with the existing security flaw taxonomies. Our taxonomy covers all types of security flaws discovered in EBSs so far and even handles additional security flaws not covered by existing taxonomies.

The remainder of paper is structured as follows: Section 2 illustrates the background and definitions, Section 3 describes the methodology that we followed and the resulting taxonomy, Section 4 describes its evaluation, and Section 6 presents the conclusions.

2. Background

In this section, we clarify the underlying concepts and terminology that we will use later to describe our taxonomy in Section 3. We first provide the definitions of key concepts that our taxonomy uses. We then introduce the fundamental mechanism of EBSs and the different types of event attacks.

2.1. Key Concepts

In this paper, our use of the terms "flaw", "vulnerability", and "attack" are based on the terms defined in the existing literature [25,26]. A flaw is a defect of a software system, which can result in a security violation [25]. A vulnerability is caused by at least one flaw and can be exploited by attacks. An attack refers to the techniques that an attacker uses for attempting to detect and exploit a vulnerability. Attack or vulnerability taxonomies might be useful when developers (or administrators or testers) need to clarify the ways their target system can be attacked and the parts of the system that should be protected. However, considering the fact that a flaw is the root cause of security violations and can be masked by another part of the system, its identification is more useful for making a target system robust to security threats. Hence, in this paper, we focus on flaws, rather than attacks or vulnerabilities.

2.2. Event-Based Systems

The EBSs' popular attributes (e.g., scalability, evolvability, and low coupling [16–20]) are fundamentally enabled by their communication model. In EBSs, the components (i.e., the units of computation and data) communicate asynchronously with each other by using messages [48]. A message typically describes one or more observed events. An event is any occurrence that can be observed by a component (e.g., a change of the component's state or a change in the environment of the system) [49]. An event and its corresponding message are often conflated in literature for convenience. In this paper, the term "event" will be used to refer to these concepts broadly. A connector is an architectural element tasked with effecting and regulating interactions among the components [1]. Although there exist several connector types, in this paper, a connector will always refer to an event-based distribution connector [1] that distributes events to associated components. We will use the term "event broker" to refer to this concept broadly.

In EBSs, the components do not have explicit references to each other and are only able to invoke an event broker directly [49]. Consequently, the addition, removal, and updating of components can be achieved relatively easily during runtime [50]. A component can be an event producer or a consumer, or both. Communication between the components is processed via "source" and "sink" [51]—a source is an event interface invoked to publish events by a producer component; a sink is an event interface that an event broker invokes to transfer an event to a consumer component. When a producer publishes an event, the event broker routes the event to the appropriate consumers based on the system configuration, along with the routing and filtering policies [49]. When the event broker transfers an event to a component's sink, the component consumes the event. Each sink declares an event type and only allows the processing of events that match its declared type. In this paper, we will target the following three event types commonly used in today's EBSs [48]: (1) nominal, (2) subject-based and (3) attribute-based. Nominal event types are explicitly declared in a system's programming language and subsequently enforced at compile-time. In subject-based event typing, each event type is defined through a string value that captures an event's name. Similar subject-based event types can be organized into naming hierarchies (e.g., Weather/Country/City). In attribute-based event typing, an event type is defined through a set of attributes, where each attribute is a pair of name and value. Event types can be further defined into more specific event subtypes.

2.3. Event Attacks

Event attacks represent the security problems caused by non-determinism in an EBS's event processing encountered by developers and end-users. Event attacks abuse, incapacitate, and damage the system by exploiting event-based communication. Different types of event attacks have been identified throughout various domains, such as mobile and web apps [22,24,47,52–61]. The research to date has identified the following types of event attacks: *Spoofing* (A1): A malicious component can send an event that spoofs a target component to exploit the target's functionality/data [22]; *Interception* (A2): A malicious component can intercept an event that is supposed to be sent to other components and can send back inappropriate replies to make a target component malfunction or to exploit the target's functionality/data [22,24]; *Eavesdropping* (A3): A malicious component can eavesdrop on an event, which contains sensitive data, and is supposed to be open only for particular components [24,60]; *Confused deputy* (A4): A malicious component can indirectly access a target component, by accessing another component that has access to the target component, to exploit the target's functionality/data [47]; *Collusion* (A5): Two or more malicious components can collude by exchanging events to exploit the functionalities or resources of a target component [47]; *Flooding* (A6): A malicious component can send an overwhelming amount of events that makes a target component malfunction [55]; *Delaying* (A7): A malicious component (or event broker) can intentionally delay a series of event interactions to make a target component malfunction [54]. We have formally defined each type of event attack as listed in Table 1.

Table 1. Types of Event Attacks.

No.	Attack Type	Definition
A1	Spoofing	For $V_1, V_2, M_1 \in C$ where $V_1 \neq V_2 \neq M_1$ and $\exists(V_1 \xrightarrow{e} V_2)$ and (V_2 contains f) and $(M_1 \overset{e}{\Rightarrow} V_2)$: M_1 sent a spoofed e to V_2 to exploit f in V_2
A2	Interception	For $V_1, V_2, M_1 \in C$ where $V_1 \neq V_2 \neq M_1$ and $\exists(V_1 \xrightarrow{e} V_2)$ and (e contains s) and $(V_1 \overset{e}{\Rightarrow} M_1) \wedge \neg(V_1 \overset{e}{\Rightarrow} V_2)$: M_1 intercepted e, which was supposed to be sent to V_2, to obtain s
A3	Eavesdropping	For $V_1, V_2, M_1 \in C$ where $V_1 \neq V_2 \neq M_1$ and $\exists(V_1 \xrightarrow{e} V_2)$ and (e contains s) and $(V_1 \overset{e}{\Rightarrow} M_1) \wedge (V_1 \overset{e}{\Rightarrow} V_2)$: M_1 eavesdropped on e, which was supposed to be open only to V_2, to obtain s
A4	Confused deputy	For $V_1, V_2, M_1 \in C$ where $V_1 \neq V_2 \neq M_1$ and $\nexists(M_1 \xrightarrow{e} V_1)$ and (V_1 contains f) and $(M_1 \overset{e_1}{\Longrightarrow} V_2) \wedge (V_2 \overset{e_2}{\Longrightarrow} V_1)$: M_1 accessed V_1 by accessing V_2, which can access V_1, to exploit f in V_1
A5	Collusion	For $V_1, M_1, M_2 \in C$ where $V_1 \neq M_1 \neq M_2$ and $\nexists(M_1 \xrightarrow{e} V_1)$ and (V_1 contains f) and $(M_1 \overset{e_1}{\Longrightarrow} M_2) \wedge (M_2 \overset{e_2}{\Longrightarrow} V_1)$: M_1 colluded with M_2, which can access V_1, to exploit f in V_1
A6	Flooding	For $V_1, V_2, M_1 \in C$ where $V_1 \neq V_2 \neq M_1$ and $\exists(V_1 \xrightarrow{e} V_2)$ and $(M_1 \overset{e*}{\Longrightarrow} V_2) \wedge \neg(V_1 \overset{e}{\Rightarrow} V_2) \wedge$ (the number of $e*$ is overwhelmingly greater than the average number of e): M_1 sent an overwhelming number of $e*$ to hinder V_1 from accessing V_2
A7	Delaying	For $V_1, V_2, V_3, M_1 \in C$ where $V_1 \neq V_2 \neq V_3 \neq M_1$ and $\exists(V_1 \xrightarrow{e} M_1 \xrightarrow{e_1} V_2 \xrightarrow{e_2} V_3)$ and $(V_1 \overset{e}{\Rightarrow} M_1 \overset{e_1}{\Longrightarrow} V_2 \overset{e_2}{\Longrightarrow} V_3) \wedge$ (the time interval between e and e_1 is overwhelmingly larger than the time interval between e_1 and e_2): M_1 delayed the publication of e_1 to make V_2 and V_3 malfunction

C: a set of components, V: a victim component, M: a malicious component, f: sensitive functionality, s: sensitive information, e: an event, $x \xrightarrow{\alpha} y$: an event communication channel for sending an event α from x to y, $x \overset{\alpha}{\Rightarrow} y$: an event α sent from x to y.

As event attacks are administered in the same manner as ordinary event exchanges and the malicious components disguise themselves as benign, it is difficult to identify and block event attacks. Preventing event attacks becomes more challenging especially when it is not possible to predict which component will compromise the system (e.g., as in the case in Android and J2EE apps). For example, in Android systems, depending on the apps installed according to different users' preferences, the components comprising the system would be different. In such a case, as it is hard to guarantee that all components in the system are benign or safe from security threats, existing techniques that require pre-defined access distribution (e.g., role-based access control [39]) cannot be used to prevent event attacks. Although the Android system was designed to enforce permission-based access control [7], some types of event attacks can bypass the permission checks (i.e., confused deputy and collusion [47,53,61]). Putting a strict limitation on event communication may address some of these security threats, but it can reduce the flexibility of and hamper the benefits of the EBSs. Although developers are required to follow security policies while building a system, they tend to lack attention and make mistakes [62]. Practice has also shown that developers are often completely unaware of potential threats or underestimate the framework's capabilities, thus placing the responsibility on the end-users to protect themselves while using the system [63].

3. Taxonomy

3.1. Literature Review Methodology

To analyze the security flaws in the EBS domain, we inspected the results of 84 literatures published in reputable journals and conferences [22,24,39–47,52,56,60,61,64–132]. We carefully followed the general guidelines for a systematic literature review process [133]. Specifically, we formulated our taxonomy by performing a content analysis over a set of literatures. The literatures were initially collected by using reliable literature search engines, such as IEEE Explore, ACM Digital Library, Springer Link, and Google Scholar. As shown in Table 2, our search query was formed as a conjunction of the domain keywords (i.e., "distributed event-based systems", "event-based systems", "android intent", and "android event") and attribute keywords (i.e., "security vulnerability", "security attack", "security flaw", and "security error"). Specifically, our search query was defined as the following formula: $\forall d \in D \wedge \forall a \in A$, where D is the set of domain keywords and A is the set of attribute keywords as specified in Table 2. Note that, to cover a larger number of literatures, synonyms were considered for the attribute keywords during the search process. For example, regarding "vulnerability", we also considered similar keywords such as "flaw" and "error". Because the scope of search for Android keywords is too large, in order to effectively collect the Android literature dealing with the characteristics of EBSs, we used "android event" and "android intent" as domain keywords. The selected keywords were applied to the search for the literatures' titles, abstracts, and tags. To exclude outdated literatures, we limited the scope of the search to literatures published from 2000 to 2020. Although the majority of the literatures regarding EBSs were almost a decade old, we decided to keep them if they had appeared in top-tier conferences or journals with significant contributions (H5-index ≥ 20 or citation counts ≥ 50). Table 2 shows the number of initially searched literatures (IEEE Explore = 104, ACM Digital Library = 624, Springer Link = 1188, Google Scholar = 3078, Total = 4994) processed by keyword-based search over the aforementioned databases. After the initial searching, because the search engines in each database may have processed our queries differently, we performed a consistent keyword validation on the searched literatures based on the same keywords (1st filtering = 2018). After the first filtering, as not all the searched literatures fit within the scope of this research, we performed a brief review based on the title and abstract of each literature (2nd filtering = 780). Our review criteria included whether they handled security issues in EBSs. After the second filtering, we performed a detailed review on the filtered literatures by inspecting if they fit within the scope of this research. Finally, 84 literatures were selected as the base ingredient for our taxonomy.

Table 2. Number of Collected References during Literature Review Process.

D: Domain Keyword	A: Attribute Keyword	IEEE	ACM	Springer	Google Scholar
distributed event-based systems, event-based systems, android event, android intent	security vulnerability, security attack, security flaw, security error	104	624	1188	3078
Initially Searched		4994			
After 1st Filtering		2018			
After 2nd Filtering		780			
After Final Filtering		84			

3.2. *Taxonomy Construction Methodology*

Although EBSs have particular attributes that general software systems do not bear, they may still inherit security issues from them. Hence, we decided to build a taxonomy upon existing taxonomies that targeted general software systems.

First, we targeted the taxonomies that classify software security flaws. The advantage of this type of taxonomy lies in the convenience of creating a common language for sharing security flaws, allowing an efficient organization of security flaws across information sources, and ultimately identifying strategies to remedy security problems, which is the final goal of this research. For example, depending on the type of flaw, developers can figure out applicable solutions from among the existing ones and also for flaws that lack appropriate solutions. According to the review of security flaw taxonomies [37,38,134], the outdated taxonomies (i.e., before the year 2000) tend to be less elaborate than recently published ones [26,33] or some of them have been adapted to the latest ones [25,27,29,37,38]. Thus, among the selected taxonomies, we filtered out the taxonomies published before the year 2000. The taxonomies that only focused on implementation-level errors were also excluded to consider design-oriented security flaws.

Consequently, from among the remaining set of candidates, "software security flaw taxonomy" by Weber et al. [25] was selected as the starting point to create a taxonomy, because it has been designed to adequately reflect the nature of security issues in an EBS. Weber's taxonomy classifies the flaws based on genesis (i.e., how they were introduced to the system). Specifically, this taxonomy is distinguished from others due to its major division between "intentional" and "inadvertent" flaws, which is pertinent to classifying security flaws in EBS. As an EBS generally provides an extensible infrastructure, unintended external source code can be included in the system, which implies that a developer's intention is an important determinant for classifying an EBS's security flaws. For example, although the Android framework was not originally designed to contain security flaws, if an Android app, intentionally designed as malicious, is installed on the system, the system will contain "intentional" security flaws. We adapted Weber's taxonomy based on 84 selected literatures on security issues in EBSs [39–44,64–77,127] as well as on Android security issues that originated from its event-based communication [22,24,45–47,52,56,60,61,78–126,128–132]. From those publications, we first extracted the security flaws each approach tries to address or introduce as an example. Then we clustered the flaws based on the similarity of ways they can be exploited. Finally, we examined if any of those flaws is related to its counterpart in Weber's taxonomy. The detailed process is as follows:

According to the existing research [40,45,79,87,105,118,121], an EBS may contain malicious code that allows different types of external access, such as a piece of code directed to unsafe URL. These types of flaws belong in the same category as "Trapdoor" in Weber's taxonomy. Prior research has defined and introduced a particular concurrency problem that only exists in EBSs, referred to as event anomalies [81,127,128]. Weber's taxonomy does define "Concurrency" flaws, but only includes time-of-check to time-of-use (TOCTTOU) errors; therefore, we expanded the scope of their characteristics and changed the name of the category to "Inadequate Concurrency" to present a more precise definition. The existing approaches indicated that the components in an EBS may communicate via covert (i.e., non-system-standard) communication channels [47,100,119]. Although some types of "Covert Channels" flaws were defined in Weber's taxonomy, we extended them to include newly identified covert channels such as the battery and vibrator in mobile devices. Authentication issues were also identified in EBSs, in the form of permission grant and authentication in a multi-domain EBS—a particular type of EBS comprising multiple event-brokers from different domains [65,80,86,90,108,118–120,135]. We extended the "Inadequate Authentication" category in Weber's taxonomy to include those authentication-related flaws. From Android apps, new types of resource leaks such as resource leaks via wifi and SQLite database were introduced [40,45,88,104,106,114,115,126,129,132], which can be added to "Resource Leak" in Weber's taxonomy. We changed the name of the category "Inadequate Resource Management" to define the scope more broadly. We also found that the flaws that existing approaches try to resolve fall under "Logic/Time Bomb" in Weber's taxonomy [56,103,115]. The existing EBS research introduced the knowledge of flaws where multiple components collude to exploit the system [45,47,61,88,100,104,106,109,117,131]. Moreover, the majority of security attacks in EBS are basically caused by its extensible event communication channels [22,24,39–41,45,47,52,60,68,70,71,77,94,97,99,130,135]. As Weber's taxonomy does not include them, we extended the definitions of "Conspirator" and "Open Event Channels," respectively. We also added "Unsafe Events" and "Unsafe Event Interface" for including cases where those open event channels are unintentionally introduced to the system [22,24,39,41,68–70,75,77,79,84,85,87,95,102,110,123,130]. Note that, to guarantee the completeness of taxonomy, all the flaws extracted from the existing publications were incorporated in the new taxonomy. However, drawing from the flaws in the Weber's taxonomy, we excluded those that were not introduced in the existing literatures under review to build a taxonomy specialized for EBSs.

3.3. Taxonomy

The security flaw taxonomy for EBSs is shown in Figure 1. As an EBS is a particular type of software system, it incorporates some flaws from general software systems. Note that the boxes highlighted in *red* (F1, F4, F6, F9, F10) indicate the flaws adapted from the existing ones [25] to better reflect the system's event-based characteristics, and the boxes highlighted in *blue* (F2, F5, F7, F8) indicate the flaws we added because they are specifically caused by event-based communication. Finally, the *green* box (F3) indicates a flaw whose definition remains unchanged from the existing one [25]. In particular, the dashed boxes (F2, F5, F6, F7, F8, F10) indicate the flaws that can be exploited by event attacks. It is important to note that every flaw in this taxonomy was validated by existing publications regarding the security of EBS and Android [22,24,39–47,52,56,60,61,64–132] In this taxonomy, a software system is defined as a combined system that comprises both application-level and framework-level elements (i.e., middleware) where an operating system is considered as a sub-component of the system. As the taxonomy considers both the design and implementation-level flaws, we will use "developer" as a term that represents both system designer and programmer. Moreover, a component is defined as an architectural unit that can communicate with other components using system-defined events.

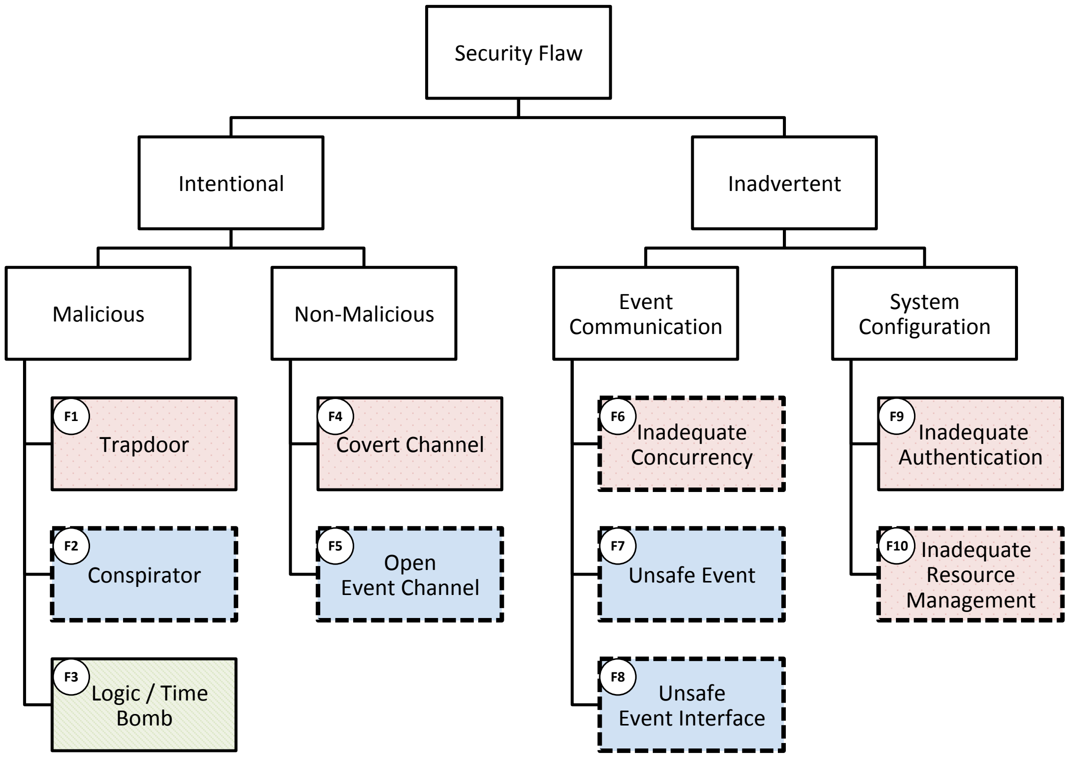

Figure 1. Security Flaw Taxonomy for event-based system (EBS). The *Red* boxes indicate the flaws adapted from the original taxonomy. The *Blue* boxes indicate the newly added flaws. The *Green* box indicates the flaw unchanged from the original taxonomy. The circled labels indicate the assigned number for each flaw.

The goal of this classification is to provide a basis for determining the appropriate security strategies to be used in a particular context. The taxonomy is first classified according to the developer's intention (*Intentional* and *Inadvertent*) because different security strategies can be used to reduce each type of flaw. For example, in a target EBS, if most of the security flaws are unintentionally and inadvertently introduced, exhaustive source code reviews and testing can be utilized to reduce the flaws [26]. However, in case most of the security flaws are intentionally introduced to an EBS, it would be more effective to minimize the proportion of externally-developed source code in the system by restricting the external components access (e.g., restrictive installation of third-party apps on Android system) or by incorporating more trustable message oriented middleware (MOM) platforms.

Intentional flaws are classified as *Malicious* and *Non-Malicious*. The *Malicious* flaws indicate the flaws that were deliberately inserted. If any part of the system was incorporated from an unreliable source, it might intentionally contain the following flaws:

- F1. *Trapdoor* [40,45,47,56,72,79,86,87,89,90,92,93,103,105,109–113,115,117–122,125,126]: Due to an EBS's flexibility and scalability, the system may contain the source code that allows someone to gain illicit access to the system, possibly at both the application and framework level. For example, a user may install an Android app comprising malicious code which directs to undesirable web site. Furthermore, an externally-developed framework for event-based communication may contain malicious code for allowing access to the system.
- F2. *Conspirator* [40,43,45,47,61,88,97,100,103,104,106,107,109,117,119,123,124,131,135,136]: EBSs may comprise components that collude by exchanging events to exploit the system functionalities or access sensitive resources. For example, in an Android system, a component belonging to an app that can access the Internet and a component belonging to an app that can access contact information could collude to send out the contact information over the Internet [47]. Furthermore, a component can help the other component indirectly access sensitive resources, such as photos, contacts, or text messages.

Non-Malicious flaws are the side-effects of features that were deliberately added to the system. These flaws are not recognized by developers in general, but we categorize them as intentional because they were designed into the system by essential system requirements. For example, functional requirements created without considering security requirements can lead to these flaws.

- F4. *Covert Channel* [47,100,119,123]: Two components that are not permitted to communicate via system-standard communication channels (e.g., event-based communication) communicate through the side-effects of the operations authorized for them. Covert channels are classified as intentional and non-malicious because they are not due to bugs in the system's implementation, but due to the system's design. Moreover, they mainly appeared in resource-sharing that are not maliciously designed in the system. This can happen either by means of manipulating storage, or by modulating the time which various operations take to perform. As EBSs can be deployed in various environments, such as mobile devices, the types of covert channels are diversified. For example, in Android systems, shared hardware resources such as audio volume, vibrator, and battery can be used as a communication channel between malicious components [137].
- F5. *Open Event Channel* [22,24,39–41,45–47,52,60,65,68,70,71,75,77,78,84,85,87,94,95,97–99,104,106,107,110,116,117,119,123,130,135,138,139]: This flaw exists when a component intentionally exposes its event communication channel to communicate with other components. Specifically, a component can advertise the types of event it can dispatch or open its event interfaces to share its functionality or data with other components. Although it would make a system more scalable and expandable, there exists a threat where malicious components can exploit the event communication channels in undesirable ways. For example, Android components can dispatch system-defined events to share their functionalities with others, but malicious components can intercept those events and exploit the functionalities [22].

Inadvertent flaws indicate software bugs. Although they can be detected and removed through testing, some flaws may remain undetected and later cause problems during the operation and maintenance stages of the system. Inadvertent flaws are classified based on the parts where the flaws reside. *Event-Based Communication* flaws represent the flaws that can be caused by the design or implementation of a system's event-based communication.

- F6. *Inadequate Concurrency* [22,81,127,128]: A particular form of concurrency flaw exists in EBSs, called event anomalies [81]. In general, EBSs' components randomly process the events that were received simultaneously. Specifically, if two different components simultaneously send the events that can access the same memory location (e.g., a variable containing state or data) of the target component, there is no guarantee that any one of the two events will be processed prior to the other. This flaw may allow spoofed events sent from malicious components to corrupt the victim component's memory location [81].

- F7. *Unsafe Event* [22,24,39–41,45,46,52,60,64,68–71,75,77–79,82,85,94,95,97,98,101,102,107,110,117,130,135,138,139]: This flaw is caused when an event containing sensitive information is insufficiently protected. For example, if a component broadcasts an event containing sensitive information without any particular protection (e.g., encryption), malicious components may intercept or eavesdrop on the event and peek at the sensitive information [22].
- F8. *Unsafe Event Interface* [22,24,39–41,45–47,52,65,68,70,71,75,77,78,82,84,87,94,97,99,104,106,107,110,116,117,119,123,130,135,138,139]: If an event interface of a component has inadequate for filtering for handling received events, the component can be exposed to spoofed events. In case a component contains sensitive functionalities that can be triggered in response to receiving events through the unsafe interface, a malicious component can inject spoofed events to the exposed event interface thereby causing the target component to malfunction or operate in undesirable ways [22].

System Configuration flaws are the ones that can be caused by a system's defective configurations or deployments.

- F9. *Inadequate Authentication* [65,80,86,90,108,118–120]: Because of a low coupling between components in EBSs, this flaw exists when a system does not completely authenticate each component (e.g., checking if each component has sufficient permissions to send or receive events). This may allow malicious components to exploit event interactions in the system (e.g., intercepting or corrupting events). Moreover, in a multi-domain EBS, as the system may comprise multiple event brokers from different domains, the identification and authentication of components may not be uniform across the event broker networks [135], which may allow unsafe access between components.
- F10. *Inadequate Resource Management* [39–41,43,45,56,64,69,72,77,88,101,103,104,106,108,114,115,124,126,129,132,135]: To achieve scalability, EBSs can be deployed on distributed clusters of heterogeneous nodes, which causes complex resource management. This flaw is caused when a system allocates resources to a component and releases them in an untimely manner. For example, if resource allocation is not appropriately designed, a malicious component can monopolize the system resources, which can result in denial of service. Furthermore, inadequate dynamic allocation may lead to convert channels where malicious components can communicate with each other [140].

The remaining flaw in *green* box indicates a flaw inherited from Weber's taxonomy [25]: *Logic/Time Bomb* [56,103,115] flaw indicates a piece of source code designed to disrupt the system when certain conditions are satisfied.

3.4. Relationship between Security Flaws and Event Attacks

The identified security flaws in EBSs can be exploited by different types of attacks including event attacks. To effectively counter each type of event attack, we identified the relationship between the flaws and the event attacks. Then we examined existing solutions that have been proposed to protect the flaws from event attacks. In this section, we demonstrate the relationship between the flaws and event attacks, and assess existing solutions for resolving those attacks.

As discussed in Section 2.3, event attacks represent the security problems faced by developers or end-users due to an EBS's non-determinism in event processing. Recall the seven types of event attacks: *Spoofing* (A1), *Interception* (A2), *Eavesdropping* (A3), *Confused deputy* (A4), *Collusion* (A5), *Flooding* (A6), and *Delaying* (A7).

Each security flaw in an EBS can be exploited by different types of event attacks as depicted in Table 3. To protect each type of security flaw from event attacks, various solutions have been studied across different EBS platforms (e.g., OASIS [77] and Android [141]). Table 3 also presents the representative solutions that prevent event attacks from exploiting each type of security flaw.

Table 3. EBS Security Flaws, Event Attacks, and Existing Solutions.

No.	Security Flaw in EBS	Event Attack	Existing Solution
F1	Trapdoor	-	-
F2	Conspirator	A5	- Detection of information leaks [46,60,142] - Detection and control of colluding apps [47]
F3	Logic/Time Bomb	-	-
F4	Covert Channel	-	-
F5	Open Event Channel	A1-7	- Encryption of events [41] - Policy enforcement [46,47,71,143]
F6	Inadequate Concurrency	A1	- Detection of event anomalies [68]
F7	Unsafe Event	A2,3,7	- Role-based access control [39,135] - Encryption of events [41] - Detection of vulnerable components [22,45,46,142] - Policy enforcement [46,47,71,143]
F8	Unsafe Event Interface	A1,4,6	- Role-based access control [39,135] - Detection of vulnerable components [22,45,46,142] - Policy enforcement [46,47,71,143]
F9	Inadequate Authentication	A1-7	- Security policy validation [39,144]
F10	Inadequate Resource Management	A6-7	- Analysis of runtime events and resources [145,146]

As indicated in Table 3, neither security flaw F1 nor F4 are the targets of event attacks. They can be resolved by general security solutions such as a signature-based detection [147–149] or identification of covert channels [47]. Flaw F2 can be exploited by the attack A5, but the threat can be minimized by detecting sensitive information flows between the components [46,60,142] or controlling unsafe event communication between components [47,53]. Flaw F5 can be exploited by multiple types of event attacks (A1-7). Existing research has tried to minimize the threat using encryption of events, but it requires safe key distribution between the components and additional resources that may become a burden for an environment with limited resources (e.g., mobile devices) [41]. While enforcement of security policies [46,47,71,143] has also been proposed, a coarse-grained policy may fail to prevent event attacks. For flaw F6, which is vulnerable to the attack A1, a static analysis for event anomalies detection [81,127,128] can help developers identify and fix the flaw. Flaw F7 can be a target for the attacks A2, A3, and A7. Although role-based access control and encryption of events [39,41,135] may prevent the attacks, those techniques require certain assumptions about the components engaged in event-based interactions, namely, they assume that "benign" components will be known. In other words, these approaches cannot properly deal with event-related security threats when the types of components are not clearly delineated and a malicious component can behave as a legitimate component. Though existing research has focused on the detection of attacks A2 and A3 in Android apps [22,45,46,142], they either target limited types of attacks or do not provide actual prevention mechanisms. Flaw F8, which is vulnerable to the attacks A1, A4, and A6, can be resolved by the same solutions that are applicable for flaw F7. Flaw F9 is exposed to all types of event attacks, because the possibility of a malicious component's existence in a system can be increased if the system's authentication mechanism is not well-defined. This threat can be minimized by validating a system's security policies [39,144]. Flaw F10, which is vulnerable to the attacks A6 and A7, can be resolved by analyzing and monitoring a system's runtime event interactions or resource usages [145,146].

Overall, existing solutions belong to prevention- or detection-type and each type has its limitations. As the prevention-type solutions are based on the assumption that the types of components are clearly delineated, they can be coarse-grained in case it is unclear how to pinpoint the benign components. Although detection-type solutions provide relatively finer-grained results for identifying the flaws vulnerable to event attacks, they suffer from inaccuracy and scalability issues in their analysis. To further secure EBSs, advanced approaches that combine detecting flaws and preventing attacks are required.

4. Evaluation

To validate our taxonomy in terms of coverage, two different types of evaluation were required: (1) completeness: if it covers all types of security flaws in EBSs; and (2) originality: if it handles particular types of security flaws not covered by existing listings or taxonomies.

Regarding the completeness of our taxonomy, as mentioned in Section 3.2, all types of flaws extracted from existing publications were incorporated in our taxonomy. We carefully collected 80 existing publications dealing with security issues in EBSs as well as Android security issues that originated from its event-based communication feature. We then derived different types of security flaws from those literature and classified them, which guarantees that our taxonomy covers all types of security flaws identified in the EBS domain so far.

To evaluate the originality of our taxonomy, we performed an analytic comparison with existing listings and taxonomies for security flaws. Among a number of studies for classifying security issues, we targeted the most cited or recently published taxonomies. To the best of our knowledge, four existing works share our taxonomy's goal of classifying security issues—Weber's [25], OWASP [36], Tsipneuk's [29], and Linares-Vásquez's [35]. The first three taxonomies mainly target general software systems and the last one targets the Android system. Considering the fact that Android is widely used and is a particular type of EBS, we included Linares-Vásquez's taxonomy in this evaluation. Although the selected taxonomies target different types of security issues (i.e., risks, errors, and vulnerabilities), they also serve the same purpose as our taxonomy in that they classify the cause of the security violations. We analyzed if each type of security issue in the selected taxonomies can be mapped to any flaw type in our taxonomy in terms of its definition. If the definitions of any two types were identical, we classified them as "*completely mapped*," and if they were partially matched in broad terms, then as"*partially mapped*." As each taxonomy has different levels in its classification, we correlated the security issues regardless of the levels of classification.

As mentioned in Section 3.2, out of 16 flaws in Weber's taxonomy [25], we adapted five in terms of their definition and added four related to event-based communication. We excluded ten flaws that mainly focused on implementation-level security issues in general software systems (e.g., aliasing and error handling).

Compared with the Open Web Application Security Project (OWASP) Top Ten 2017 [36], which is a list of the 10 most critical web application security risks, three risk types can be mapped to the flaws in our taxonomy (see Table 4). Specifically, "*Injection*" in the OWASP list can be partially mapped to the flaws F1 and F8 in our taxonomy. It represents an exploitation of a victim to perform unintended behaviors, which can be implemented via flaws F1 and F8. In a broad sense, "*Sensitive data exposure*" in the OWASP list can be partially mapped to the flaw F7, because an unsafe event may expose sensitive data. To be more exact, however, the flaws F7 and F8 are more specific to event-based communication. The remaining seven types of risks in the OWASP list such as "*Cross site scripting*" and "*Insecure deserialization*" are more focused on the inherent characteristics of web applications.

Table 4. Correlation with Existing Security Flaw Taxonomies.

No.	Security Flaw in EBS	Weber's [25]	OWASP [36]	Tsipenyuk's [29]	Linares-Vásquez's [35]
F1	Trap door	○	○	○	○
F2	Conspirator				
F3	Logic/Time Bomb	●			
F4	Covert Channel	○			
F5	Open Event Channel				
F6	Inadequate Concurrency	○			
F7	Unsafe Event		○		○
F8	Unsafe Event Interface		○	○	○
F9	Inadequate Authentication	○			
F10	Inadequate Resource Management	○		○	●

○ : partially mapped, ● : completely mapped

Tsipenyuk's taxonomy [29] handles implementation-level errors that affect a system's security. It classifies seven main categories and 76 underlying errors. Among those errors, three types can be mapped to the flaws in our taxonomy (see Table 4). Specifically, both "*Command injection*" and "*Process control*" can be partially mapped to the flaws F1 and F8 in our taxonomy. They also represent the exploitation of a victim to perform unintended behaviors, which can be implemented via flaws F1 and F8. "*Unreleased resource*" can be partially mapped to the flaw F10 in our taxonomy. It represents a system's failure to release system resources, which can be caused by inadequate resource allocation. However, none of these error types consider the inherent characteristics of EBSs, such as event-based communication. The remaining 73 types of errors in Tsipenyuk's taxonomy do not correlate with the flaws in our taxonomy.

Linares-Vásquez's taxonomy [35] targets security vulnerabilities in Android, and classifies 15 main categories with 126 underlying vulnerabilities. Similar to the aforementioned taxonomies, both "*Code injection*" and "*Command injection*" in Linares-Vásquez's taxonomy can be partially mapped to the flaws F1 and F8 in our taxonomy. "*Resource management errors*" can be completely mapped to our flaw F10 in terms of its definition. Although "*Race condition*" in Linares-Vásquez's can be partially mapped to flaw F6, it does not consider event anomalies [81]. "*Missing encryption of sensitive data*" and "*Insufficient verification of data authenticity*" can be partially mapped to flaw F7 to consider an event containing sensitive information without any particular protection. The remaining 120 types of vulnerabilities in Linares-Vásquez's taxonomy are more focused on Android-specific security issues.

Overall, although existing taxonomies for security issues handle some of the flaws in our taxonomy, most of them are partially matched. Our taxonomy covers additional security flaws related to the inherent characteristics of EBSs, which are not covered by existing listings or taxonomies. However, it is important to note that existing taxonomies cover the flaws related to general software systems that are not the focus on our taxonomy.

5. Discussion

In this paper, we analyzed security flaw patterns and trends in the existing literature, and underlined challenges that will shape the focus of future research. Our taxonomy can help engineers assessing security problems in EBSs they built. A finer-grained classification of the most common flaws or attacks is useful because system administrators need to anticipate what they will experience in their system. It also provides a baseline for collecting and organizing security-related data, and consequently the information can help engineers strengthen their EBSs. Furthermore, our taxonomy will be useful for security practitioners to organize the problem space. Security problems are caused by an unexpected combination of flaws in general. In these cases, finer-grained distinctions between security flaws can help define a specific problem space. Our taxonomy will be useful for researchers to develop and evaluate potential research directions. Despite significant research efforts to mitigate the security threats in EBSs, solutions targeting these types of systems still lack. We believe that the results of our review (see Section 3.4) will help initiate the required research in this area.

In this research, we carefully followed the general guidelines for a systematic literature review (SLR) process in order to minimize the threats to validity. Nevertheless, there exist inherent threats that require further discussion. Our SLR process includes the utilization of search engine and keyword construction. To maximize the completeness of our taxonomy—whether all of the appropriate literature was included—, we adopted multiple search engines and employed an iterative approach for keyword construction. Furthermore, our SLR process inevitably relies on the interpretation of individual reviewers. To address any resulting bias, we additionally conducted the crosschecking of the literatures, such that no paper reviewed by a single reviewer. Although new variations of security flaws in EBSs can be encountered, to mitigate this threat, our taxonomy has adapted existing classification method which has proven to be rich enough to adequately classify the characteristics of security flaws. This implies that our taxonomy can be adapted to counter new types of security flaws in EBSs.

6. Conclusions

Event-based systems (EBSs) have become popular in mobile cyber physical systems, IoT applications, mobile applications, and web applications because of their inherent advantages. However, their reliance on non-determinism in event processing can be exploited by different types of attacks (e.g., event attacks). In the light of current interest in the security threats within EBSs, we developed a novel security flaw taxonomy for EBS. Each flaw is categorized based on the common factors present among flaws, enabling a systematic approach to resolving the security problems in an EBS. We showed the correlations between each flaw and different types of attacks as well as between each flaw and the applicable existing solutions for preventing the corresponding attacks. We also demonstrated that our taxonomy covers all types of security flaws identified in EBSs so far and even handles additional security flaws not covered by existing taxonomies.

Our taxonomy will help developers determine the types of security flaws existing in their target system and decide the appropriate techniques suitable to resolve each one. In addition, our taxonomy will shed light on potential research directions for securing EBSs.

Author Contributions: Y.K.L. was the main researcher who initiated and organized research reported in the paper, and all authors including Y.K.L. and D.K. were responsible for analyzing the literatures and writing the paper. D.K. performed detailed writing-review and validation. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by the National Research Foundation of Korea(NRF) grant funded by the Korea government(MSIT) (No. 2020R1F1A1068774) and a research grant from Seoul Women's University (2020-0451).

Acknowledgments: Dohoon Kim is the corresponding author of this paper.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Taylor, R.; Medvidovic, N.; Dashofy, E. *Software Architecture: Foundations, Theory, and Practice*; John Wiley & Sons: Hoboken, NJ, USA, 2009.
2. Bonte, P.; Ongenae, F.; De Turck, F. Subset Reasoning for Event-Based Systems. *IEEE Access* **2019**, *7*, 107533–107549. [CrossRef]
3. Kędzierski, D.; Matuszak, P. *IoT System for Sensors Data Acquisition and Controlling Devices via Web*; Technical Report; Wrocław University of Science and Technology: Wrocław, Poland, 2019.
4. Ma, M.; Wang, P. Efficient Event Inference and Context-Awareness in Internet of Things Edge Systems. *IEEE Trans. Big Data* **2019**. [CrossRef]
5. Koldehofe, B. *Principles of Building Scalable and Robust Event-Based Systems*; Technical Report; Technische Universität Darmstadt: Darmstadt, Germany, 2019.

6. Java Message Service (JMS). 2016. Available online: http://www.oracle.com/technetwork/java/jms/index.html (accessed on 16 September 2016).
7. Android Open Source Project. 2015. Available online: https://source.android.com (accessed on 16 September 2016).
8. Lee, E.A. Cyber Physical Systems: Design Challenges. In Proceedings of the IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC), Orlando, FL, USA, 5–7 May 2008; pp. 363–369.
9. Li, X.; Wang, Y.; Zhou, X. An event-based architecture for cyber physical systems. In Proceedings of the IEEE International Conference on Information Science and Technology (ICITS), Shenzhen, China, 26–28 April 2014; pp. 96–99.
10. Ochoa, S.F.; Fortino, G.; Di Fatta, G. Cyber-Physical Systems, Internet of Things and Big Data. *Future Gener. Comput. Syst.* **2017**, *5*, 40. [CrossRef]
11. Pu, C. A World of Opportunities: CPS, IOT, and Beyond. In Proceedings of the IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing, New York, NY, USA, 11–14 July 2011; pp. 229–230.
12. Tan, Y.; Vuran, M.C.; Goddard, S.; Yu, Y.; Song, M.; Ren, S. A Concept Lattice-Based Event Model for Cyber-Physical Systems. In Proceedings of the ACM/IEEE International Conference on Cyber-Physical Systems (ICCPS), Stockholm, Sweden, 12–15 April 2010; pp. 50–60.
13. Ollesch, J.; Hesenius, M.; Gruhn, V. Engineering Events in CPS-Experiences and Lessons Learned. In Proceedings of the 2017 IEEE/ACM 3rd International Workshop on Software Engineering for Smart Cyber-Physical Systems (SEsCPS), Buenos Aires, Argentina, 21 May 2017; pp. 3–9.
14. Rosenthal, F.; Jung, M.; Zitterbart, M.; Hanebeck, U.D. CoCPN–Towards Flexible and Adaptive Cyber-Physical Systems Through Cooperation. In Proceedings of the 2019 16th IEEE Annual Consumer Communications & Networking Conference (CCNC), Las Vegas, NV, USA, 11–14 January 2019; pp. 1–6.
15. Ollesch, J.; Hesenius, M.; Gruhn, V.; Alias, C. The Requirements Engineering Perspective on Events in Cyber-Physical Systems: Poster. In Proceedings of the 11th ACM International Conference on Distributed and Event-based Systems, Barcelona, Spain, 19–23 June 2017; pp. 349–350.
16. Eugster, P.; Felber, P.A.; Guerraoui, R.; Kermarrec, A.M. The Many Faces of Publish/Subscribe. *ACM Comput. Surv.* **2003**, *35*, 114–131. [CrossRef]
17. Taylor, R.N.; Medvidovic, N.; Anderson, K.M.; Whitehead, E.J.; Robbins, J.E.; Nies, K.A.; Oreizy, P.; Dubrow, D.L. A Component- and Message-Based Architectural Style for GUI Software. *IEEE Trans. Softw. Eng.* **1996**, *22*, 390–406. [CrossRef]
18. Cugola, G.; Di Nitto, E.; Fuggetta, A. The JEDI Event-Based Infrastructure and Its Application to the Development of the OPSS WFMS. *ACM Comput. Surv.* **2001**, *27*, 827–850. [CrossRef]
19. Correira, J. *Market Share: AIM and Portal Software, Worldwide, 2005*; Gartner Market Research Report; Gartner Research: Stamford, CT, USA, 2006.
20. Biscotti, F.; Raina, A. *Market Share Analysis: Application Infrastructure and Middleware Software, Worldwide, 2011*; Gartner Market Research Report; Gartner Research: Stamford, CT, USA, 2012.
21. Mohamed, S.; Forshaw, M.; Thomas, N.; Dinn, A. Performance and Dependability Evaluation of Distributed Event-based Systems: A Dynamic Code-injection Approach. In Proceedings of the 8th ACM/SPEC on International Conference on Performance Engineering (ICPE), L'Aquila, Italy, 22–27 April 2017; pp. 349–352
22. Chin, E.; Felt, A.P.; Greenwood, K.; Wagner, D. Analyzing Inter-Application Communication in Android. In Proceedings of the 9th International Conference on Mobile Systems, Applications, and Services (MobiSys), Washington, DC, USA, 28 June–1 July 2011; pp. 239–252.
23. Lee, Y.K. Detecting Inter-Component Vulnerabilities in Event-based Systems. *Int. J. Adv. Comput. Sci. Appl.* **2019**, *10*, 22–28. [CrossRef]
24. Salvia, R.; Cortesi, A.; Ferrara, P.; Spoto, F. Intents Analysis of Android Apps for Confidentiality Leakage Detection. In *Advanced Computing and Systems for Security*; Springer: Berlin/Heidelberg, Germany, 2020; pp. 43–65.
25. Weber, S.; Karger, P.A.; Paradkar, A. A Software Flaw Taxonomy: Aiming Tools at Security. *SIGSOFT Softw. Eng. Notes* **2005**, *30*, 1–7. [CrossRef]
26. Landwehr, C.E.; Bull, A.R.; McDermott, J.P.; Choi, W.S. A Taxonomy of Computer Program Security Flaws. *ACM Comput. Surv.* **1994**, *26*, 211–254. [CrossRef]

27. Sufatrio; Tan, D.J.J.; Chua, T.W.; Thing, V.L.L. Securing Android: A Survey, Taxonomy, and Challenges. *ACM Comput. Surv.* **2015**, *47*, 58:1–58:45. [CrossRef]
28. Simmons, C.; Ellis, C.; Shiva, S.; Dasgupta, D.; Wu, Q. *AVOIDIT: A Cyber Attack Taxonomy*; Technical Report; University of Memphis: Memphis, TN, USA, 2009.
29. Tsipenyuk, K.; Chess, B.; McGraw, G. Seven Pernicious Kingdoms: A Taxonomy of Software Security Errors. *IEEE Secur. Priv.* **2005**, *3*, 81–84. [CrossRef]
30. Jiwnani, K.; Zelkowitz, M. Susceptibility Matrix: A New Aid to Software Auditing. *IEEE Secur. Priv.* **2004**, *2*, 16–21. [CrossRef]
31. Igure, V.M.; Williams, R.D. Taxonomies of Attacks and Vulnerabilities in Computer Systems. *IEEE Commun. Surv. Tutor.* **2008**, *10*, 6–19. [CrossRef]
32. Joshi, C.; Singh, U.K. ADMIT- A Five Dimensional Approach towards Standardization of Network and Computer Attack Taxonomies. *Int. J. Comput. Appl.* **2014**, *100*, 30–36. [CrossRef]
33. Aslam, T. A taxonomy of Security Faults in the Unix Operating System. Master's Thesis, Purdue University, West Lafayette, IN, USA, 1995.
34. Piessens, F. A Taxonomy of Causes of Software Vulnerabilities in Internet Software. In Proceedings of the Supplementary 13th International Symposium on Software Reliability Engineering (ISSRE), Annapolis, MD, USA, 12–15 November 2002; pp. 47–52.
35. Linares-Vásquez, M.; Bavota, G.; Escobar-Velásquez, C. An Empirical Study on Android-related Vulnerabilities. In Proceedings of the 14th International Conference on Mining Software Repositories (MSR), Buenos Aires, Argentina, 20–21 May 2017; pp. 2–13. [CrossRef]
36. Category:OWASP Top Ten 2017 Project. 2017. Available online: https://owasp.org/www-project-top-ten/ (accessed on 9 October 2017).
37. Joshi, C.; Singh, U. A Review on Taxonomies of Attacks and Vulnerability in Computer and Network System. *Int. J.* **2015**, *5*, 742–747.
38. Hui, Z.; Huang, S.; Ren, Z.; Yao, Y. Review of Software Security Defects Taxonomy. In Proceedings of the 5th International Conference on Rough Set and Knowledge Technology (RSKT), Beijing, China, 15–17 October 2010; pp. 310–321.
39. Belokosztolszki, A.; Eyers, D.M.; Pietzuch, P.R.; Bacon, J.; Moody, K. Role-Based Access Control for Publish/Subscribe Middleware Architectures. In Proceedings of the 2nd International Workshop on Distributed Event-Based Systems (DEBS), San Diego, CA, USA, 8 June 2003; pp. 1–8.
40. Shand, B.; Pietzuch, P.; Papagiannis, I.; Moody, K.; Migliavacca, M.; Eyers, D.; Bacon, J. Security Policy and Information Sharing in Distributed Event-Based Systems. In *Reasoning in Event-Based Distributed Systems*; Springer: Berlin/Heidelberg, Germany, 2011; pp. 151–172.
41. Pesonen, L.I.W.; Eyers, D.M.; Bacon, J. Encryption-Enforced Access Control in Dynamic Multi-Domain Publish/Subscribe Networks. In Proceedings of the Inaugural International Conference on Distributed Event-based Systems (DEBS), Toronto, ON, Canada, 20–22 June 2007; pp. 104–115.
42. Srivatsa, M.; Liu, L.; Iyengar, A. EventGuard: A System Architecture for Securing Publish-Subscribe Networks. *ACM Trans. Comput. Syst. (TOCS)* **2011**, *29*, 10:1–10:40. [CrossRef]
43. Fiege, L.; Mezini, M.; Mühl, G.; Buchmann, A.P. Engineering Event-Based Systems with Scopes. In Proceedings of the 16th European Conference on Object-Oriented Programming (ECOOP), Malaga, Spain, 10–14 June 2002; pp. 309–333.
44. Templeton, S.J.; Levitt, K.E. Detecting Spoofed Packets. In Proceedings of the DARPA Information Survivability Conference and Exposition, Washington, DC, USA, 22–24 April 2003; Volume 1, pp. 164–175.
45. Bagheri, H.; Sadeghi, A.; Garcia, J.; Malek, S. COVERT: Compositional Analysis of Android Inter-App Permission Leakage. *IEEE Trans. Softw. Eng.* **2015**, *41*, 866–886. [CrossRef]
46. Bagheri, H.; Sadeghi, A.; Jabbarvand, R.; Malek, S. *Automated Dynamic Enforcement of Synthesized Security Policies in Android*; Technical Report GMU-CS-TR-2015-5; George Mason University: Fairfax, VA, USA, 2015 .
47. Bugiel, S.; Davi, L.; Dmitrienko, R.; Fischer, T. Towards Taming Privilege-Escalation Attacks on Android. In Proceedings of the 19th Annual Network & Distributed System Security Symposium (NDSS), San Diego, CA, USA, 5–8 February 2012.
48. Mühl, G.; Fiege, L.; Pietzuch, P. *Distributed Event-Based Systems*; Springer Inc.: New York, NY, USA, 2006.

49. Popescu, D.; Garcia, J.; Bierhoff, K.; Medvidovic, N. Impact Analysis for Distributed Event-based Systems. In Proceedings of the 6th ACM International Conference on Distributed Event-Based Systems (DEBS), Berlin, Germany, 16–20 July 2012; pp. 241–251.
50. Oreizy, P.; Medvidovic, N.; Taylor, R.N. Architecture-based Runtime Software Evolution. In Proceedings of the 20th International Conference on Software Engineering (ICSE), Kyoto, Japan, 19–25 April 1998; pp. 177–186.
51. Garcia, J.; Popescu, D.; Safi, G.; Halfond, W.G.J.; Medvidovic, N. Identifying Message Flow in Distributed Event-Based Systems. In Proceedings of the 9th Joint Meeting on Foundations of Software Engineering (ESEC/FSE), Saint Petersburg, Russia, 18–26 August 2013; pp. 367–377.
52. Bagheri, H.; Sadeghi, A.; Jabbarvand, R.; Malek, S. Practical, Formal Synthesis and Automatic Enforcement of Security Policies for Android. In Proceedings of the 46th IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), Toulouse, France, 28 June–1 July 2016; pp. 514–525.
53. Bugiel, S.; Davi, L.; Dmitrienko, A.; Fischer, T.; Sadeghi, A.R. *XManDroid: A New Android Evolution to Mitigate Privilege Escalation Attacks*; Technical Report TR-2011-04; Technische Universität Darmstadt: Darmstadt, Germany, 2011 .
54. Zhang, R.; Cai, K.; Wonham, W.M. Delay-robustness in distributed control of timed discrete-event systems based on supervisor localization. In Proceedings of the 53rd IEEE Conference on Decision and Control, Los Angeles, CA, USA, 15–17 December 2014; pp. 6719–6724.
55. Dolk, V.S.; Tesi, P.; Persis, C.D.; Heemels, W.P.M.H. Event-Triggered Control Systems Under Denial-of-Service Attacks. *IEEE Trans. Control Netw. Syst.* **2017**, *4*, 93–105. [CrossRef]
56. Rasthofer, S.; Arzt, S.; Lovat, E.; Bodden, E. DroidForce: Enforcing Complex, Data-centric, System-wide Policies in Android. In Proceedings of the 9th International Conference on Availability, Reliability, and Security (ARES), Fribourg, Switzerland, 8–12 September 2014; pp. 40–49.
57. Tripp, O.; Pistoia, M.; Cousot, P.; Cousot, R.; Guarnieri, S. ANDROMEDA: Accurate and Scalable Security Analysis of Web Applications. In Proceedings of the 16th International Conference on Fundamental Approaches to Software Engineering (FASE), Rome, Italy, 16–24 March 2013; pp. 210–225. [CrossRef]
58. Sridharan, M.; Artzi, S.; Pistoia, M.; Guarnieri, S.; Tripp, O.; Berg, R. F4F: Taint Analysis of Framework-based Web Applications. In Proceedings of the 2011 ACM International Conference on Object Oriented Programming Systems Languages and Applications (OOPSLA), Portland, OR, USA, 22–27 October 2011; pp. 1053–1068. [CrossRef]
59. Pietzuch, P. Building Secure Event Processing Applications. In Proceedings of the First International Workshop on Algorithms and Models for Distributed Event Processing (AlMoDEP), Rome, Italy, 19 September 2011; p. 11. [CrossRef]
60. Li, L.; Bartel, A.; Bissyandé, T.F.; Klein, J.; Le Traon, Y.; Arzt, S.; Rasthofer, S.; Bodden, E.; Octeau, D.; Mcdaniel, P. IccTA: Detecting Inter-Component Privacy Leaks in Android App. In Proceedings of the 37th International Conference on Software Engineering (ICSE), Florence, Italy, 16–24 May 2015; pp. 280–291.
61. Demissie, B.F.; Ceccato, M.; Shar, L.K. Security analysis of permission re-delegation vulnerabilities in Android apps. In *Empirical Software Engineering*; Springer: Berlin/Heidelberg, Germany, 2020; pp. 1–53.
62. Developer Error: The Most Dangerous Programming Mistakes | InfoWorld. 2016. Available online: http://www.infoworld.com/article/2622611/application-security/developer-error--the-most-dangerous-programming-mistakes.html (accessed on 25 September 2016).
63. End User Device (EUD) Security Guidance. 2018. Available online: https://www.ncsc.gov.uk/collection/end-user-device-security/ (accessed on 17 November 2018).
64. Petroni, F.; Querzoni, L.; Beraldi, R.; Paolucci, M. Exploiting User Feedback for Online Filtering in Event-based Systems. In Proceedings of the 31st Annual ACM Symposium on Applied Computing (SAC), Pisa, Italy, 4–8 April 2016; pp. 2021–2026. [CrossRef]
65. Aniello, L.; Baldoni, R.; Ciccotelli, C.; Di Luna, G.A.; Frontali, F.; Querzoni, L. The Overlay Scan Attack: Inferring Topologies of Distributed Pub/Sub Systems Through Broker Saturation. In Proceedings of the 8th ACM International Conference on Distributed Event-Based Systems (DEBS), Mumbai, India, 26–29 May 2014; pp. 107–117. [CrossRef]
66. Tariq, M.A.; Koldehofe, B.; Altaweel, A.; Rothermel, K. Providing Basic Security Mechanisms in Broker-less Publish/Subscribe Systems. In Proceedings of the Fourth ACM International Conference on Distributed Event-Based Systems (DEBS), Cambridge, UK, 12–15 July 2010; pp. 38–49. [CrossRef]

67. Srivatsa, M.; Liu, L. Secure Event Dissemination in Publish-Subscribe Networks. In Proceedings of the 27th International Conference on Distributed Computing Systems (ICDCS), Toronto, ON, Canada, 25–29 June 2007; p. 22. [CrossRef]
68. Raiciu, C.; Rosenblum, D.S. Enabling Confidentiality in Content-Based Publish/Subscribe Infrastructures. In Proceedings of the Securecomm and Workshops (SecureComm), Baltimore, MD, USA, 28 August–1 September 2006; pp. 1–11.
69. Nabeel, M.; Shang, N.; Bertino, E. Efficient Privacy Preserving Content Based Publish Subscribe Systems. In Proceedings of the 17th ACM Symposium on Access Control Models and Technologies (SACMAT), Newark, NJ, USA, 20–22 June 2012; pp. 133–144. [CrossRef]
70. Bacon, J.; Eyers, D.M.; Singh, J.; Pietzuch, P.R. Access Control in Publish/Subscribe Systems. In Proceedings of the Second International Conference on Distributed Event-Based Systems (DEBS), Rome, Italy, 1–4 July 2008; pp. 23–34. [CrossRef]
71. Singh, J.; Bacon, J.; Eyers, D. Policy Enforcement Within Emerging Distributed, Event-based Systems. In Proceedings of the 8th ACM International Conference on Distributed Event-Based Systems (DEBS), Bombay, Mumbai, India, 26–29 May 2014; pp. 246–255. [CrossRef]
72. Dave, S.; Mahadevia, J.; Trivedi, B. Security Policy Implementation Using Connection and Event Log to Achieve Network Access Control. In Proceedings of the International Conference on Advances in Computing and Artificial Intelligence (ACAI), Freiburg, Germany, 7–10 June 2011; pp. 29–33. [CrossRef]
73. Wun, A.; Jacobsen, H.A. A Policy Management Framework for Content-based Publish/Subscribe Middleware. In Proceedings of the ACM/IFIP/USENIX 2007 International Conference on Middleware (Middleware), Newport Beach, CA, USA, 26–30 November 2007; pp. 368–388.
74. Singh, J.; Eyers, D.M.; Bacon, J. Disclosure Control in Multi-domain Publish/Subscribe Systems. In Proceedings of the 5th ACM International Conference on Distributed Event-based System (DEBS), Yorktown Heights, NY, USA, 11–14 July 2011; pp. 159–170. [CrossRef]
75. Papagiannis, I.; Migliavacca, M.; Pietzuch, P.; Shand, B.; Eyers, D.; Bacon, J. PrivateFlow: Decentralised Information Flow Control in Event Based Middleware. In Proceedings of the Third ACM International Conference on Distributed Event-Based Systems (DEBS), Nashville, TN, USA, 6–9 July 2009; pp. 38:1–38:2. [CrossRef]
76. Pietzuch, P.; Migliavacca, M.; Bacon, J.; Eyers, D.; Sigh, J.; Shand, B. Security in Multi-domain Event-based Systems. *IT Inf. Technol.* **2009**, *51*. [CrossRef]
77. Bacon, J.; Moody, K.; Yao, W. A Model of OASIS Role-based Access Control and Its Support for Active Security. *ACM Trans. Inf. Syst. Secur.* **2002**, *5*, 492–540. [CrossRef]
78. Fuchs, A.P.; Chaudhuri, A.; Foster, J.S. *SCanDroid: Automated Security Certification of Android Applications*; Technical Report; University of Maryland: College Park, MD, USA, 2009 .
79. Gibler, C.; Crussell, J.; Erickson, J.; Chen, H. AndroidLeaks: Automatically Detecting Potential Privacy Leaks in Android Applications on a Large Scale. In Proceedings of the 5th International Conference on Trust and Trustworthy Computing (TRUST), Vienna, Austria, 13–15 June 2012; pp. 291–307.
80. Zhou, Y.; Jiang, X. Detecting Passive Content Leaks and Pollution in Android Applications. In Proceedings of the 19th Annual Network & Distributed System Security Symposium (NDSS), San Diego, CA, USA, 24–27 February 2013.
81. Safi, G.; Shahbazian, A.; Halfond, W.G.; Medvidovic, N. Detecting Event Anomalies in Event-Based Systems. In Proceedings of the 10th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE), Bergamo, Italy, 30 August–4 September 2015; pp. 25–37.
82. Enck, W.; Gilbert, P.; Chun, B.G.; Cox, L.P.; Jung, J.; McDaniel, P.; Sheth, A.N. TaintDroid: An Information-flow Tracking System for Realtime Privacy Monitoring on Smartphones. In Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation (OSDI), Vancouver, BC, Canada, 4–6 October 2010; pp. 393–407.
83. Arzt, S.; Rasthofer, S.; Fritz, C.; Bodden, E.; Bartel, A.; Klein, J.; Le Traon, Y.; Octeau, D.; McDaniel, P. Flowdroid: Precise Context, Flow, Field, Object-Sensitive and Lifecycle-Aware Taint Analysis for Android Apps. In Proceedings of the 35th Annual ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI), Edinburgh, UK, 9–11 June 2014; pp. 259–269.

84. Grace, M.C.; Zhou, Y.; Wang, Z.; Jiang, X. Systematic Detection of Capability Leaks in Stock Android Smartphones. In Proceedings of the 19th Annual Network & Distributed System Security Symposium (NDSS), San Diego, CA, USA, 5–8 February 2012.
85. Gordon, M.I.; Kim, D.; Perkins, J.H.; Gilham, L.; Nguyen, N.; Rinard, M.C. Information Flow Analysis of Android Applications in DroidSafe. In Proceedings of the 22nd Annual Network & Distributed System Security Symposium (NDSS), San Diego, CA, USA, 8–11 February 2015.
86. Yang, Z.; Yang, M.; Zhang, Y.; Gu, G.; Ning, P.; Wang, X.S. AppIntent: Analyzing Sensitive Data Transmission in Android for Privacy Leakage Detection. In Proceedings of the ACM SIGSAC Conference on Computer Communications Security (CCS), Berlin, Germany, 4–8 November 2013; pp. 1043–1054.
87. Chan, P.P.; Hui, L.C.; Yiu, S.M. DroidChecker: Analyzing Android Applications for Capability Leak. In Proceedings of the 5th Conference on Security and Privacy in Wireless and Mobile Networks (WISEC), Tucson, AZ, USA, 16–18 April 2012; pp. 125–136.
88. Felt, A.P.; Chin, E.; Hanna, S.; Song, D.; Wagner, D. Android Permissions Demystified. In Proceedings of the 18th ACM Conference on Computer and Communications Security (CCS), Chicago, IL, USA, 17–21 October 2011; pp. 627–638.
89. Kim, J.; Yoon, Y.; Yi, K.; Shin, J. ScanDal: Static Analyzer for Detecting Privacy Leaks in Android Applications. In Proceedings of the Mob. Secur. Technol. (MoST), San Francisco, CA, USA, 24 May 2012.
90. Yang, Z.; Yang, M. LeakMiner: Detect Information Leakage on Android with Static Taint Analysis. In Proceedings of the 3rd World Congress on Software Engineering (WCSE), London, UK, 4–6 July 2012; pp. 101–104.
91. Yang, S.; Yan, D.; Wu, H.; Wang, Y.; Rountev, A. Static Control-Flow Analysis of User-driven Callbacks in Android Applications. In Proceedings of the 37th International Conference on Software Engineering (ICSE), Florence, Italy, 16–24 May 2015; pp. 89–99.
92. Mann, C.; Starostin, A. A Framework for Static Detection of Privacy Leaks in Android Applications. In Proceedings of the 27th Symposium on Applied Computing (SAC), Trento, Italy, 26–30 March 2012; pp. 1457–1462.
93. Huang, J.; Zhang, X.; Tan, L.; Wang, P.; Liang, B. AsDroid: Detecting Stealthy Behaviors in Android Applications by User Interface and Program Behavior Contradiction. In Proceedings of the 36th International Conference on Software Engineering (ICSE), Hyderabad, India, 31 May–7 June 2014; pp. 1036–1046.
94. Wei, F.; Roy, S.; Ou, X.; Robby. Amandroid: A Precise and General Inter-Component Data Flow Analysis Framework for Security Vetting of Android Apps. *ACM Trans. Priv. Secur.* **2018**, *21*, 1–32. [CrossRef]
95. Klieber, W.; Flynn, L.; Bhosale, A.; Jia, L.; Bauer, L. Android Taint Flow Analysis for App Sets. In Proceedings of the 3rd International Workshop on the State of the Art in Java Program Analysis (SOAP), Santa Barbara, CA, USA, 14 June 2016; pp. 1–6.
96. Octeau, D.; Luchaup, D.; Dering, M.; Jha, S.; McDaniel, P. Composite Constant Propagation: Application to Android Inter-Component Communication Analysis. In Proceedings of the 37th International Conference on Software Engineering (ICSE), Florence, Italy, 16–24 May 2015; pp. 77–88.
97. Octeau, D.; McDaniel, P.; Jha, S.; Bartel, A.; Bodden, E.; Klein, J.; Le Traon, Y. Effective Inter-component Communication Mapping in Android with Epicc: An Essential Step Towards Holistic Security Analysis. In Proceedings of the 22nd USENIX Conference on Security (SEC), Washington, DC, USA, 14–16 August 2013; pp. 543–558.
98. Li, L.; Bartel, A.; Klein, J.; Traon, Y.L.; Arzt, S.; Rasthofer, S.; Bodden, E.; Octeau, D.; Mcdaniel, P. I know what leaked in your pocket: Uncovering privacy leaks on Android Apps with Static Taint Analysis. *arXiv* **2014**, arXiv:1404.7431 .
99. Li, L.; Bartel, A.; Klein, J.; Traon, Y.L. Automatically Exploiting Potential Component Leaks in Android Applications. In Proceedings of the 13th International Conference on Trust, Security and Privacy in Computing and Communications (TRUSTCOM), Beijing, China, 24–26 September 2014; pp. 388–397.
100. Ravitch, T.; Creswick, E.R.; Tomb, A.; Foltzer, A.; Elliott, T.; Casburn, L. Multi-App Security Analysis with FUSE: Statically Detecting Android App Collusion. In Proceedings of the 4th Program Protection and Reverse Engineering Workshop (PPREW), San Diego, CA, USA, 24 January 2014; pp. 4:1–4:10.

101. Shen, F.; Vishnubhotla, N.; Todarka, C.; Arora, M.; Dhandapani, B.; Lehner, E.J.; Ko, S.Y.; Ziarek, L. Information Flows As a Permission Mechanism. In Proceedings of the 29th ACM/IEEE International Conference on Automated Software Engineering (ASE), Vasteras, Sweden, 15–19 September 2014; pp. 515–526.
102. Choi, K.; Chang, B.M. A Type and Effect System for Activation Flow of Components in Android Programs. *Inf. Process. Lett.* **2014**, *114*, 620–627. [CrossRef]
103. Bartsch, S.; Berger, B.; Bunke, M.; Sohr, K. The Transitivity-of-Trust Problem in Android Application Interaction. In Proceedings of the 8th International Conference on Availability, Reliability and Security (ARES), Regensburg, Bavaria, Germany, 2–6 September 2013; pp. 291–296.
104. Zhongyang, Y.; Xin, Z.; Mao, B.; Xie, L. DroidAlarm: An All-Sided Static Analysis Tool for Android Privilege-Escalation Malware. In Proceedings of the 8th ACM SIGSAC Symposium on Information, Computer and Communications Security (CCS), Hangzhou, China, 7–10 May 2013; pp. 353–358.
105. Ernst, M.D.; Just, R.; Millstein, S.; Dietl, W.; Pernsteiner, S.; Roesner, F.; Koscher, K.; Barros, P.B.; Bhoraskar, R.; Han, S.; Vines, P.; Wu, E.X. Collaborative Verification of Information Flow for a High-Assurance App Store. In Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security (CCS), Scottsdale, AZ, USA, 3–7 November 2014; pp. 1092–1104.
106. Wu, J.; Cui, T.; Ban, T.; Guo, S.; Cui, L. PaddyFrog: Systematically Detecting Confused Deputy Vulnerability in Android Applications. *Secur. Commun. Netw.* **2015**, *8*, 2338–2349. [CrossRef]
107. Octeau, D.; Jha, S.; Dering, M.; McDaniel, P.; Bartel, A.; Li, L.; Klein, J.; Traon, Y.L. Combining Static Analysis with Probabilistic Models to Enable Market-Scale Android Inter-Component Analysis. In Proceedings of the 43rd Symposium on Principles of Programming Languages (POPL), St. Petersburg, FL, USA, 20–22 January 2016; pp. 469–484.
108. Wu, L.; Grace, M.; Zhou, Y.; Wu, C.; Jiang, X. The Impact of Vendor Customizations on Android Security. In Proceedings of the 2013 ACM SIGSAC Conference on Computer and Communications Security (CCS), Berlin, Germany, 4–8 November 2013; pp. 623–634.
109. Elish, K.O.; Yao, D.; Ryder, B.G. On the Need of Precise Inter-App ICC Classification for Detecting Android Malware Collusions. In Proceedings of the IEEE Mobile Security Technologies (MoST), in Conjunction with the IEEE Symposium on Security and Privacy, San Jose, CA, USA, 21 May 2015.
110. Backes, M.; Gerling, S.; Hammer, C.; Maffei, M.; von Styp-Rekowsky, P. AppGuard: Enforcing User Requirements on Android Apps. In Proceedings of the 19th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS), Rome, Italy, 16–24 March 2013; pp. 543–548.
111. Xu, R.; Saïdi, H.; Anderson, R. Aurasium: Practical Policy Enforcement for Android Applications. In Proceedings of the 21st USENIX Conference on Security Symposium (USENIX), Bellevue, WA, USA, 8–10 August 2012; pp. 539–552.
112. Davis, B.; Sanders, B.; Khodaverdian, A.; Chen, H. I-ARM-Droid: A Rewriting Framework for In-App Reference Monitors for Android Applications. In Proceedings of the Mobile Security Technologies (MoST), San Francisco, CA, USA, 24 May 2012.
113. Davis, B.; Chen, H. RetroSkeleton: Retrofitting Android Apps. In Proceedings of the 11th Annual International Conference on Mobile Systems, Applications, and Services (MobiSys), Taipei, Taiwan, 25–28 June 2013; pp. 181–192.
114. Jeon, J.; Micinski, K.K.; Vaughan, J.A.; Fogel, A.; Reddy, N.; Foster, J.S.; Millstein, T. Dr. Android and Mr. Hide: Fine-grained Permissions in Android Applications. In Proceedings of the 2nd ACM Workshop on Security and Privacy in Smartphones and Mobile Devices (SPSM), Raleigh, NC, USA, 19 October 2012; pp. 3–14.
115. Chen, K.Z.; Johnson, N.; Dai, S.; Macnamara, K.; Magrino, T.; Wu, E.; Rinard, M.; Song, D. Contextual Policy Enforcement in Android Applications with Permission Event Graphs. In Proceedings of the 20th Annual Network & Distributed System Security Symposium (NDSS), San Diego, CA, USA, 24–27 February 2013.
116. Zhang, M.; Yin, H. Appsealer: Automatic Generation of Vulnerability-Specific Patches for Preventing Component Hijacking Attacks in Android Applications. In Proceedings of the 21st Annual Network & Distributed System Security Symposium (NDSS), San Diego, CA, USA, 23–26 February 2014.
117. Bartel, A.; Klein, J.; Monperrus, M.; Allix, K.; Le Traon, Y. *Improving Privacy on Android Smartphones through In-Vivo Bytecode Instrumentation*; Technical Report; University of Luxembourg: Luxembourg, 2012.

118. Ongtang, M.; McLaughlin, S.; Enck, W.; McDaniel, P. Semantically rich application-centric security in Android. *Secur. Commun. Netw.* **2012**, *5*, 658–673. [CrossRef]
119. Bugiel, S.; Heuser, S.; Sadeghi, A.R. Flexible and Fine-grained Mandatory Access Control on Android for Diverse Security and Privacy Policies. In Proceedings of the 22nd USENIX Conference on Security (USENIX Security), Washington, DC, USA, 14–16 August 2013; pp. 131–146.
120. Zhao, Z.; Colon Osono, F.C. "TrustDroid^{TM}": Preventing the Use of Smartphones for Information Leaking in Corporate Networks Through the Used of Static Analysis Taint Tracking. In Proceedings of the 7th International Conference on Malicious and Unwanted Software (MALWARE), Fajardo, PR, USA, 16–18 October 2012; pp. 135–143.
121. Enck, W.; Ongtang, M.; McDaniel, P. On Lightweight Mobile Phone Application Certification. In Proceedings of the 16th ACM Conference on Computer and Communications Security (CCS), Chicago, IL, USA, 9–13 November 2009; pp. 235–245.
122. Schreckling, D.; Posegga, J.; Köstler, J.; Schaff, M. Kynoid: Real-Time Enforcement of Fine-Grained, User-Defined, and Data-Centric Security Policies for Android. In Proceedings of the 6th IFIP International Conference on Information Security Theory and Practice: Security, Privacy and Trust in Computing Systems and Ambient Intelligent (WISTP), Egham, UK, 20–22 June 2012; pp. 208–223.
123. Dietz, M.; Shekhar, S.; Pisetsky, Y.; Shu, A.; Wallach, D.S. Quire: Lightweight Provenance for Smart Phone Operating Systems. In Proceedings of the 20th USENIX Conference on Security (SEC), San Francisco, CA, USA, 8–11 August 2011; p. 23.
124. Felt, A.P.; Wang, H.J.; Moshchuk, A.; Hanna, S.; Chin, E. Permission Re-delegation: Attacks and Defenses. In Proceedings of the 20th USENIX Conference on Security (SEC), San Francisco, CA, USA, 8–11 August 2011; p. 22.
125. Hornyack, P.; Han, S.; Jung, J.; Schechter, S.; Wetherall, D. These Aren't the Droids You're Looking for: Retrofitting Android to Protect Data from Imperious Applications. In Proceedings of the 18th ACM Conference on Computer and Communications Security (CCS), Chicago, IL, USA, 17–21 October 2011; pp. 639–652.
126. Wang, X.; Sun, K.; Wang, Y.; Jing, J. DeepDroid: Dynamically Enforcing Enterprise Policy on Android Devices. In Proceedings of the 22nd Annual Network & Distributed System Security Symposium (NDSS), San Diego, CA, USA, 8–11 February 2015.
127. Hu, Y.; Neamtiu, I. Static Detection of Event-Based Races in Android Apps. In Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), Williamsburg, VA, USA, 24–28 March 2018; pp. 257–270.
128. Lau, P.T. Static detection of event-driven races in HTML5-based mobile apps. In *International Conference on Verification and Evaluation of Computer and Communication Systems*; Springer: Berlin/Heidelberg, Germany, 2019; pp. 32–46.
129. Wu, H.; Qin, Z.; Tian, X.; Sun, E.; Xu, F.; Zhong, S. Broken Relationship of Mobile User Intentions and Permission Control of Shared System Resources. In Proceedings of the 2019 IEEE Conference on Dependable and Secure Computing (DSC), Hangzhou, China, 18–20 November 2019; pp. 1–8.
130. El-Zawawy, M.A.; Losiouk, E.; Conti, M. Do not let Next-Intent Vulnerability be your next nightmare: Type system-based approach to detect it in Android apps. *Int. J. Inf. Sec.* **2020**, 1–20. [CrossRef]
131. Casolare, R.; Martinelli, F.; Mercaldo, F.; Nardone, V.; Santone, A. Colluding Android Apps Detection via Model Checking. In *Workshops of the International Conference on Advanced Information Networking and Applications*; Springer: Berlin/Heidelberg, Germany, 2020; pp. 776–786.
132. Mahesh, P.S.; Muthumanickam, K. A Security Scheme for Discovering Battery Draining Attacks in Android Smartphone. In *ICDSMLA 2019*; Springer: Berlin/Heidelberg, Germany, 2020; pp. 1908–1915.
133. Kitchenham, B. *Procedures for Performing Systematic Reviews*; Keele University: Keele, UK, 2004; Volume 33, pp. 1–26.
134. Howard, J.D. *An Analysis of Security Incidents on the Internet 1989–1995*; Technical Report; Carnegie-Mellon Univiersity: Pittsburgh, PA, USA, 1997.
135. Bacon, J.; Eyers, D.; Singh, J.; Shand, B.; Migliavacca, M.; Pietzuch, P. Security in Multi-domain Event-based Systems Sicherheit in ereignis-basierten Mehrdomänensystemen. *IT Inf. Technol.* **2009**, *51*, 277–284.
136. Chen, H.; Su, J.; Qiao, L.; Xin, Q. Malware collusion attack against SVM: Issues and countermeasures. *Appl. Sci.* **2018**, *8*, 1718. [CrossRef]

137. Rangwala, M.; Zhang, P.; Zou, X.; Li, F. A Taxonomy of Privilege Escalation Attacks in Android Applications. *Int. J. Secur. Netw.* **2014**, *9*, 40–55. [CrossRef]
138. Roy, S.; Chaulagain, D.; Bhusal, S. Static Analysis for Security Vetting of Android Apps. In *From Database to Cyber Security*; Springer: Berlin/Heidelberg, Germany, 2018; pp. 375–404.
139. Garcia, J.; Hammad, M.; Ghorbani, N.; Malek, S. Automatic Generation of Inter-Component Communication Exploits for Android Applications. In Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering (ESEC/FSE), Paderborn, Germany, 6–8 September 2017; pp. 661–671.
140. Portokalidis, G.; Homburg, P.; Anagnostakis, K.; Bos, H. Paranoid Android: Versatile protection for smartphones. In Proceedings of the 26th Annual Computer Security Applications Conference (ACSAC), Austin, TX, USA, 6–10 December 2010; pp. 347–356.
141. android.App | Android Developers. 2016. Available online: http://developer.android.com/reference/android/app/package-summary.html (accessed on 16 August 2016).
142. Li, L.; Bissyandé, T.F.; Papadakis, M.; Rasthofer, S.; Bartel, A.; Octeau, D.; Klein, J.; Traon, L. Static analysis of android apps: A systematic literature review. *Inf. Softw. Technol.* **2017**, *88*, 67–95. [CrossRef]
143. Fuentes Carranza, J.C.; Fong, P.W.L. Brokering Policies and Execution Monitors for IoT Middleware. In Proceedings of the 24th ACM Symposium on Access Control Models and Technologies (SACMAT), Toronto, ON, Canada, 4–6 June 2019; Association for Computing Machinery: New York, NY, USA; pp. 49–60.
144. Pistoia, M.; Fink, S.J.; Flynn, R.J.; Yahav, E. When Role Models Have Flaws: Static Validation of Enterprise Security Policies. In Proceedings of the 29th International Conference on Software Engineering (ICSE), Minneapolis, MN, USA, 19–27 May 2007; pp. 478–488. [CrossRef]
145. Lee, B.; Kim, S.M.; Park, E.; Han, D. MemScope: Analyzing Memory Duplication on Android Systems. In Proceedings of the 6th Asia-Pacific Workshop on Systems (APSys), Tokyo, Japan, 27–28 July 2015; pp. 19:1–19:7.
146. Silva, A.; Simmonds, J. BehaviorDroid: Monitoring Android Applications. In Proceedings of the International Conference on Mobile Software Engineering and Systems (MOBILESoft), Austin, TX, USA, 16–17 May 2016; pp. 19–20.
147. Sahu, M.K.; Ahirwar, M.; Hemlata, A. A Review of Malware Detection Based on Pattern Matching Technique. *Int. J. Comput. Sci. Inf. Technol.* **2014**, *5*, 944–947.
148. Wu, H.; Schwab, S.; Peckham, R.L. Signature Based Network Intrusion Detection System and Method. U.S. Patent 7,424,744, 9 September, 2008.
149. Anjum, F.; Subhadrabandhu, D.; Sarkar, S. Signature based intrusion detection for wireless ad-hoc networks: A comparative study of various routing protocols. In Proceedings of the Vehicular Technology Conference (VTC), Jeju, Korea, 22–25 April 2003; Volume 3, pp. 2152–2156.