A Taxonomy for Security Flaws in Event-Based Systems

Youn Kyu Lee and Dohoon Kim

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.
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: $\forall d \in D \land \forall a \in A$, where $D$ is the set of domain keywords and $A$ is the

### Table 1. Types of Event Attacks.

<table>
<thead>
<tr>
<th>No.</th>
<th>Attack Type</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>A1</td>
<td>Spoofing</td>
<td>For $V_1, V_2, M_1 \in C$ where $V_1 \neq V_2 \neq M_1$ and $\exists (V_1 \xrightarrow{x} V_2)$ and $(V_2$ contains $f)$ and $(M_1 \xrightarrow{\alpha} V_2)$: $M_1$ sent a spoofed $x$ to $V_2$ to exploit $f$ in $V_2$</td>
</tr>
<tr>
<td>A2</td>
<td>Interception</td>
<td>For $V_1, V_2, M_1 \in C$ where $V_1 \neq V_2 \neq M_1$ and $\exists (V_1 \xrightarrow{\alpha} V_2)$ and $(\epsilon$ contains $s)$ and $(V_1 \xrightarrow{\epsilon} M_1) \land \neg (V_1 \xrightarrow{\alpha} V_2)$: $M_1$ intercepted $\epsilon$, which was supposed to be sent to $V_2$, to obtain $s$</td>
</tr>
<tr>
<td>A3</td>
<td>Eavesdropping</td>
<td>For $V_1, V_2, M_1 \in C$ where $V_1 \neq V_2 \neq M_1$ and $\exists (V_1 \xrightarrow{\alpha} V_2)$ and $(\epsilon$ contains $s)$ and $(V_1 \xrightarrow{\epsilon} M_1) \land (V_1 \xrightarrow{\alpha} V_2)$: $M_1$ eavesdropped on $\epsilon$, which was supposed to be open only to $V_2$, to obtain $s$</td>
</tr>
<tr>
<td>A4</td>
<td>Confused deputy</td>
<td>For $V_1, V_2, M_1 \in C$ where $V_1 \neq V_2 \neq M_1$ and $\exists (M_1 \xrightarrow{\epsilon} V_1)$ and $(V_1$ contains $f)$ and $(M_1 \xrightarrow{\epsilon} V_2) \land (V_2 \xrightarrow{\epsilon} V_1)$: $M_1$ accessed $V_1$ by accessing $V_2$, which can access $V_1$, to exploit $f$ in $V_1$</td>
</tr>
<tr>
<td>A5</td>
<td>Collusion</td>
<td>For $V_1, M_1, M_2 \in C$ where $V_1 \neq M_1 \neq M_2$ and $\exists (M_1 \xrightarrow{\epsilon} V_1)$ and $(V_1$ contains $f)$ and $(M_1 \xrightarrow{\epsilon} M_2) \land (M_2 \xrightarrow{\epsilon} V_1)$: $M_1$ colluded with $M_2$, which can access $V_1$, to exploit $f$ in $V_1$</td>
</tr>
<tr>
<td>A6</td>
<td>Flooding</td>
<td>For $V_1, V_2, M_1 \in C$ where $V_1 \neq V_2 \neq M_1$ and $\exists (V_1 \xrightarrow{\epsilon} V_2)$ and $(M_1 \xrightarrow{\epsilon} V_2) \land (\neg (V_1 \xrightarrow{\epsilon} V_2))$: the number of $\epsilon$ is overwhelmingly greater than the average number of $\epsilon$: $M_1$ sent an overwhelming number of $\epsilon$ to hinder $V_1$ from accessing $V_2$</td>
</tr>
<tr>
<td>A7</td>
<td>Delaying</td>
<td>For $V_1, V_2, V_3, M_1 \in C$ where $V_1 \neq V_2 \neq V_3 \neq M_1$ and $\exists (V_1 \xrightarrow{\epsilon} M_1 \xrightarrow{\epsilon} V_1 \xrightarrow{\epsilon} V_2)$ and $(V_3 \xrightarrow{\epsilon} M_1 \xrightarrow{\epsilon} V_2 \xrightarrow{\epsilon} V_3)$ and $(\neg (V_1 \xrightarrow{\epsilon} M_1 \xrightarrow{\epsilon} V_2 \xrightarrow{\epsilon} V_3))$: the time interval between $\epsilon$ and $\epsilon_1$ is overwhelmingly larger than the time interval between $\epsilon_1$ and $\epsilon_2$: $M_1$ delayed the publication of $\epsilon_1$ to make $V_2$ and $V_3$ malfunction</td>
</tr>
</tbody>
</table>

$C$: a set of components, $V$: a victim component, $M$: a malicious component, $f$: sensitive functionality, $x$: sensitive information, $\epsilon$: an event, $x \xrightarrow{\alpha} y$: an event communication channel for sending an event $x$ from $x$ to $y$, $x \xrightarrow{\epsilon} y$: an event $\epsilon$ 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: $\forall d \in D \land \forall a \in A$, 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.

<table>
<thead>
<tr>
<th>D: Domain Keyword</th>
<th>A: Attribute Keyword</th>
<th>IEEE</th>
<th>ACM</th>
<th>Springer</th>
<th>Google Scholar</th>
</tr>
</thead>
<tbody>
<tr>
<td>distributed event-based systems,</td>
<td>security vulnerability,</td>
<td>104</td>
<td>624</td>
<td>1188</td>
<td>3078</td>
</tr>
<tr>
<td>event-based systems, android event,</td>
<td>security attack,</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>android intent</td>
<td>security flaw,</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>security error</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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 website. 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 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.

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

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.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>F1</td>
<td>Trap door</td>
<td>◦</td>
<td>◦</td>
<td>◦</td>
<td>◦</td>
</tr>
<tr>
<td>F2</td>
<td>Conspirator</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>F3</td>
<td>Logic/Time Bomb</td>
<td>•</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>F4</td>
<td>Covert Channel</td>
<td>◦</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>F5</td>
<td>Open Event Channel</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>F6</td>
<td>Inadequate Concurrency</td>
<td>◦</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>F7</td>
<td>Unsafe Event</td>
<td>◦</td>
<td>◦</td>
<td></td>
<td></td>
</tr>
<tr>
<td>F8</td>
<td>Unsafe Event Interface</td>
<td>◦</td>
<td>◦</td>
<td>◦</td>
<td></td>
</tr>
<tr>
<td>F9</td>
<td>Inadequate Authentication</td>
<td>◦</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>F10</td>
<td>Inadequate Resource Mgmt</td>
<td>◦</td>
<td>◦</td>
<td>●</td>
<td></td>
</tr>
</tbody>
</table>

*: 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


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]


Publisher’s Note: MDPI stays neutral with regard to jurisdictional claims in published maps and institutional affiliations.

© 2020 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (http://creativecommons.org/licenses/by/4.0/).