Intrusion Recovery Systems: A survey

David R. Matos, *Member, IEEE,* Ibéria Medeiros, *Member, IEEE,* Miguel L. Pardal, *Member, IEEE*
Miguel Correia, *Member, IEEE*

***Abstract*—Many computing services need to be accessed through the Internet, which makes them inherently exposed to a large number of cybersecurity threats. Several attacks are thwarted by intrusion *prevention* mechanisms, but with the size and complexity of current systems, an attacker can break in sooner or later. Attackers can exploit a system vulnerability or steal user access credentials. Once the intrusion occurs, it is very likely that the state of the system is corrupted by the attacker. The classical solution against these corruptions is to keep regular backups to allow the reversal of the effects of intrusions. However, changes to the state made since the last backup are permanently lost. This situation has led to the development of work on *intrusion recovery systems* that *revert* the effect of attacks on the state of the system *without losing legitimate changes*. We present a survey of the literature in the area, explaining the different approaches and suggesting open directions for future research.**

***Index Terms*—Intrusion Recovery, Integrity, Databases, File Systems, Web Applications, Cloud Computing**

Recovery (which includes return and renewal of health) is a re-gaining - regaining of a clear view.
— J.R.R. Tolkien. On Fairy Stories

I. Introduction

INTRUSION RECOVERY aims to reverse the effects of undesired operations that modify the state of an application. This is a broad definition because *state* and *application* can refer to different things. The state of a database will consist of data records that can have relationships between themselves, while the state of a file system is comprised of its files and folders. Although the term intrusion refers to illegal activity that breaks the integrity of a system [1], an intrusion recovery mechanism can also be used to reverse accidental operations caused by legitimate users. In other words, intrusion recovery techniques can also be used for general data recovery.

A cautionary note: computer security is about ensuring several properties, e.g., confidentiality, integrity, and availability [1]. Intrusion recovery is concerned with *state integrity, not confidentiality*. Undoing the effects of an attack that steals data, and breaks confidentiality is usually infeasible. So, intrusion recovery is focused on fixing the state, i.e., recovering integrity.

A. Security mechanisms

Organizations try to be aware of security risks that come from the Internet and other sources. After risks are identified, organizations select and apply intrusion prevention mechanisms, and they rely on them to prevent attacks or, at least, to reduce the probability that attacks succeed. Some of these mechanisms are configured and managed by service providers, e.g., Cloud Service Providers (CSPs), whereas others can be deployed by the client organizations. Some of these security mechanisms include: firewalls that filter inbound traffic and prevent unauthorized or risky connections from accessing private networks; access control mechanisms that prevent unauthorized or unknown users from accessing private content; and intrusion detection systems that monitor the organization's network infrastructure and systems for malicious activity.

A firewall [2] can protect a system connected to the network against unauthorized access. This kind of mechanism regulates incoming and outgoing network traffic following a set of rules. However, a system administrator may fail to configure a firewall with every necessary rule [3], allowing an attacker to find a way to illegally access a service or other resources. Once attackers have access, they may corrupt the users' data. Even if all the rules are correctly configured, in some situations, it is necessary to allow all traffic to certain components of an application, e.g., the front-end of a web application.

Access control mechanisms [4], [5] regulate which users are allowed to use the computing resources. These mechanisms are responsible for identifying, authorizing, and authenticating users, allowing system administrators to audit and manage access records. Such mechanisms are only effective against attackers when they are correctly configured and managed. If an attacker gains access to a user's credentials by, for example, stealing his computer, he will be able to perform an attack and corrupt data.

Intrusion Detection Systems (IDS) [6], [7] monitor activities, e.g., network traffic or operations in logs, to identify malicious actions. If an IDS suspects that an attack is underway, it notifies the system administrator so that he can trigger the appropriate countermeasures. Some activities may be falsely reported to the system administrator, the so-called *false positives*. Moreover, there may be attacks that are not detected, the *false negatives*.

An evolution of IDSs are Security Information and Event Management (SIEM) systems [8], which analyze activity alongside other sources and filter possible attacks from false positives and become aware of false negatives. The combination of these mechanisms provides the Security Operation Center (SOC) team with information for them to identify malicious activities, i.e., attacks, and indicators of attack (IoA), that detects an intent of an attack, or indicators of compromise (IoC), that detects when the security of the network has been breached, allowing them to take appropriate countermeasures. When some attack does get through, the system administrator

David R. Matos, Miguel L. Pardal and Miguel Correia are with INESC-ID, Instituto Superior Técnico, Universidade de Lisboa - Portugal.

Ibéria Medeiros is with LASIGE, Faculdade de Ciências, Universidade de Lisboa - Portugal.

needs to repair the system, which includes applying security patches, re-configuring the network, and reverting the corrupted data to the most recent version existing prior to the attack.

These security mechanisms make attacks less likely to succeed and compromise the state of the system. However, despite these mechanisms, malicious users may still be able to intrude systems, e.g., by exploiting a vulnerability or poor configuration [9], and, once an attacker gains access, he may corrupt data. Data corruption may result in significant losses for organizations. Ransomware is a serious problem. We mention Wannacry [10] that successfully infected more than 230,000 systems and resulted in an estimated loss of $4 (USD) billion. According to Accenture [11], the most expensive impact of a cyber-attack is information loss due to data corruption, which represents 43% of the total costs of the attack. Although cyber-attacks remain the leading cause of data loss incidents – 55% of the total incidents in 2017 – overall, human error caused even more data corruption [12]. Therefore, it can be argued that services that are capable of reversing the effects of intentional and accidental state modifications are always useful for organizations.

B. Intrusion Recovery

Intrusion recovery involves the tasks that need to be performed to reverse the effects of an attack that caused data corruption in the system *state*. It plays a critical role in incident handling and response process after an incident occurs. According to [13], [14], recovery is the third action, after preparation and detection, that needs to be taken to restore the attacked system. Some intrusions may not aim to cause data corruption but rather leak sensitive data. The works discussed in this paper do not target this attack vector. The goal of the presented papers is to provide mechanisms that allow one to restore the affected data. More specifically, tasks in an intrusion recovery system involve identifying the data that was corrupted and reverting it back to the state prior to the attack, while keeping legitimate data intact. To achieve this state, intrusion recovery mechanisms use a combination of periodic backups/checkpoints with logs of the executed operations. Using only backups/checkpoints of the state would not be a good idea, since they would allow the entire state of the system to be reverted to some point in time but lose the effect of later legitimate operations in the process. Intrusion recovery mechanisms differ from the rollback mechanisms used in transactional databases in which it is possible to revert a transaction that was not completed successfully. When an intrusion occurs, it is not possible to perform a rollback.

There are two distinct ways to perform intrusion recovery: backward recovery and forward recovery. Backward recovery involves restoring the system to a known good state that was previously saved, such as a backup or a checkpoint. Any changes made to the system after that point are discarded, and the system is returned to the saved state. This can be a time-consuming process, as it requires the system to be shut down and restarted, but it ensures that the system is restored to a consistent state.

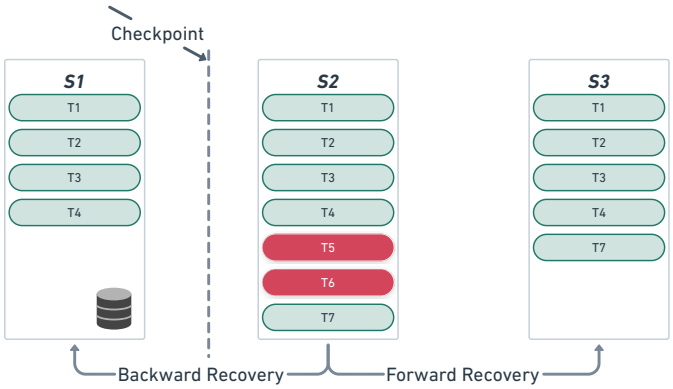

Fig. 1: Restoring a system to a previous backup vs recovery (S1, S2, S3 are states; T1 to T7 are operations).

Forward recovery, on the other hand, involves continuing to operate the system after a failure has occurred, using some form of error correction or redundancy to compensate for the failure. This approach is often used in systems that cannot afford to be shut down, such as real-time systems or critical infrastructure. In forward recovery, the system is designed to detect and correct errors as they occur, so that the system can continue to operate without interruption. While forward recovery can be more complex to implement than backward recovery, it can also be more resilient to failures, as the system can continue to operate even when errors occur.

Figure 1 presents two different approaches to restore data that was corrupted by an attack. In the figure, the state of the system evolves from *S1* to *S2* after executing the operations T5, T6, and T7. The problem is that operations T5 and T6 are malicious, corrupt the state of the system, and need to be reverted. One possible approach, called *backward recovery*, consists in reverting the entire system state back to *S1*. This is possible since a backup was performed before the attack. This discards the effects of the malicious operations T5 and T6. The problem is that transaction T7 is also discarded. A preferable solution would be to only undo the effects of the attack. This approach is called *forward recovery* which, in this example, would only discard the effects of operations T5 and T6 while keeping the effects of the transaction T7 and reaching the state in *S3*.

An intrusion recovery system can be used to achieve the desirable state *S3* of Figure 1, in which the effects of malicious operations are reverted and the legitimate one is kept [15]–[19]. Intrusion recovery systems work by recording in a log every operation that affects the state of the application. With this log, a system administrator can select unintended actions that were caused by the attack and use the intrusion recovery system to undo them. In other words, intrusion recovery mechanisms aim to revert the damage intentionally caused by attackers or accidentally by authorized users, while keeping intact data created and modified by legitimate users. These mechanisms assume that attacks have already occurred and that it is necessary to revert their effects on the state of the system. This assumption is realistic given that even adopting intrusion prevention techniques will reduce the probability of

attacks to be successful, and attackers may always discover new ways to exploit the system.

In terms of how recovery is performed, there are different approaches, but in general all of them rely on a combination of *checkpoints* and *message logs*. Checkpoints are copies of the state of the application that can be used to revert the application back to a previous point in time. Message logs record operations that can be re-executed on a checkpoint to reconstruct the state of the application until the present time. For file systems, there is an alternative approach that uses several versions of each file – multiversioned file systems – that allow corrupted files to be reverted to previous versions without affecting the remaining file system.

Recovery occurs after the effects of an intrusion are recorded in the state of the system and made available to external users. It is not possible to ensure that no user saw the effects of the intrusion before recovery was complete. This problem of external inconsistency cannot be completely avoided [20], but it can be managed. One possible solution consists of using compensating or explanatory actions to inform the user about the inconsistencies he may experience.

This survey provides an overview of the state-of-the-art in *intrusion recovery systems*. We group the systems into five sets:

1) *Generic applications* – the original work in the area, which presented intrusion recovery mechanisms that can be used with different types of application (Section III);
2) *Virtual machines* – systems that take advantage of the virtualization platform to perform snapshots and log system calls (Section IV);
3) *Multiversioned file systems* – systems that allow recovering file systems by leveraging versioning (Section V);
4) *File systems with selective re-execution* – systems that also allow recovering file systems that, instead of versioning, employ a selective re-execution approach (Section VI);
5) *Web and cloud applications* – systems for recovering web applications, including those provided as cloud computing services (Section VII).

II. Methodology

In this paper we present a systematic review of the literature on intrusion recovery. We follow the review process steps proposed in [21]. The steps are described in detail in the next sections.

A. Research questions

The research questions we aim to answer in this paper are the following.

- RQ1: How much research has been published on intrusion recovery in the last 25 years?
- RQ2: What are the target systems of the Intrusion Recovery solutions that were proposed?
- RQ3: What kind of intrusions do these works aim to recover from?
- RQ4: What are the limitations of current intrusion recovery systems?
- RQ5: How can we compare the performance of these intrusion recovery mechanisms?

TABLE I: Selected journals and conference proceedings. The number of publications is shown for each source.

Source	Acronym	Type	Nr.
USENIX Annual Technical Conf.	ATC	Conf.	1
Symp. on Operating System Principles	SOSP	Conf.	4
IEEE Network Operations and Management Symposium	NOMS	Conf.	1
Annual Computer Security Applications Conf.	ACSA	Conf.	2
ACM SIGOPS Symp. on Operating Systems Principles	SIGOPS	Conf.	1
Symp. on Operating System Design & Implementation	OSDI	Conf.	2
International Conf. on Dependable Systems and Networks	DSN	Conf.	2
ACM SIGOPS/EuroSys European Conf. on Computer Systems	EuroSys	Conf.	1
ACM/IFIP/USENIX International Middleware Conf.	Middleware	Conf.	2
International Symp. on Network Computing and Applications	NCA	Conf.	1
Conference on Distributed Computing Systems	CDCS	Conf.	1
IEEE Transactions on Dependable and Secure Computing	TDSC	Journal	1
IEEE Transactions on Cloud Computing	TCC	Journal	1
IEEE Transactions on Knowledge and Data Engineering	TKDE	Journal	1

With RQ1 we wanted to establish a time frame in which the research on intrusion recovery can be somehow comparable. In our research, we collected studies from the last 25 years. This allowed us to consider different types of systems (from email servers to blockchain applications). With RQ2 we wanted to identify the different types of systems that are targeted by the intrusion recovery mechanisms. This allowed us to categorize the selected studies and perform an apples-to-apples comparison between the systems. With RQ3 we wanted to define what an intrusion is in the context discussed systems. In our research, we noticed that the definition of intrusion depends on the target system. For example, in a web application an intrusion can be an HTTP request while in a database an intrusion can be a SQL statement. This question is answered in each of the categories that we identified. With RQ4 we wanted to know the limitations of the existing intrusion recovery mechanisms, as this can help to extent the current state-of-the-art. With RQ5 we wanted to compare the existing solutions. This comparison can only be made with system in the same category; as such, we answer this question in each category.

B. Research process

Our research process was a manual search in a selection of conferences and journals related to the field of intrusion recovery. Our criteria was selecting publications from the last 25 years. The selection of conferences and journals is presented in Table I. In the table, the first two columns refer to the publication name and acronym, the third column refers to the type, and the fourth column refers to the number of intrusion recovery publications discussed in this paper.

C. Data collection

We collected two different metrics: a) performance metrics, and b) research impact metrics. For a) for each study we collected the following metrics: performance overhead, mean recovery time, and storage overhead. These three metrics were commonly evaluated in the different intrusion recovery systems, allowing us to compare the different approaches. For b) we collected the citation number of each paper over time.

III. Recovery in generic applications

The mechanisms presented in the following sections assume that users interact with the system by executing operations, which, in turn, modify its state. Users with malicious intent may execute illegal operations that will modify the state of the application by exploiting vulnerabilities or by accessing the system on the other users' behalf. These illegal operations are intrusions that the system administrator wishes to reverse. Normal users may also accidentally execute unintended operations, and, although these are not intrusions, in the sense that they were not purposely executed, they should be undone from the state of the system.

A. The three R's approach

The three R's approach [22] consists of a technique that provides a system-level undo operation, offering the possibility to roll back unintended actions performed by human operators, viruses, hacker attacks and unpredictable problems that are detected too late to be contained. The motivation behind this work is the fact that human operator error has been the leading cause of outages [23]–[25] and the fact that it takes a considerable amount of time to reverse the effects of the error and restore the service. The authors of the article defend that future systems should be designed from the beginning with recovery mechanisms similar to the undo operations found in widely used applications, such as word processors and spreadsheets.

The mechanism works in three steps: rewind, repair, and replay. In the *rewind* step, the system state is reverted to a backup prior to the error. In the *repair* step, a system administrator applies the required corrections to the system: applying a software patch to the system, omitting erroneous operations, or fixing a bug in the code. In the *replay* step, the undo system reexecutes every user operation since the backup, allowing the corrections added in the repair step to be executed as well.

The three R's approach is capable of undoing human operator errors. For that, it is necessary to record the executed operations at the user level. The proposed architecture consists in having a proxy intercepting every user-level operation so they can be recorded in an operation log and a *time-travel* storage unit is then responsible for making periodic backups in order to allow rewinding the system to a previous point in time.

B. An implementation of the three R's approach

An implementation of the three R's approach is *Operator Undo* [15]. It offers the undo operation in e-mail systems so that a system administrator can revert unintended actions. The system architecture is similar to the one presented in [22] with the addition of a Control UI that serves as an interface for system administrators. Figure 2 presents the system architecture of Operator Undo. In the figure, the *time-travel storage* and the *service application* (the blue components) belong to the application (with some minor modifications) that is protected by the mechanism. The *undo proxy*, *control UI*, *undo manager* and the *timeline storage* (the green components) are from Operator Undo. The proxy, used to intercept user requests, is specific to the application it is wrapping. This allows the proxy to identify and register operations at the user level, making it possible for the administrator to select and undo user operations, instead of storage level operations. While the proxy adopts the same protocol as the application, to identify the user-level operations, the remaining components of the system assume a generic interpretation for the operations. This allows porting Operator Undo to different kinds of systems by requiring only the proxy to be modified.

The implementation of Operator Undo in an e-mail service was made without code modifications to the e-mail server; instead, the components of Operator Undo are set up to wrap the e-mail server. The compensation operations that deal with the inconsistencies observed by the user, such as missing e-mail messages, are solved by informing the users that maintenance was done. The *time-travel* storage performs automatic backups hourly, which are converted into daily backups at the end of the day. This allows one to rewind the system to at most a day prior to the intrusion, before repairing and replaying the verbs.

Most of the works presented in this survey share some similarities with the papers presented in this section.

C. The Response and Recovery Engine

The Response and Recovery Engine (RRE) [26] is an automated cost-sensitive intrusion response system that uses game theory to model the security battle between itself and an attacker. It employs a multistep, sequential, hierarchical, nonzero-sum, two-player stochastic game approach to evaluate various security properties of individual host systems in a network. RRE leverages attack-response trees (ARTs) to consider inherent uncertainties in intrusion detection system

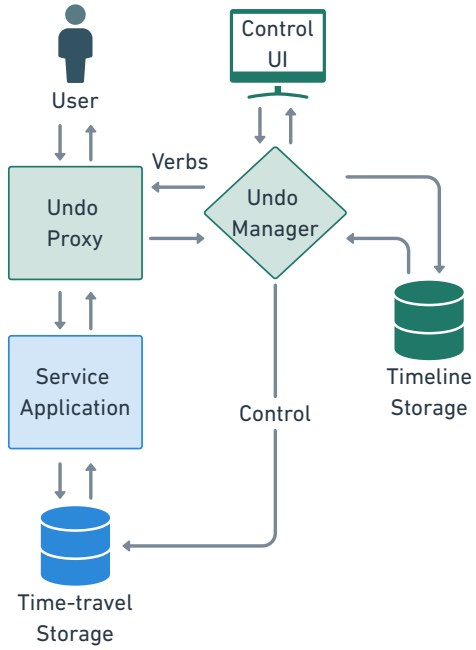

Fig. 2: System architecture of Operator Undo [15]

(IDS) alerts when estimating system security and deciding on response actions. It converts ARTs into partially observable competitive Markov decision processes to find the optimal response action against the attacker. RRE has a two-layer architecture, consisting of local engines and a global engine, to deal with security issues of different granularities. It uses a fuzzy control-based technique to support network-level intrusion response, allowing network security administrators to define high-level network security properties through easy-to-understand linguistic terms. RRE extends the state-of-the-art in intrusion response by accounting for planned adversarial behavior, inherent uncertainties in IDS alert notifications, and the ability to define high-level network security properties. It has been demonstrated to be computationally efficient for relatively large networks and practical for critical infrastructure (e.g., power grid) networks.

IV. Intrusion Recovery in Virtual Machines

Some intrusion recovery mechanisms take advantage of the capabilities of virtualization to log system-level operations and provide recovery at the system level. The following works explore the problem of intrusion recovery in servers that run in *virtual machine appliances*, i.e., pre-configured virtual machines that run on a *hypervisor*.

The intrusion recovery systems presented in this section allow to revert unintended actions that occur in a virtual machine (VM). This includes every unintended operation that causes modifications to the state of the virtual machine. The works presented in this section make the following assumptions:

- **State:** is contained in a file system that is managed by the hypervisor and is isolated from the host file system;
- **Interface:** users interact with the virtual machine through an hypervisor, the user's actions can be described in the form of *system calls* that may or may not modify the state of the virtual machine;
- **Intrusion:** is any operation that is performed in through the hypervisor that generated system calls that cause modifications to the state of the virtual machine;
- **Recovery:** is done by reverting the files that were modified and deleting any file that was created by the intrusion.

A. Tracking intrusions through the operating system

There are solutions that are able to track intrusions in the operating system. In this section we discuss BackTracker [27], Bezoar [28], and SHELF [29].

1) BackTracker: Backtracker [27] is a tool that analyzes how an intrusion propagates through the operating system to perform adequate recovery measures. Backtracker helps system administrators identify the sequence of steps that occur in an intrusion. It works backwards from the detection point, i.e., it starts from the state in the file system that alerts the administrator of the intrusion, and identifies a chain of events that could have led to the erroneous state. During runtime, *EventLogger* collects logs of events in the system. When a system administrator wishes to analyze a detection point, a graph generator – *GraphGen* – creates dependency graphs with the chain of events that cause the state modification.

BackTracker does not provide recovery mechanisms to revert the effects of intrusions, but a forensic tool that helps system administrators analyzing the trace of events of an intrusion in the system. This tool can be used together with a file system recovery mechanism to effectively eliminate the effects of an attack.

2) Bezoar: Bezoar [28] is an operating system and application-independent intrusion recovery system that is capable of tracking the effects of intrusions and recovering the entire state of the system while imposing low overhead on the virtual machine.

To start the re-execution of legitimate operations and recover the system, the administrator needs to identify the source of the network by selecting the network source identification in the memory unit. Then, for each operation in the log, Bezoar compares the network source of the operation with the one pointed out by the administrator. If they are different, then they are executed since they did not originate from the network intruder. If they are equal, then the operation is not executed and the system starts a *semi-replay* phase. In this phase, the algorithm presented in [22] is executed, i.e., a selective reexecution of the legitimate and non-tainted operations. When every valid operation in the log is executed, the system enters a new valid state in which the intrusions have never occurred.

3) Online recovery with quarantined objects: Bezoar provides online recovery by performing recovery concurrently with the normal execution of the system. This technique maintains availability during recovery, however, it fails to contain the propagation of the intrusion. A system that recovers the effects of an intrusion, avoiding having it infect other legitimate files in the system is SHELF [29].

SHELF maintains a cumulative clean state of the system for applications and files affected by the intrusion, allowing them to resume execution with the most recent version prior to the attack. SHELF uses taint tracking techniques to assess the damage caused by the intrusion and quarantine methods to contain infected files, allowing uninfected process and files to be available to the users.

SHELF was implemented on top of a light-weight virtual machine, performing most of its functionality at the hypervisor layer. This provides a transparent execution environment for the users and imposes minimal interference to the guest system. It works in three phases: *normal* phase, when it logs operations and performs state recording for the system objects; *damage assessment* phase, when it determines the infected objects taking into account the dependencies; and *recovery* phase, when infected object are quarantined and reverted to the latest, non-infected, versions.

V. Intrusion Recovery in File Systems

Intrusion recovery in a file system can be done by reverting the affected files to the latest legitimate version. This technique imposes two requirements: one, the file system keeps multiple versions of files; and two, attackers cannot tamper with the previous versions of the files. The file system also needs to manage multiple versions of the files.

The intrusion recovery systems presented in this section and in Section VI allow to revert unintended actions that occur in

a file system. To this end the works presented in this section make the following assumptions:

- **State:** is a set of files and folders that are hierarchically structured, with every object being dependent (i.e., lower in the hierarchy) to a folder or the root of the file system;
- **Interface:** users interact with the file system by executing file system operations. The following operations are expected in the following works: read, create, update, delete and move;
- **Intrusion:** is an unintended file system operation that creates, modifies or deletes a file or folder from a file system;
- **Recovery:** is done by restoring the files and folders that were modified or deleted and deleting any files and folders that were created by the intrusion.

A. Elephant File System

The Elephant File System (EFS) [30] retains every version of each file, allowing users to revert unwanted operations by recovering a previous copy of the affected files. By keeping previous versions of the files in a secure storage, a user is able to revert any type of attack that corrupts his data.

In order to recover corrupted files, the file system has to satisfy two requirements: first, users should be able to undo changes to recent versions; and second, it is important to keep a long-term history of important files.

EFS allows users to define landmarks of versions, but it also implements an automatic mechanism to do so. This strategy protects users against their own mistakes. The automatic mechanism works as follows: in the short-term, every version of the file is kept, then an automatic routine cleans up the versions that should not be necessary for the user because they are too similar. In other words, in a short time frame, every version is important for the user, but the longer it passes, the less important closer versions are to the user. For older versions, only the most recent version of each batch of updates should be relevant for the user. This allows the automatic routine to propose intermediate versions to be discarded.

B. Self-Securing Storage

S4 [31] is a self-securing storage server that allows system administrators to analyze the effects of intrusions and recover from them by reverting files to previous versions. It uses a log-structured object system to keep versions of the stored objects and a journal-based structure for the versions of the metadata.

An *history pool* keeps previous versions of files that can be restored through a *copy forward* operation that copies a previous version to the current one. Old versions of the files are only kept for a specific time window called *detection window*, after which the old versions are garbage collected and cannot be recovered.

The S4 system uses a *client daemon* that translates file system requests to S4 requests. This allows S4 to be deployed in an existing file system without modifying the code of the file system. S4 provides a multiversioned object storage feature to existing network-attached storage servers. The recorder logs during normal execution allow the system administrator and users to analyze and revert unintended actions. The recovery approach is based on reverting the affected files to previous versions, as opposed to re-executing legitimate operations.

C. An intrusion recovery plugin for file systems

In some organizations, it is complex and costly to update or completely substitute existing systems. This can happen because of: licensing, lack of know-how or complexity of the infrastructure. This is a problem for system administrators who want to be able to recover their systems from intrusions. One solution for this problem is the *Repairable File Service* (RFS) [32].

RFS adds a repair feature to an existing Network File Server (NFS). RFS provides roll-back features that allow any file to be reverted to previous versions and keeps track of inter-process dependencies to determine the damage caused by an intrusion. During normal operation of the file system, RFS records write operations and inter-process dependencies to a log. When undesired operations need to be undone, RFS selectively rolls back the affected and contaminated files.

RFS provides two possible recovery approaches: forward recovery, which works by rolling back the system state to the most recently cleaned snapshot and selectively re-executing legitimate operations; and backward recovery, which works by undoing contaminated operations until the system is cleaned using *undo records* which are operations that revert the file to the previous version.

D. Intrusion recovery in cloud file storage

Cloud storage offers an API that is accessible through the Internet that allows applications to access it the same way they access network file systems. Cloud storage services or systems are managed by a CSP, meaning that the file owners do not have access to modify the file system. This makes the process of recovering from intrusion complex, as it is not possible to implement, for example RFS.

RockFS [33] is a recovery system for cloud-backed file systems that allows to recover files that were affected by intrusions. RockFS assumes a system architecture in which the cloud file system can be composed of a single cloud or a cloud-of-clouds [34]. This second model consists in a cloud service that is supported by a set of clouds.

RockFS relies on file versions provided by the backend cloud storage service(s). It protects data confidentiality before the CSP using a combination of secret sharing [35] and erasure codes [36] to split the logs in several cloud providers. RockFS also needs to deal with the fact that the attacker may try to tamper with the logs (e.g., removing the trace of his attack) and, that if that happens, they should not be used to recover the file system. To do so, RockFS uses a mechanism called Forward Secure stream integrity [37] that links every entry of the log using secret keys and hashing, making it possible to detect when a single entry of the log was deleted or modified.

VI. Intrusion Recovery in File Systems with Selective Re-execution

In the previous section we presented works that explore multiversioned file systems. These systems allow administrators to

recover from intrusions by reverting affected files to previous versions. Another technique that can be used in file systems consists in adopting the three R's approach, i.e., reverting the state or affected files to a previous point in time, repair the system and replay every legitimate operation. The following works perform intrusion recovery in file systems using this approach.

A. Tracking intrusions using taint techniques

One technique used to track the effects of intrusions in the system is by marking the affected files and propagating the mark throughout every file that shares a relation with it. This technique is called *tainting*, and the marked files are called *tainted*. In a file system, in order to apply the taint tracking technique, it is necessary to monitor files and processes and keep track of the operations in which they interact. One system that does this is *Taser* [16].

Taser uses taint tracking techniques to mark processes and files when they are read and written, creating a dependency graph. This allows to isolate intrusion operations from legitimate ones and keep the effects of legitimate users' operations in the system even if they depend on tainted operations.

Taser allows to perform recovery with two different algorithms: *simple redo* that executes the legitimate operations on top of the snapshot of the entire file system; and a *selective redo* that first checks if the non-tainted objects are legitimate; if so, it only reverts the affect files to the version present in the snapshot and only re-executes the legitimate operations that affect these objects.

B. Quarantining untrusted files

While Taser is capable of tracking the effects of intrusions and recovering from them, it does so at the cost of keeping logs of every executed operation in the system. This can be costly in many systems and given that only some processes may be responsible for intrusions, keeping logs of every operation is a waste of space. A different approach would be to keep logs only of the operations that can be performed by an attacker. One way to achieve this it by having isolation environments for suspect files and processes. This way the logger would only monitor operations from the suspect and isolated, environment. One system that adopts this approach is *Solitude* [38].

Solitude is an application-level isolation and recovery system that, besides simplifying the intrusion recovery process, also limits the effects of attacks by using a restricted privilege isolation environment to run untrusted applications. This strategy is different from common file systems that share the same namespace between every user and process in the system. Instead, Solitude uses its own file system, called IFS, that uses a copy-on-write technique to provide an isolation environment for each untrusted application. If a user detects that an application is malicious, he can discard the application's IFS environment without losing valid legitimate data from other users and applications in the file system.

Recovery starts when the user selects the file from the IFS and the synchronization commit as a starting point for recovery. Then, the recovery process generates the tainted dependency graph. The affected files are manually verified to ensure the correctness of recovery. Then the files are rolled back to an untainted state, which is marked in a snapshot right before the commit of IFS. Finally, every operation that affected the files is re-executed until the before moment commit, marked by the user as the intrusion, was done.

C. Configurable recovery algorithms

Back to the Future allows malware removal and, as a result, repairs the system by undoing the effects of the malware while keeping every valid effect caused by legitimate operations intact. The framework uses sandbox environments to bound the scope of untrusted processes in the system. These untrusted processes are defined by the user.

Back to the Future provides two approaches to recover from intrusions: a basic approach and a refined approach. The basic approach works by monitoring every operation in the system and, after an intrusion is detected, undoing all the logged write operations of every process (trusted and untrusted) and redoing all the logged operations for the trusted processes. The refined approach avoids undoing and redoing write operations that do not need to be recovered. This approach is motivated by the fact that it is only be necessary to undo untrusted operations when they occur after trusted operations since, according to the authors, trusted process legitimates the file by writing on it.

D. Intrusion recovery using selective re-execution

Retro [39] is an intrusion recovery system for desktops and servers. During normal operation, Retro logs user's operations in an *action history graph*, which describes in detail the system execution of operations triggered by users. When a system administrator detects an intrusion he selects the faulty operation (or operations) issued by the attacker, for example, a TCP connection or an HTTP request, and starts recovery. Retro performs recovery using *selective re-execution*, in other words, it rolls back the state of the system to a point in time prior to the intrusion, then it re-executes every valid operation in the log. The system administrator is responsible for dealing with any conflict that arises during recovery. This way the effects of the intrusion are wiped out of the system while every valid state modification remains intact.

After rolling back the system state to a point in time previous to the intrusion, the recovery starts. First, malicious operations in the action history graph are replaced by benign ones. For example, a faulty operations that consists in appending data to an existing file then is replaced with a similar operation that appends zero bytes to that file. Second, Retro re-executes the operations in the action history graph, ignoring those that maintain the same state after execution.

VII. Intrusion Recovery in Web and cloud

Most web applications (*web apps*) store their state in databases and cloud storage offerings. It is common to find web apps that use a single database and the file system to store their state. For example, most Wordpress [40] sites have

an architecture similar to the one presented in Figure 3, in which the code of the application runs in an Application Server and the state is distributed between the database and the file system.

Intrusion recovery systems that are designed for web applications allow to revert unintended actions that occur in the application level. To this end, the works presented in this section make the following assumptions:

- **State:** is in a database in the form of records, documents or a different data structure. It can also be stored in a file system in the form of files and folders;
- **Interface:** users interact with by executing HTTP requests which may cause modifications to the state;
- **Intrusion:** is an unintended HTTP request that was issued by an attacker or by a legitimate user that caused modifications to the state;
- **Recovery:** reverts the state of the application that was corrupted which includes database and file system;

A. Intrusion recovery through transaction support

Database Management System (DBMS) allow to store the state of applications in a structured form. Applications interact with DBMS by executing transactions using a Structured Query Language (SQL). The intrusion recovery systems presented in this section were designed to analyse the semantic of the DBMS in order to revert the effects of intrusions.

1) Recovery from malicious transactions: Ammann et al. [19] proposed an intrusion recovery framework designed to be built on top of an off-the-shelf Database Management System (DBMS).

Dependencies between transactions are taken into account by the recovery algorithm, which marks every transaction with some dependencies to a bad transaction also as bad. These dependency rules help the recovery process in two ways: first, it does not require a good transaction that does not depend on any bad transaction to be undone and re-executed; and second, only the effects caused, directly or indirectly, by the intrusion are undone.

The recovery algorithm works by analyzing the logs forward, from the intrusion point, and marking every suspicious transaction. Then, the algorithm does backwards to undo every marked transaction. The recovery algorithm can be executed in a *coldstart*, which requires the database to be halted during recovery, or with a *warmstart*, which repairs the database concurrently with the user's transactions. For the warmstart approach every transaction is submitted to a *scheduler* which prioritizes the undo transactions over the user's operations.

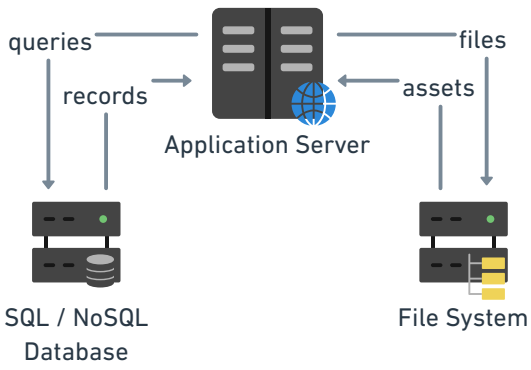

Fig. 3: Example of a Web application with a Database and a Filesystem.

2) Data recovery for web applications: Akkuş et al. [41] present a recovery system for web apps that use databases to store their state. It was designed for web apps that follow a layered architecture, i.e., presentation, logic and state.The system is composed of a proxy that logs application-level requests and two components that analyze data before recovery.

When an administrator selects one or more initial requests that cause the intrusion, the analyzer traces the dependency graph. By using a taint-based interpreter, the system is capable of associating the application-level requests with the corresponding database queries.At the user level, the system collects information available in the session cookies, which allows the administrator to identify which user caused which actions.

Recovery is done by executing compensating operations to the application state. The compensating transactions are calculated by computing a reverse query that when executed, reverts the database row to the previous value. These compensating transactions are executed in reverse order in the database undoing, version by version, every undesired update.

3) NoSQL Undo: The previous works [19], [41] allow to recover web applications that use SQL databases (DBs) to store their state. This type of DB is used by a considerable share of web applications [42], however, NoSQL databases are an alternative to SQL databases. NoSQL DBs provide a different interface and a different data model. NoSQL Undo [43] was designed to be used without requiring modifications to the source code of the database or application using it. It leverages the logging mechanism of the database to perform recovery after the fact, i.e., it is not necessary to setup NoSQL Undo before running the application instead, when a system administrator finds a vulnerability he can install NoSQL undo to revert the affected documents.

NoSQL undo provides two recovery mechanisms: *full recovery* and *focused recovery*. Full recovery works by reverting the entire database to the most recent snapshot prior to the intrusion, then NoSQL Undo reconstructs the entire database without re-executing the operations caused by the intrusion. This method requires the entire database to be offline during recovery. The focused recovery mechanism works by reconstructing the documents affected by the intrusion, allowing the database to be online during recovery.

4) Warp: The classic system model of web applications assumes the existence of a database in which the entire state of the application is stored. Data in the database is produced by statements that are generated by an application server that receives HTTP requests. In other words, attackers perform malicious HTTP requests that will result in one or more malicious database queries that need to be undone. One system that adopts this model is Warp [44].

Recovery in Warp starts when the administrator applies a security patch to the application's code. Then Warp rolls back

the system to a checkpoint prior to the time the administrator wants to apply the patch and replays every action that happened after the checkpoint. Warp reuses two components from Retro [39]: the action log and the repair controller. Besides these components, it also includes a browser with an extension, a versioning database, an HTTP log, an application runtime and a manager that is attached to the repair controller.

Warp uses a time travel database that allows re-executing only a portion of the total number of database queries in the log, and it allows recovery while the application is online serving users. For recovery Warp does not revert an entire table, instead it is only reverts the affected rows. The time travel database stores every version of every row in the database. This allows to rollback the affected rows and re-execute queries in the same version as the original execution.

B. Recovery in microservices applications

The previous works presented in this section assume a system model in which the web application is running in an isolated environment. Although this may be true for many web applications, in some cases this system model does not apply. Some web applications need to interact with external web services during their execution. In this kind of system, intrusions may propagate through several services requiring the recovery process to repair every affected service.

1) AIRE: Aire is one system that assumes this system model in which the application may be distributed to external web services [45]. Aire was designed for applications composed of interconnected web services. Aire runs in each web service gathering information about the received and sent requests in order to track dependencies across services. Aire performs recovery locally by rolling back the state and selective re-executing every valid operation. Once it finishes, it propagates the recovery actions to the dependent web services. Aire performs the repair in a distributed manner, as opposed to having a central coordinator managing the web services. This is because of the distributed nature of this kind of application and the fact that during recovery some services may be down, which would delay a centralized recovery process. Instead, each service executes recovery immediately and queues the recovery messages to be propagated when the target services are online.

2) μVerum: μVerum is an intrusion recovery service designed for microservices applications [46]. It recovers the affected services by executing compensating operations. μVerum can be deployed progressively, meaning that developers can start with a subset of microservices and gradually extend it to the entire application (if necessary). Recovery is guided by a dependency graph that traces the user's request from the moment it reaches the application, passes through every microservice it invokes until it leaves the application. Recovery is done while the application is running, not requiring it to be offline during this process.

One characteristic of μVerum is that it requires the developers to write the compensating transactions that will be executed during recovery. This allows the recover process to be more suited for particular instances, e.g., when some specific operations need to be executed during recovery like notifying the user that he may experience some inconsistencies in the application.

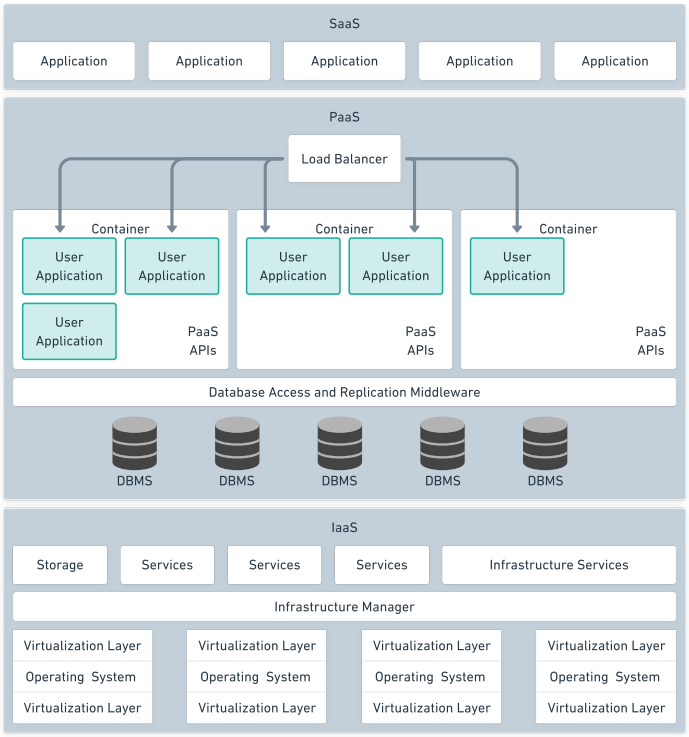

Fig. 4: Simplified architecture of a cloud showing the three models: SaaS, PaaS, IaaS.

C. Recovering web applications in the cloud

The cloud computing model facilitates how administrators deploy their applications, but also limits the functionality of the application servers. There are different flavours of the cloud that aim to provide services for different ends. In [47] the authors present three cloud computing models: the Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS). Figure 4 presents an example of the architecture of a cloud composed of these three computing models. In the figure, the computing models are stacked, i.e., the IaaS provides virtual machines (VMs) for the PaaS which, in turn, provides an execution environment for applications that are provided in a SaaS mode.

1) Shuttle: In the Platform-as-a-Service model, the system administrator has access to an execution environment with automatic scaling capabilities. One limmitation is that the administrator cannot modify the execution environment. This complicates the process of setting up an intrusion recovery mechanism, since most of them require heavy and system specific modifications in order to work. An intrusion recovery system that was build with this limitation in mind is Shuttle [48].

Shuttle works in two modes: *normal operation* and *recovery*. In normal operation, the application serves the users while Shuttle collects operation logs and performs periodic checkpoints of the application's state. These checkpoints are

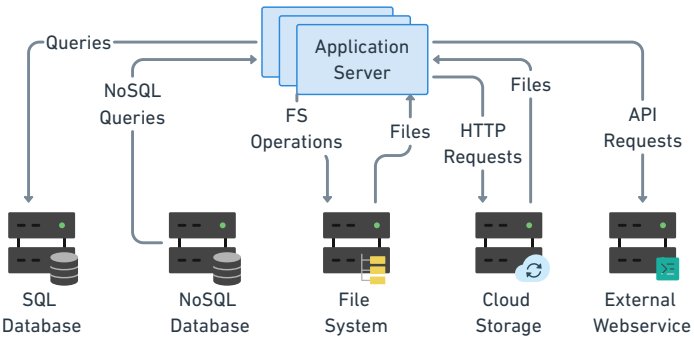

Fig. 5: Example system architecture of a web application with multiple data repositories.

created without interrupting the application. To do so, Shuttle employs a *copy-on-write* strategy in which an object is only replicated when it is being written. In recovery, Shuttle creates a branch of the application loaded with a snapshot of the application created before the intrusion. It then re-executes the valid requests in this branch while the application still continues to serve users.

To recover, Shuttle allows the administrator to select from *full replay* and *selective replay*. Full replay requires every legitimate request in the log that occurred after the last backup to be re-executed. In selective replay Shuttle only re-executes the legitimate requests that affect the data that was modified by the attack.

2) Rectify: Shuttle requires some modifications to the source code of the application in order to recover from intrusions. In some scenarios, this is not possible, e.g., when the web application was developed by a third party. An intrusion recovery system that solves this problem is *Rectify* [49]. Like Shuttle, Rectify was designed to recover web applications that are deployed in a PaaS. Rectify uses machine learning models to correlate the HTTP requests with the corresponding database statements.

Rectify works in three distinct phases: *learning* phase, *normal* phase and *recovery* phase. In the learning phase Rectify executes endpoints of the web application and trains the machine learning models. In the normal phase Rectify collects every HTTP request and database statement that gets executed by the application and logs them in two distinct logs. In the recovery phase Rectify uses the machine learning models to find the database statements that were caused by the malicious request and executes the compensating operations to undo the damage caused by the attack.

3) Sanare: Sanare was designed to recover web applications that use more than one repository to keep their state. Figure 5 represents a web application that uses several data repositories to keep its state: one or more databases, a file system, cloud storage, and external web services.

Sanare logs the operations that are executed in the different data repositories with agents that understand the semantics of the data repository to which they are attached. These agents filter and log every operation that gets executed in the log.

Matchare, the algorithm that is responsible for finding the operations that are caused by an HTTP request, uses several Deep Convolutional Neural Networks [50], to find matches between HTTP requests and database statements, file system operations, and web services requests. Matchare needs to train one model for each data repository. This is required because each data repository has its own semantic and features.

4) MIRES: The previous works presented in this section assume that the application is based on the Platform-as-a-Service model. This requires application developers to setup the database and servers hosting the application. There is an alternative computing model called Backend-as-as-Service (BaaS) [51] that allows developers to access several backend services in a cloud, such as user authentication, database, push notifications, and storage. This gives the benefits of auto scaling to cope with unpredictable demands and requiring less development effort. For this computing model, MIRES [52] is an intrusion recovery system designed for applications that use a BaaS to store the state.

MIRES provides two different types of recovery: *administrator recovery*, which allows a system administrator to undo any transaction in the system; and *user recovery*, that allows a user to revert an operation he has executed in the application. During recovery, MIRES locks the database to any write operation, only allowing read operations. The recovery process starts by finding dependencies, i.e., every operation that reads data that is created by a malicious operation. Once the dependencies are collected, MIRES unlocks the database and reconstructs the tampered documents. It does this by executing a *focused recovery* algorithm, similar to the one used in NoSQL Undo [43]. During recovery, the users can interact with the application except access the documents that are being reconstructed.

VIII. Experimental results

The intrusion recovery systems discussed in this paper are evaluated by executing a workload in a prototype. There are some metrics that are commonly collected in the different systems. In our research, we noticed that, for almost every intrusion recovery system, the authors calculate three metrics: a) performance overhead, b) Mean Time to Recover (MTTR), and c) storage overhead. Using these metrics, our aim is to compare the systems discussed and answer the research question RQ5 presented in Section II-A. In the following sections we present the experimental results of the works discussed in this paper group using these three metrics.

A. Performance overhead

The performance overhead of an intrusion recovery system consists of the downgrade, performance-wise, of intercepting and/or logging the operations for future recovery purposes. In Table II we present a collection of some performance values we collected from the papers discussed. The overhead was calculated as a percentage range relative to the decreased performance (T) or additional latency (L). We were able to collect these values for every paper discussed, except for [19] which did not include these numbers in their experimental evaluation. Also, for [15] the authors did not include the normal execution time for comparison, making it impossible

to compute a percentage. For the remaining systems, there is a performance overhead that varies from 0% (negligible) to 500%. This disparity in these values is related to the target systems for which these intrusion recovery mechanisms are aimed. For example, the largest overhead is in the file system group because the intrusion recovery mechanisms interfere with the file system to append the required meta-data for future recovery. Some systems, such as [16], [27] provide an overhead close to 0% due to some techniques, such as parallel logging, that allow the application to serve the clients concurrently with the logging mechanism.

TABLE II: Performance overhead of the different recovery approaches. The overhead is in percentage range and it refers to the decreased performance (T) or additional latency (L) for each request.

System	Target system	Overhead
Operator Undo [15]	Email systems	62ms - 484ms (L)
Backtracking [27]	VMs	0% - 9% (L)
Bezoar [28]	VMs	40%(T)
SHELF [29]	VMs	7.5% - 65% (L) / 12.4% - 21.4% (T)
EFS [30]	File systems	1,5% - 24% (L)
S4 [31]	File systems	1% - 3% (T)
RFS [32]	File systems	4% - 5% (T)
Taser [16]	File systems	0.6% - 7.4% (T)
Back to the Future [53]	File systems	12% - 52%(L)
Solitude [38]	File systems	1% - 500% (L)
Retro [39]	File systems	53% - 227%(T)
RockFS [33]	Cloud FSs	11% - 26% (L)
Amman et al. [19]	DBs	-
NoSQL Undo [43]	NoSQL DBs	6%-8% / 20%30% (T)
Akkus et al. [41]	Web apps	3.99% / 4.12% (T)
Warp [44]	Web apps	24% - 27% (T)
Aire [45]	Web apps	18,5% - 30,35% (T)
Shuttle [48]	Web apps	13% - 16% (T)
Sanare [54]	Web apps	12% - 17% (T)
Rectify [49]	Web apps	14% - 18% (T)
MIRES [52]	BaaS	15% - 23% (L)

B. Mean Time to Recover

The Mean Time to Recover (MTTR) is the average time it takes to recover a set of one or more operations. In Table III we present the collected MTTR values from the papers discussed. As we can see in the table, this metric is not as broadly calculated as the performance overhead. This is explained by the fact that the recovery process differs for the different target systems. By analyzing the values in the table it is clear that it is not possible to compare the MTTR between the different recovery mechanisms. Some papers calculate this metric taking into account the number of operations that are being reverted, while others calculate the time it takes to revert a set of files. Another characteristic that hampers the comparison is that each system is evaluated with a different scale. For example, some systems [16], [32] perform this experiment in a time frame (one day of system logs) while other systems [39], [43]–[45], [48], [49], [52], [54] perform this experiment with a set of operations. In this latter group we calculated the MTTR for a single operations (last column of the table); this allows for some comparison between the different system (although not a fair comparison since there are several variables to take into account such as, computing power, cold start delay that affects the calculation and other aspects related with the target systems).

TABLE III: MTTR of the different recovery approaches.

System	Target system	MTTR (batch)	Batch size	MTTR (unit)
Operator Undo [15]	Email systems	590s (rewind)	10,000 (users)	-
Backtracking [27]	VMs	-	-	-
Bezoar [28]	VMs	-	-	-
SHELF [29]	VMs	-	-	-
EFS [30]	File systems	-	-	-
S4 [31]	File systems	-	-	-
RFS [32]	File systems	9m - 20m	1 day	-
Taser [16]	File systems	20s - 330s	1 day	-
Back to the Future [53]	File systems	-	-	-
Solitude [38]	File systems	-	-	-
Retro [39]	File systems	4.7s	10,000 (ops)	0.3s
RockFS [33]	Cloud FSs	40s	100 (files)	2s
Amman et al. [19]	DBs	-	-	-
NoSQL Undo [43]	NoSQL DBs	150s - 200s	10,000 (ops)	1s / 700s
Akkus et al. [41]	Web apps	-	-	-
Warp [44]	Web apps	3,538s	2,093	1.69s
Aire [45]	Web apps	84.06s	5,444	16ms
Shuttle [48]	Web apps	544s - 1,717s	1,000,000	0.5ms - 1.7ms
Sanare [54]	Web apps	90 - 340s	10 - 60	1.8s - 6s
Rectify [49]	Web apps	960s	1,000	12s
MIRES [52]	BaaS	55s	1,000	1s

IX. Related Approaches

This section mentions two different areas of research that are, in some sense, related to intrusion recovery.

A related research line is often designated *reversible computing* [55]–[58]. This area focuses on the study of invertible primitives and physical reversibility. Some examples of applications in this field of study involve reversible logic circuits, reversible Turing machines, and reversible cellular automata. Although this field explores the reversibility of computing operations, such mechanisms are not suitable for the computing models approached in this survey, i.e., file systems, databases, web applications, and the cloud. These works do not aim to revert the corrupt state; instead, they focus on generating reversible operations. Given this difference, these works are not discussed in this survey.

Selective re-execution reverts a system state to a previous point in time and reconstructs it by executing the legitimate

TABLE IV: Storage overhead of saving the log entries for the executed operations and relevant metadata for recovery in the different approaches

System	Target system	Overhead (batch)	Batch size	Overhead (unit)
Operator Undo [15]	Email systems	206.5MB	10,000 users 30 minutes	- -
Backtracking [27]	VMs	0.002GB - 1.2GB	1 day	-
Bezoar [27]	VMs	-	-	-
SHELF [29]	VMs	82.7MB 541MB	133,308 1,344,712 events	- -
EFS [30]	File systems	-	-	-
S4 [31]	File systems	10GB	50 - 470 days	-
RFS [32]	File systems	260MB	1,863,971 reqs	-
Taser [16]	File systems	1.9GB - 2.3GB	1 day	-
Back to the Future [53]	File systems	6KB - 12552KB	1 file	
Solitude [38]	File systems	30,8GB / 26.1GB	10 / 14 days	-
Retro [39]	File systems	100GB - 150GB	1 day	-
RockFS [33]	Cloud FSs	100%	1 file	100%
Amman et al. [19]	DBs	-	-	-
NoSQL Undo [43]	NoSQL DBs	100MB - 120MB	6,000 ops	16.6KB - 20KB
Akkus et al. [41]	Web apps	-	-	4KB
Warp [44]	Web apps	-	1	11.05KB
Aire [45]	Web apps	-	1	5.52KB - 9,24KB
Shuttle [48]	Web apps	10,684MB	1,000,000 reqs	106,84KB
Sanare [54]	Web apps	17.38GB - 20.24	1,000,000 reqs	17.38KB - 20.24KB
Rectify [49]	Web apps	5.13GB - 8.20GB	1,000,000 reqs	5.13KB - 8.20KB
MIRES [52]	BaaS	0.11GB - 0.41GB	1,000,000 ops	-

operations. This approach resembles a line of work known by *operational transformation* (OT) [59]–[62]. This technique aims to maintain data consistency in collaborative applications, allowing several users to simultaneously work on common data records using distributed computers connected through the network. This technique allows, among other operations, to update, delete, lock edits, and undo operations. The undo operation has a similar goal to the one used to recover from intrusions, namely to revert the effects of a previous operation. However, when intrusions occur, they may propagate their effects to different data records, requiring a more delicate undo operation capable of tracking and reverting every data record. Again, this is related to the topic of the survey, but different.

X. Conclusion

In this article we present a literature survey on intrusion recovery solutions for services and applications that are able to preserve the legitimate state changes done since the last backup/checkpoint. This notion was introduced circa 2000 in the context of the three R's approach [22] and Operator Undo [15]. We present the two decades of work that followed, considering the original approach, virtualization, storage systems, and web applications.

Table V summarizes the recovery strategies adopted by each recovery system. The intrusion recovery mechanisms appear in the first column, the target system for the recovery mechanism is in the second column, the third, fourth and fifth columns correspond to the adopted recovery strategy employed by the mechanism. The sixth column presents how the mechanism deals with external inconsistencies observed by the user, the seventh column indicates if the mechanism is capable of recovering the system without requiring it to be offline, and, finally, the last column indicates if the intrusion recovery system is capable of starting a recovery process automatically when it detects an intrusion.

References

[1] A. Avizienis, J. C. Laprie, B. Randell, and C. Landwehr, "Basic concepts and taxonomy of dependable and secure computing," *IEEE Transactions on Dependable and Secure Computing*, vol. 1, no. 1, pp. 11–33, Jan 2004.

[2] B. Cheswick, "The design of a secure internet gateway," in *USENIX Summer Conference Proceedings*, 1990.

[3] A. Wool, "A quantitative study of firewall configuration errors," *Computer*, vol. 37, no. 6, pp. 62–67, 2004.

[4] R. S. Sandhu and P. Samarati, "Access control: principle and practice," *IEEE Communication Magazine*, vol. 32, no. 9, pp. 40–48, 1994.

[5] M. Gasser, *Building a Secure Computer System*. Van Nostrand Reinhold, 1988.

[6] D. E. Denning and P. G. Neumann, "Requirements and model for IDES - a real-time intrusion detection expert system," Computer Science Laboratory, SRI International, Menlo Park, CA, Tech. Rep., 1985.

[7] H. Debar, M. Dacier, and A. Wespi, "Towards a taxonomy of intrusion detection systems," *Computer Networks*, vol. 31, no. 8, pp. 805–822, Apr. 1999.

[8] D. R. Miller, S. Harris, A. A. Harper, S. VanDyke, and C. Blask, "Security information and event management (SIEM) implementation (Network Pro Library)," 2010.

[9] "OWASP Top 10 2021," https://owasp.org/Top10/, 2021.

[10] S. Mohurle and M. Patil, "A brief study of Wannacry threat: Ransomware attack 2017," in *International Journal of Advanced Research in Computer Science*, vol. 8, no. 5, 2017.

[11] Varonis. (Nov. 2018) 60 Must-Know Cybersecurity Statistics for 2018. https://www.varonis.com/blog/cybersecurity-statistics/. Accessed: Jul. 27, 2021.

[12] Info Security. (2018) 2017: Worst Year Ever for Data Loss and Breaches. https://www.infosecurity-magazine.com/news/2017-worst-year-ever-for-data-loss/.

[13] P. Cichonski, T. Millar, T. Grance, K. Scarfone *et al.*, "Computer security incident handling guide," *NIST Special Publication*, vol. 800, no. 61, pp. 1–147, 2012.

[14] M. J. West-Brown, D. Stikvoort, K.-P. Kossakowski, G. Killcrece, and R. Ruefle, "Handbook for computer security incident response teams (csirts)," Carnegie Mellon Software Engineering Institute, Tech. Rep., 2003.

[15] A. Brown and D. Patterson, "Undo for operators: Building an undoable e-mail store," in *Proceedings of the USENIX Annual Technical Conference*, 2003, pp. 1–14.

[16] A. Goel, K. Po, K. Farhadi, Z. Li, and E. De Lara, "The Taser intrusion recovery system," in *Proceedings of the 20th ACM Symposium on Operating Systems Principles*, vol. 39 (5), 2005, pp. 163–176.

[17] H. F. Korth, E. Levy, and A. Silberschatz, "A formal approach to recovery by compensating transactions," in *Proceedings of the 16th International Conference on Very Large Data Bases*, 1990, pp. 95–106.

[18] P. A. Bernstein, V. Hadzilacos, and N. Goodman, *Concurrency control and recovery in database systems*. Addison-Wesley Pub. Co. Inc., Reading, MA, 1987.

[19] P. Ammann, S. Jajodia, and P. Liu, "Recovery from malicious transactions," *IEEE Transactions on Knowledge and Data Engineering*, vol. 14, no. 5, pp. 1167–1185, 2002.

[20] E. N. Elnozahy, L. Alvisi, Y.-M. Wang, and D. B. Johnson, "A survey of rollback-recovery protocols in message-passing systems," *ACM Computing Surveys*, vol. 34, no. 3, pp. 375–408, 2002.

[21] B. Kitchenham, "Procedure for undertaking systematic reviews," *Computer Science Depart-ment, Keele University (TRISE-0401) and National ICT Australia Ltd (0400011T. 1), Joint Technical Report*, vol. 33, 2004.

[22] A. B. Brown and D. A. Patterson, "Rewind, repair, replay: three r's to dependability," in *Proceedings of the 10th ACM SIGOPS European Workshop*, 2002, pp. 70–77.

[23] A. Brown and A. A. Patterson, "To err is human," in *Proceedings of the First Workshop on Evaluating and Architecting System Dependability (EASY'01)*, 2001.

[24] P. Enriquez, A. Brown, and D. A. Patterson, "Lessons from the PSTN for dependable computing," in *Workshop on Self-Healing, Adaptive and Self-Managed Systems*, 2002.

[25] A. Oppenheimer, A. Ganapathi, and A. A. Patterson, "Why do internet services fail, and what can be done about it?" in *Proceedings of the 4th Conference on USENIX Symposium on Internet Technologies and Systems - Volume 4*, 2003.

[26] S. A. Zonouz, H. Khurana, W. H. Sanders, and T. M. Yardley, "RRE: a game-theoretic intrusion response and recovery engine," *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, no. 2, pp. 395–406, 2013.

[27] S. T. King and P. M. Chen, "Backtracking intrusions," in *Proceedings of the 19th ACM symposium on Operating System Principles*, 2003, pp. 223–236.

[28] D. Oliveira, J. R. Crandall, G. Wassermann, S. Ye, S. F. Wu, Z. Su, and F. T. Chong, "Bezoar: Automated virtual machine-based full-system recovery from control-flow hijacking attacks," in *Proceedings of the IEEE Network Operations and Management Symposium*, 2008, pp. 121–128.

[29] X. Xiong, X. Jia, and P. Liu, "Shelf: Preserving business continuity and availability in an intrusion recovery system," in *Proceedings of the Annual Computer Security Applications Conference*, 2009, pp. 484–493.

[30] D. S. Santry, M. J. Feeley, N. C. Hutchinson, A. C. Veitch, R. W. Carton, and J. Ofir, "Deciding when to forget in the Elephant file system," in *Proceedings of ACM SIGOPS Symposium on Operating Systems Principles*, 1999, pp. 110–123.

[31] J. D. Strunk, G. R. Goodson, M. L. Scheinholtz, C. A. N. Soules, and G. R. Ganger, "Self-securing storage: protecting data in compromised system," in *OSDI'00: Proceedings of the 4th conference on Symposium on Operating System Design & Implementation*, vol. 4. USENIX, 2000.

[32] N. Zhu and T.-c. Chiueh, "Design, implementation, and evaluation of repairable file service," in *Proceedings of the International Conference on Dependable Systems and Networks*, 2003.

[33] D. R. Matos, M. L. Pardal, and M. Correia, "RockFS: Cloud-backed file system resilience to client-side," in *Proceedings of the 2018 ACM/IFIP/USENIX International Middleware Conference*, 2018, p. 107–119.

TABLE V: Recovery strategies for each system.

Recovery System	Target	Selective re-execution	Compensating operations	Multiversioned	External inconsistencies	Online recovery	Real-Time Recovery
Operator Undo [15]	Email systems	X			Compensating operations	No	No
Backtracking [27]	VMs	X			Not mentioned	No	No
Bezoar [28]	VMs	X			Not mentioned	No	Yes
SHELF [29]	VMs	X			Not mentioned	Yes	Yes
EFS [30].	File systems			X	Not mentioned	Yes	No
S4 [31]	File systems			X	Not mentioned	Yes	No
RFS [32]	File systems		X		Not mentioned	Yes	No
Taser [16]	File systems	X			Compensating operations	No	Yes
Back to the Future [53]	File systems	X			Not mentioned	No	Yes
Solitude [38]	File systems	X			Compensating operations	No	No
Retro [39]	File systems	X			Compensating operations	No	No
RockFS [33]	Cloud FSs	X	X	X	Compensating operations	Yes	No
Amman et al. [19]	DBs		X		Not mentioned	Yes	No
NoSQL Undo [43]	NoSQL DBs	X	X		Compensating operations	Yes	Yes
Akkuş et al. [41]	Web apps		X		Not mentioned	No	No
Warp [44]	Web apps	X			Compensating operations	Yes	No
AIRE [45]	Web apps		X		Parallel recovery (like Git)	Yes	No
μVerum [46]	Web apps		X		Compensating operations	Yes	No
Shuttle [48]	Web apps	X	X		Compensating operations	Yes	No
Sanare [54]	Web apps	X	X	X	Compensating operations	Yes	No
Rectify [49]	Web apps	X	X		Compensating operations	Yes	No
MIRES [52]	BaaS	X	X		Compensating operations	Yes	No

[34] A. N. Bessani, M. Correia, B. Quaresma, F. André, and P. Sousa, "DepSky: dependable and secure storage in a cloud-of-clouds," *EuroSys'11 Proceedings of the 6th Conference on Computer Systems*, pp. 31–46, 2011.

[35] H. Krawczyk, "Secret sharing made short," *Proceedings of the 13th International Cryptology Conference – CRYPTO'93*, pp. 136–146, 1993.

[36] J. S. Plank, S. Simmerman, and C. D. Schuman, "Jerasure: A library in C/C++ facilitating erasure coding for storage applications-version 1.2," *University of Tennessee, Tech. Rep. CS-08-627*, vol. 23, 2008.

[37] D. Ma and G. Tsudik, "A new approach to secure logging," *ACM Transactions on Storage*, vol. 5, no. 1, pp. 1–21, 2009.

[38] S. Jain, F. Shafique, V. Djeric, and A. Goel, "Application-level isolation and recovery with solitude," in *Proceedings of the 3rd ACM SIGOPS/EuroSys European Conference on Computer Systems*, 2008, pp. 95–107.

[39] T. Kim, X. Wang, N. Zeldovich, and M. F. Kaashoek, "Intrusion recovery using selective re-execution," in *Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation*, 2010, pp. 89–104.

[40] A. Brazell, *WordPress Bible*. John Wiley and Sons, 2011.

[41] İ. E. Akkuş and A. Goel, "Data recovery for web applications," in *Proceedings of the 40th IEEE/IFIP International Conference on Dependable Systems and Networks*, 2010, pp. 81–90.

[42] J. Lieponienė, "Recent trends in database technology," *Baltic Journal of Modern Computing*, vol. 8, no. 4, pp. 551–559, 2020.

[43] D. R. Matos and M. Correia, "NoSQL Undo: Recovering NoSQL databases by undoing operations," in *Proceedings of the 15th IEEE International Symposium on Network Computing and Applications*, 2016.

[44] R. Chandra, T. Kim, M. Shah, N. Narula, and N. Zeldovich, "Intrusion recovery for database-backed web applications," in *Proceedings of the 23rd ACM Symposium on Operating Systems Principles*, 2011, pp. 101–114.

[45] R. Chandra, T. Kim, and N. Zeldovich, "Asynchronous intrusion recovery for interconnected web services," in *Proceedings of the 24th ACM Symposium on Operating Systems Principles*, 2013, pp. 213–227.

[46] D. R. Matos, M. L. Pardal, A. R. Silva, and M. Correia, "μverum: Intrusion recovery for microservice applications," *IEEE Access*, 2023.

[47] J. Geelan *et al.*, "Twenty-one experts define cloud computing," *Cloud Computing Journal*, vol. 4, pp. 1–5, 2009.

[48] D. Nascimento and M. Correia, "Shuttle: Intrusion recovery for PaaS," in *Proceedings of the 35th IEEE International Conference on Distributed Computing Systems*, 2015, pp. 653–663.

[49] D. R. Matos, M. L. Pardal, and M. Correia, "Rectify: Black-box intrusion recovery in PaaS clouds," in *Proceedings of the 2017 ACM/IFIP/USENIX International Middleware Conference*, 2017, p. 209–221.

[50] H. Lee, R. Grosse, R. Ranganath, and A. Y. Ng, "Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations," in *Proceedings of the 26th annual international conference on machine learning*, 2009, pp. 609–616.

[51] K. Lane, "Overview of the Backend as a Service (BaaS) space White Paper," in *API Evangelist*, 2015.

[52] D. Vaz, D. Matos, M. Pardal, and M. Correia, "MIRES: Intrusion recovery for applications based on backend-as-a-service," *IEEE Transactions on Cloud Computing*, pp. 1–1, 2022.

[53] F. Hsu, H. Chen, T. Ristenpart, J. Li, and Z. Su, "Back to the future: A framework for automatic malware removal and system repair," in *Proceedings of the 22nd Annual Computer Security Applications Conference*, 2006, pp. 257–268.

[54] D. Matos, M. Pardal, and M. Correia, "Sanare: Pluggable intrusion recovery for web applications," *IEEE Transactions on Dependable and Secure Computing*, pp. 1–1, 2021.

[55] T. Toffoli, "Reversible computing," in *International Colloquium on Automata, Languages, and Programming*. Springer, 1980, pp. 632–644.

[56] M. P. Frank, "Introduction to reversible computing: motivation, progress, and challenges," in *Proceedings of the 2nd ACM Conference on Computing Frontiers*, 2005, pp. 385–390.

[57] K. Morita, "Reversible computing and cellular automata—a survey," *Theoretical Computer Science*, vol. 395, no. 1, pp. 101–131, 2008.

[58] A. De Vos, *Reversible computing: fundamentals, quantum computing, and applications*. John Wiley & Sons, 2011.

[59] C. Sun and C. Ellis, "Operational transformation in real-time group editors: issues, algorithms, and achievements," in *Proceedings of the ACM Conference on Computer Supported Cooperative Work*, 1998, pp. 59–68.

[60] D. Sun, A. Xia, C. Sun, and D. Chen, "Operational transformation for collaborative word processing," in *Proceedings of the ACM Conference on Computer Supported Cooperative Work*, 2004, pp. 437–446.

[61] A. H. Davis, C. Sun, and J. Lu, "Generalizing operational transformation to the standard general markup language," in *Proceedings of the 2002 ACM Conference on Computer Supported Cooperative Work*, 2002, pp. 58–67.

[62] D. Sun and C. Sun, "Context-based operational transformation in distributed collaborative editing systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 20, no. 10, pp. 1454–1470, 2009.