RectifyPlus

Black-Box Intrusion Recovery in MEAN Applications

Rui Miguel Pereira Barata

Thesis to obtain the Master of Science Degree in
Information Systems and Computer Engineering

Supervisors: Prof. Miguel Filipe Leitão Pardal
Prof. Miguel Nuno Dias Alves Pupo Correia

Examination Committee

Chairperson: Prof. José Carlos Martins Delgado
Supervisor: Prof. Miguel Filipe Leitão Pardal
Member of the Committee: Prof. Rui Filipe Lima Maranhão de Abreu

October 2019
Acknowledgments

I would like to thank my parents for their friendship, encouragement and caring over all these years, for always being there for me and without whom this project would not be possible.

To my sister, Marta, I would like to thank for her support throughout this work.

To my second family, Tuna Mista do Instituto Superior Técnico (TMIST), I would like to thank all the nights of party and fun.

To my friends for all the advice and for keeping me sane.

To my supervisors, Miguel Pardal and Miguel Correia, I would like to thank, in the first place, for trusting in me for this work, and for all the patience, advices, guidance and constant encouragement throughout the course of this thesis.

To David Matos for all the guidance, motivation and advices throughout this thesis.

Thank you.
Abstract

Cloud computing makes provisioning, scaling, and maintenance of web applications easy. All these benefits are leading to a growing number of web applications hosted in the cloud. A very common application architecture is called MEAN, a Javascript full stack framework that uses MongoDB, Express.js, AngularJS and NodeJS. Because these applications are exposed on the Internet, they are prone to attacks. Even with good defenses, it is possible that some attacks are effective and execute malicious requests. A recovery process is needed to remove the effects of the attack on the state of a system. Rectify is a tool that allows the application’s administrator to recover from attacks by executing compensation operations. In this work, we introduce RectifyPlus an evolution of the Rectify web application intrusion recovery tool, furthering its scope, giving it the ability to recover web applications written with the MEAN stack. It does not require modifications to the protected application source code and the recovery can be performed by a system administrator. Machine learning techniques, more specifically the naïve Bayes classifier, are used to associate faulty HTTP requests to their corresponding database statements. RectifyPlus was evaluated using the MEANie content management system (CMS) and the results show that RectifyPlus is able to recover from intrusions effects whilst preserving the valid application data.

Keywords

Intrusion Removal, Intrusion Recovery, Rollback, Recovery, MEAN Web Applications, AngularJS, MongoDB
Resumo

A computação na nuvem faz com que o provisionamento, o escalonamento e a manutenção de aplicações seja fácil. Todos estes benefícios estão a fazer com que o número de aplicações na cloud esteja a aumentar. Uma das arquiteturas mais conhecidas de aplicações é a MEAN que é uma arquitetura de software Javascript que usa MongoDB, Express, AngularJS e Node.js. Por estarem expostas na Internet, estas aplicações estão susceptíveis a serem alvos de ataques e mesmo com boas medidas de segurança informática é possível que alguns ataques sejam eficazes e pedidos maliciosos sejam executados na aplicação. Para remover os efeitos dos pedidos maliciosos é necessário executar um processo de recuperação. Rectify é uma ferramenta que permite recuperar uma aplicação dos efeitos de um ataque através de operações de compensação com a intervenção do administrador do sistema. Neste trabalho nós propomos o RectifyPlyus, uma evolução da ferramenta de recuperação de intrusões para aplicações Rectify aumentando o seu campo de acção, dando-lhe a habilidade de recuperar aplicações que foram escritas com recurso à arquitetura MEAN. O RectifyPlus não requer alterações ao código fonte da aplicação alvo e a recuperação da aplicação pode ser feita por um administrador do sistema. O RectifyPlus usa técnicas de aprendizagem automática, mais especificamente o classificador naïve Bayes, para associar os pedidos HTTP maliciosos com os pedidos à base de dados correspondentes. O RectifyPlus foi avaliado usando o sistema de gestão de conteúdos MEANie e os resultados mostram que o RectifyPlus é capaz de recuperar a aplicação alvo dos efeitos de intrusões preservando os dados da aplicação.

Palavras Chave

Remoção de Intrusões, Recuperação de Intrusões, Rollback, Recuperação, Aplicações MEAN, AngularJS, MongoDB
# Contents

1 Introduction 1
   1.1 Problem Description .................................................. 3
   1.2 Approach and Solution ............................................... 4
   1.3 Contributions ......................................................... 4
   1.4 Document outline ..................................................... 4

2 Background 7
   2.1 Dependability ............................................................ 9
      2.1.1 Dependability means ............................................. 10
   2.2 Cloud Computing ....................................................... 12
   2.3 Representational State Transfer (REST) ............................. 13
   2.4 JavaScript Object Notation (JSON) ................................ 14
   2.5 AngularJS ................................................................. 14
   2.6 MEAN stack ............................................................... 16
      2.6.1 MongoDB ............................................................ 16
      2.6.2 Express.js ......................................................... 17
      2.6.3 Node.js ............................................................ 17
   2.7 Machine Learning ....................................................... 17
   2.8 WEKA ....................................................................... 18
   2.9 Summary ................................................................. 19

3 Related Work 21
   3.1 Intrusion Recovery .................................................... 23
   3.2 Recovery at Application Level ....................................... 23
   3.3 Comparison ............................................................. 27

4 RectifyPlus 29
   4.1 PaaS Architecture ..................................................... 31
   4.2 Architecture ............................................................ 32
   4.3 RectifyPlus Phases ................................................... 33
List of Figures

2.1 Fault Tolerance Techniques. Adapted from [1] ................................. 11
2.2 JSON Object Example ................................................................. 14
2.3 Overview of MEAN stack ............................................................... 17

4.1 Typical PaaS Architecture. Adapted from [2]. .............................. 31
4.2 RectifyPlus system architecture. RectifyPlus itself is the set of components greyed out.
   Adapted from [2] .................. ............................................. 32
4.3 Stored HTTP Request example ..................................................... 34
4.4 Signature Record example ........................................................... 35
4.5 Example of an ARFF file header created by RectifyPlus .................. 36
4.6 StringToNominal filter application example ................................. 37
4.7 Data flow diagram of the generic tasks performed to recover the protected application.
   Diagram adapted from [2] ....................................................... 38
4.8 Data flow diagram with the tasks performed to identify the database statements issued by
   a malicious HTTP request. Diagram adapted from [2] ...................... 39

5.1 Performance overhead of using RectifyPlus with MEANie application measured in re-
   quests per second ................................................................. 44
5.2 Total time to recover the protected application ................................ 45
List of Tables

2.1 Weather data file example from Weka ............................................. 18
3.1 State recovery extended from Shuttle [3] ........................................... 28
Introduction

Contents

1.1 Problem Description ................................................. 3
1.2 Approach and Solution ............................................. 4
1.3 Contributions ....................................................... 4
1.4 Document outline .................................................. 4
Cloud computing is transforming the IT industry [4], making software even more attractive as a service and shaping the way Information Technology (IT) hardware is designed and purchased. The appearance of almost unlimited computing resources available on demand and the ability to pay-per-use of computing resources on a short-term basis are compelling characteristics for customers of information technology. Cloud computing services provide the resources that customers need, eliminating the need to plan far ahead for provisioning.

In this work we will focus on the PaaS (platform-as-a-service) model. In a PaaS system, the administrator is able to maintain and configure complex applications without the burden of managing the full software stack in a set of servers. It provides a set of programming and middleware services to support application design, implementation and maintenance. Examples of these services are load-balancing and automatic server configuration. The load-balancing service automatically improves the distribution of workloads across multiple computing resources. This allows having stable throughput even when dealing with high peaks of traffic. The PaaS model provides more versatility to run user applications than the SaaS model, and easier management than the IaaS model. Some well-known cloud providers that provide PaaS systems are: Amazon (Amazon AWS), Microsoft (Windows Azure), Google (Google Cloud Platform) and Force.com.

Given this reality, there have been popular implementation frameworks that simplify application development assuming that they will be deployed in a PaaS. One of the popular combinations is called the MEAN architecture. MEAN is a full-stack javascript framework that stands for MongoDB, Express, AngularJS and Node.js, each one providing a letter to the acronym.

1.1 Problem Description

Many customers and companies are migrating their applications and valuable information to cloud environments, leading to a large number of critical and complex applications deployed in the cloud to increase rapidly. Despite its advantages, cloud computing is inherently more risky because physical hardware is shared between customers of the cloud provider, called tenants. As a result, the risk of intrusion is higher because the exploitation of vulnerabilities is more attractive and profitable. PaaS offerings allow developers and system administrators to focus on the application. As result, those applications are prone to have implementation and configuration vulnerabilities introduced, involuntarily, by developers and system administrators. Moreover, web applications are known to be prone to many security risks [5] that can compromise the integrity and the state of the database. When a vulnerability is exploited and the state of a web application is illegitimately modified, the system administrator may have to recover the application state to a previous one by rolling it back to a point in time before intrusion, e.g., by restoring a backup or by using a checkpointing mechanism. This approach removes the effect
of the intrusion, but will likely discard legitimate state modifications that occurred after the intrusion. This means that business transaction data can be lost.

Earlier work on intrusion recovery used logs and snapshots to recover from unwanted changes. The majority of the recent work done in the recovery field considers a replay phase [3, 6] where, after the recovery to a previous state, all the legitimate requests done after the date of the new state are replayed. This guarantees that the effects of an attack or illegitimate change on the state are rolled back and the legitimate requests after the attack persist in the new state.

1.2 Approach and Solution

The approach followed in this work consists in evolving the Rectify tool [2] by widening its scope. Rectify is a black-box intrusion recovery service for PaaS applications. This tool does not require modifications to the application source code. In its previous version, Rectify was able to recover 1000 malicious HTTP requests in around 16 minutes as reported in experiments performed by the creators of the tool on Wordpress and other widely used web applications. This work extended the scope of Rectify to recover more web applications, in particular those that were developed with MEAN full-stack Javascript framework. The improved tool, called RectifyPlus, is able to recover web applications that uses as communication the REST API with JSON objects and with the NoSQL database MongoDB as the persistent data store of the web application.

1.3 Contributions

To the best of our knowledge, RectifyPlus is the first intrusion recovery system for MEAN applications that does not require changes in the application code, and can recover applications that use the REST API with JSON objects as their communication system. RectifyPlus uses machine learning to be able to recover an application from a malicious HTTP request, making it possible to consider the protected application as black-box. To validate our work we tested our tool with the MEANie application, issuing 152.000 HTTP requests. We extrated results about the performance overhead, the space overhead and the total time to recover the application.

1.4 Document outline

The remainder of the document is structured as follows. Chapter 2 presents the technologies behind the problem that our proposal aims to resolve. Section 3 presents the fundamental concepts behind recovery field and previous intrusion recovery work. Section 4 describes the architecture of PaaS systems and
the RectifyPlus architecture. Section 5 defines the evaluation methods and the results to validate the proposed architecture. Finally, Section 6 concludes the document, pointing to future work directions made possible by this work.
2 Background

Contents

2.1 Dependability ...................................................... 9
2.2 Cloud Computing .................................................. 12
2.3 Representational State Transfer (REST) ........................ 13
2.4 JavaScript Object Notation (JSON) ............................ 14
2.5 AngularJS .......................................................... 14
2.6 MEAN stack ......................................................... 16
2.7 Machine Learning .................................................. 17
2.8 WEKA .............................................................. 18
2.9 Summary ........................................................... 19
This chapter explains the frameworks and the technologies behind them that are used in our work. AngularJS is a client-side JavaScript framework and it uses REST API with JSON objects to communicate with the server-side and users. AngularJS is mainly used within the MEAN stack. This stack represents the main frameworks used with AngularJS such as MongoDB for the databases and Node.js with Express develop the server-side. We start with a taxonomy of dependability in section 2.1. In section 2.2 we introduce the cloud computing fundamentals and explain the models that cloud computing providers offer. Section 2.3 explains the Representational State Transfer (REST) used by AngularJS as communication method. Section 2.4 explains the JavaScript Object Notation (JSON) used with the REST API to transfer the arguments needed by the users requests. AngularJS is explain in section 2.5, and it is the framework used to develop the client-side of the application. Finally, section 2.6 explains in more detail the MEAN stack, i.e, the frameworks and technologies used with AngularJS applications.

2.1 Dependability

Dependability is the ability to deliver a service that can justifiably be trusted [1]. This definition requires the justification of trust and what the notion of trust includes. Dependability is an integrating concept that contains the following attributes:

- **availability**: the degree to which a system is in a operable state, i.e, that is ready to use.
- **reliability**: the correctness of the service, i.e, the service ability to function correctly.
- **safety**: absence of catastrophic consequences on the user(s) and the environment.
- **integrity**: absence of improper system alterations, i.e., assurance of the accuracy and consistency of the system.
- **maintainability**: the system’s ability to undergo modifications and repairs.

Dependability impairments are usually defined in terms of faults, errors, or failures [7]; they are undesired circumstances that cause an unexpected system behavior. A common feature of the three terms is that they give the message that something went wrong. A fault is what causes an error. A fault can be related to a physical defect, imperfection, or flaw that occurs in hardware or software components, e.g., a software bug is a fault. An error is the deviation of correctness in computing, which occurs as a result of a fault. Errors are usually associated with the incorrect behavior of a system, causing incorrect values in the system state. All faults that may affect a system are classified according to eight basic viewpoints [1]. There are three major faults classes. The **development faults** class includes all faults occurring during development. The **physical faults** class include all faults that affect hardware, and the **interaction faults** classes include all external faults. A service failure is what occurs when the system
deviates from correct service, i.e., a failure occurs when a system deviates from its expected behavior in a specified period of time. The connection between these impairments is that faults are reasons for errors and errors are reasons for failures.

In this work we consider faults that are generated by humans in a malicious or non-malicious way. Software bugs are faults that can be used by attackers to interact in a malicious way with a system, causing an undesired change on the system state and behavior.

An intrusion is a malicious fault resulting from an intentional vulnerability exploitation. There are four major means to deal with faults [1] that aims to attain the various attributes of dependability (described on the beginning of this section).

2.1.1 Dependability means

Dependability means are the methods and techniques that enables the development of dependable systems. Those means are: fault prevention, fault tolerance, fault removal and fault forecasting [1].

Fault prevention deals with preventing faults being incorporated into a system, i.e., prevent the occurrence or introduction of faults. Prevention of development faults is an obvious aim for development methodologies. However, preventing development faults is difficult because software is complex and it is prone to have flaws. Besides that, administrators can make security configuration mistakes, or users may grant access to attackers. Improvement of the development process in order to reduce the number of faults introduced in the systems and the modification of systems in order to eliminate the causes of the faults are a step further on fault prevention.

Fault tolerance is the way in which a system responds to a failure. Fault tolerance techniques try to avoid service failures in the presence of faults, i.e., fault tolerance targets the development of systems which functions correctly in presence of faults. Fault tolerance is carried out via error detection and system recovery. There are a few processes that help to attain a fault tolerant system [7]:

- **Fault masking**: It is a technique to prevent error result to be visible to the user/operator. It is a process of ensuring that only correct values get passed to the system output in the presence of a fault. For example, one fault masking technique is the majority voting system. In a system with three servers running the same requests, if one of them has a fault and produces a wrong response, with the majority voting system the correct response will be delivered to the client.

- **Fault detection**: It is the process of determining that a fault has occurred within a system. One fault detection technique is acceptance tests. In this process the system is tested for a specific result. If the result is different from the expected one, the system has a fault.

- **Fault location**: It is the process of determining where a fault has occurred.
Figure 2.1: Fault Tolerance Techniques. Adapted from [1]

- **Fault containment**: Is the process of isolating a fault and preventing it to spread the effect of that fault throughout the system.

Figure 2.1 gives more detail about the techniques involved in fault tolerance. The fault tolerance techniques are split in two categories. The *error detection* category aims to identify the presence of an error. The detection can be achieve by two methods: the concurrent detection, where the system is checked while normal service is delivered, and the preemptive detection, where the normal service delivery is suspended and the system is checked for errors. The techniques corresponding to the *recovery* category aims to transform a system state that contains errors and faults into a state without errors and faults that can be activated again. Recovery is done after an error or fault detection. When an error is detected the system administrator has to recover the system state. This can be accomplished by *rollback* technique, where the system is brought back to a state prior to the error occurrence. The *fault handling* techniques aims to prevent faults from being activate again.

**Fault removal** is performed during development phase as well as during system use. During the development phase there are three steps for fault removal to be accomplished: verification, diagnosis and correction [1]. Verification is the process of checking if a system meets a set of conditions. If the conditions are not met the other two steps have to be undertaken. The system has to be diagnosed to catch the faults that prevented the verification conditions from being fulfilled, and then the proper correction to the system has to be done. During the system usage, fault removal can be done by the corrective or preventive maintenance processes. Corrective maintenance aims to remove faults that
have been reported, while preventive maintenance aims to remove faults before they cause errors during normal operation.

Fault forecasting is done by performing an evaluation of the system behavior in respect to fault occurrences or activation. The evaluation can be qualitative, aiming to identify and classify the failure modes or event combinations that lead to system failure, or quantitative, that aims to evaluate in terms of probabilities the extent to which some attributes of dependability are satisfied. Cloud computing is based in five essential characteristics.

2.2 Cloud Computing

Cloud computing is a model for enabling ubiquitous, convenient, on-demand access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) and higher-level services over the Internet [8]. Those resources can be released and provisioned with minimal management effort or service provider interaction.

1. On-demand self-service - Computer services that do not require human interaction with each service provider, i.e., cloud resources are provisioned on demand whenever they are required without the need of interaction with the cloud provider employees. This services are accessed through an online control panel. Those services can be email, applications or server services. Cloud on-demand self-service providers include Amazon, Microsoft, Google and IBM.

2. Broad network access - Cloud capabilities are available through standard mechanisms by client platforms such as mobile phones and laptops.

3. Resource pooling - The providers’ computing resources are pooled to serve multiple consumers using multiple-tenant model. The resources are dynamically assigned to consumers according to their demand.

4. Rapid elasticity - Cloud services can be provisioned rapidly and elastically to scale according to the consumer demand.

5. Measured service - Cloud computing resource usage can be measured, controlled, and reported for both the provider and consumer of the service.

Cloud computing providers offer their services according to different models. There are three cloud computing standard models [8].

Software as a Service (SaaS): The capability provided to the consumer is to use the provider’s applications running on a cloud infrastructure. The applications are accessible from various client devices
through either a client interface or a program interface. The provider is responsible for all the structure that the client needs to use the software (servers, connectivity, safety of the service).

**Platform as a Service (PaaS):** Provides the consumer the capability to deploy onto the cloud infrastructure their own applications without the concern of manage or control the underlying cloud infrastructure including network, servers, or storage. This means that the consumer are provided with a platform and environment that allow developers build applications and services over the Internet without the costly hardware required to deploy their services.

**Infrastructure as a Service (IaaS):** Provides the consumer the capability to provision processing, storage, networks, and other fundamental computing resources where the consumer can deploy and run any software. The consumer does not manage or control the underlying cloud infrastructure but has control over operating systems, storage, and deployed applications.

Our work focuses on the Platform as a Service (PaaS) model.

### 2.3 Representational State Transfer (REST)

Representational State Transfer (REST) [9] is an idealized model of the interactions within an overall web application. And it became the foundation for the modern web architecture, providing guiding principles for exposing data and functionality to remote clients. This model is composed by architectural constraints that aims to minimize latency and network communication, while maximizing the independence and scalability of web applications components implementation.

REST distinguishes three classes of architectural elements [10]:

- **Data elements** - The key abstraction of information in REST is a resource. A resource is a conceptual mapping to a set of entities. The semantic of a resource have to be static but the entities can change over time. Any information that can be named can be a resource, e.g., a document or image, a collection of other resources. This concept allows to reference a concept instead of a single representation. Is the use of resource identifiers that allows to distinguish between resources. An example of a resource identifier is the Uniform Resource Identifier (URI). REST components perform actions with a resource by using a representation to capture the current or intended state of that resource and transferring that representation between components.

- **Connectors** - These manage the network communication for a component. They encapsulate the activities of accessing resources and transferring resource representations.

- **Components** - REST components are identified by their role within an application.

REST’s primary goal is to allow Internet-scale systems by reducing network latency with caching and reducing server load by omitting session states.
2.4 JavaScript Object Notation (JSON)

JavaScript Object Notation (JSON) [11] is a lightweight data interchange format that uses human-readable text to transmit data objects consisting of attribute-value pairs and array data types. JSON is derived from the ECMAScript Programming Language Standard and it defines a set of formatting rules for portable representation of structured data.

JSON is built on two structures:

- A collection of name/value pairs (object, record, struct).
- An ordered list of values (array, vector, list).

There are four primitive types (strings, numbers, booleans, and null) and two structured types (objects and arrays) that can be represent by JSON. Figure 2.2 shows an example of an JSON object that is composed by two objects (Image member and the Thumbnail member) and an array (IDs member).

```
{
    "Image": {
        "Width": 800,
        "Height": 600,
        "Title": "View from 15th Floor",
        "Thumbnail": {
            "Url": "http://www.example.com/image/481989943",
            "Height": 125,
            "Width": "100"
        },
        "IDs": [116, 943, 234, 38793]
    }
}
```

*Figure 2.2: JSON Object Example*

Normally JSON is compared to XML since both are a text-based data-interchange format. In comparison, what makes JSON the adopted format to many developers are the benefits and the easy usage of this format, especially when programming in JavaScript. In JSON there is no concept as “tag”, it is easier and faster to parse, and it needs less data size to describe the same data as XML.

2.5 AngularJS

AngularJS is an open source, client-side JavaScript framework that promotes a high-productivity web development experience. The fundamental belief behind AngularJS is that declarative programming is the best choice to construct the user interface, while imperative programming is much better to build the application logic. AngularJS was created to be a framework that ensures a rapid speed of development and a long-term maintainability. It provides a scalable structure that simplifies developing large and complex applications. AngularJS extends traditional HTML by extending its vocabulary. This results
in expressive, reusable, and maintainable application components, that allow code reuse and avoid having a lot of repetitive code. One of the most important benefits is that AngularJS provides a clear separation of the concerns between the application layer, providing modularity, flexibility, and testability. In terms of concepts, a typical AngularJS application consists primarily of a view, model, and controller, but there are other important components, such as services, directives and filters. The first one, view, also called template, is entirely written in HTML. The controller is behind the view, and this one contains all the business logic implementation. However, as the application grows, it becomes really important to perform some refactoring activities, such as moving the code from the controller to other components. The connection between the view and the controller is done by a shared object called scope. The model is a simple JavaScript object, called a Plain-Old-JavaScript-Object (POJO). The scope is located between the view and the controller and is used to exchange information related to the model.

In the past, without the usage of REST API with JSON, the most common way to interact with the back-end was through HTTP with the help of the GET and POST methods. The first one was used to retrieve data, while the second one was used to create and update data.

The following are some examples of this concept:

- GET /retrieveChapters HTTP/1.1
- GET /getChapters HTTP/1.1
- GET /listChapters HTTP/1.1

Normally, if any HTTP method requires arguments, those are sent in the URL as part of the URL path (/resource/parametervalue) or as a query argument (/resource?parameter=value). Examples of this are:

- GET /retrieveChapters?chapterId=2 HTTP/1.1
- GET /getChapters?chapterTitle=Background HTTP/1.1
- GET /listChapters?idChapter=4 HTTP/1.1

The communication between an AngularJS application and the back-end relies on the HTTP protocol to transfer data through JSON objects. Normally, REST is used to communicate, that is based on the HTTP protocol by means of the use of most of its methods. The primary concept is to keep the URLs as simple and intuitive as possible by the use of a resource and not actions. In this way, the method will have as target the resource specified and if parameters are needed it will be transmitted within the request body using JSON.

The following are some examples of HTTP protocol used with JSON:

- GET /chapter HTTP/1.1
• POST /chapter HTTP/1.1

In the example above, the URL /chapter is kept as simple as possible and it specify a resource instead of an action. Whenever a method needs arguments, those are transfer within HTTP body in a JSON object. For example:

• `{“section”:“related work”, “chapter”:“1”}`

AngularJS is a widely used framework because it allows the creation of web applications easy to maintain with reusable components.

2.6 MEAN stack

MEAN is a full JavaScript stack and stands for MongoDB, Express.js, AngularJS, and Node.js [12]. AngularJS is used to develop the application front end, the Express is used to built the application backend and it is interpreted by the Node.js server. MongoDB is the database used by the server to store all the data needed. The next subsections explain these in more detail.

2.6.1 MongoDB

MongoDB is a NoSQL database management system designed for web applications and internet infrastructures. MongoDB stores data as JSON documents in a binary representation called BSON (Binary JSON). BSON extends the JSON representation to include additional types such as int, long, date, floating point, and decimal128. BSON documents contain one or more fields, and each field contains a value of a specific data type, including arrays, binary data, and sub-documents [13]. There are some differences between MySQL (SQL database) and MongoDB (NoSQL database). The main difference are the differences in the structures that store the data. In the MySQL data is stored in tables that have rows and columns, and in MongoDB the data is stored in collections of documents, and each document has fields. The statements each to each database type is different too. For example, to select a red car from a table in SQL the following query is issued: SELECT * FROM inventory WHERE status = "D", and to extract the same thing from a document in MongoDB it is used the query

```
db.inventory.find( { status: "D" } ).
```

MongoDB has a special capped collection, called OpLog, that keeps a rolling record of all operations that modify the data stored in the databases.
2.6.2 Express.js

Express is a minimal and flexible Node.js web application framework that allows the creation of web servers in an easy way. Express provides request routing, a static file server, view engine integration, and many modules developed by the developer community. [12]. In the MEAN stack Express.js is normally used to provide REST Endpoints that are used by AngularJS application to get data.

2.6.3 Node.js

Node.js is a framework used to create scalable network applications [12] built on Chrome’s V8 JavaScript engine. Node support server-side JavaScript development by having libraries for, or compatible, with it. Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and efficient in many use cases. There are third-party entities that develop Node modules that are published to npm, Node’s package manager. One of Node’s biggest use cases is the development of web servers, and there are a high number of modules that implement web servers. The most popular of these modules is Express (explain in the subsection above) [12].

Figure 2.3 illustrates the MEAN stack and the usage of its components. AngularJS is used to develop the client-side of the application and uses REST API with JSON objects to communicate with the server. Node.js is used to develop the server-side of the application and the Express.js module is used to communicate with the client-side. Express.js uses JSON objects to communicate with the AngularJS application. Finally, MongoDB is used as database and communicates with the server via JSON objects.

2.7 Machine Learning

Machine Learning is relevant for our work because, as will discussed in detail in Chapter 4, it allows learning how requests translate to database statements. Arthur Samuel defined machine learning in 1959 as the ability to learn without being explicitly programmed. At its most basic form, machine learning is the practice of using algorithms to parse data, learn from it, and then make a prediction about something.
Machine learning algorithms differ in their approach, the type of the input and output data, and the type of task or problem that they are intended to solve. In this dissertation we only use the classification, more precisely, the naïve Bayes classifier.

Naïve Bayes [14,15] is one of the most efficient and effective supervised machine learning algorithm. Naïve Bayes classifiers are linear classifiers that are known for being simple yet very efficient. The naïve Bayes classifiers are based on the Bayes’ theorem. Being relatively robust, easy to implement, fast, and accurate, naïve Bayes classifiers are used in many different fields. The goal is to accurately predict the label of test instances by using training instances that include label information to train the model.

<table>
<thead>
<tr>
<th>outlook</th>
<th>temperature</th>
<th>humidity</th>
<th>windy</th>
<th>play</th>
</tr>
</thead>
<tbody>
<tr>
<td>sunny</td>
<td>hot</td>
<td>high</td>
<td>FALSE</td>
<td>no</td>
</tr>
<tr>
<td>overcast</td>
<td>hot</td>
<td>high</td>
<td>FALSE</td>
<td>yes</td>
</tr>
</tbody>
</table>

Table 2.1: Weather data file example from Weka

Table 2.1 shows an example of a dataset. Each column represents an attribute and each row represents an instance.

A machine learning algorithm should be able to learn from this kind of data and given a set of instances evaluate their attributes and predict a label to a new set of instances to classify. In this case the label is the play attribute.

### 2.8 WEKA

WEKA [16] (Waikato Environment for Knowledge Analysis) is a workbench for machine learning that is intended to aid in the application of machine learning techniques to a variety of problems. WEKA provides an integrated environment which not only gives easy access to a variety of machine learning techniques through an interactive interface, but also incorporates those pre- and post-processing tools that are essential when working with real-world data sets. The goal behind WEKA is to move away from supporting a computer scientist or machine learning researcher, and move towards supporting the end user of machine learning.

ARFF (Attribute Relation File Format) was created to maintain format independence. ARFF files contain blocks describing relations and their attributes, together with all the instances of the relation. Relations are a single word or string naming the concept to be learned. Each attribute has a name, a data type (string, integer, etc) and a value. The instances of the relation are provided in comma-separated form. Missing or unknown values are specified by the ‘?’ character.
2.9 Summary

This chapter presented the frameworks and technologies that are used in our work, with special relevance to AngularJS, MongoDB, and their role in the MEAN architecture for applications. A short introduction to Machine Learning was also provided.
3

Related Work

Contents

3.1 Intrusion Recovery ......................................................... 23
3.2 Recovery at Application Level .............................................. 23
3.3 Comparison ............................................................... 27
In this chapter we present the fundamental concepts behind the recovery field and we discuss the previous intrusion recovery work. Section 3.1 introduces the intrusion recovery field and the phases of the intrusion recovery process. Section 3.2 discusses the relevant work done in the intrusion recovery field and ends with a comparison between them.

### 3.1 Intrusion Recovery

Intrusion recovery is an important security task that servers and data centers have to perform when the system has been compromised by intruders. Figure 2.1 summarizes some techniques that can be used to allow a system to recover from an intrusion. Those techniques remove all malicious actions related to the intrusion and return the application to a state where those effects are removed. The most relevant techniques in this work are *rollback* and the *rollforward* [1]. The first one aims to bring the system to a point before the intrusion, i.e., brings the system back to a state prior to the intrusion occurrence. This techniques aims to, after bringing the system to a state prior to the intrusion, redo all the requests that were not related to the intrusion, bringing the system to the same state in which the intrusion happened but without the effects of it.

When dealing with an intrusion, the first phase is to detect it. Intrusion detection is the process of monitoring the events in a system or network and analyzing them for signs of intrusion. Intrusion Detection Systems (IDS) [17–20] are used to monitor and detect suspicious behaviors and intrusions. These systems might need human interaction to deal with false positives so recovery mechanisms can be triggered.

Vulnerability management is the second phase on intrusion recovery. It is in this phase, after the detection of an intrusion that a group of experts identify, classify and mitigate vulnerabilities. Vulnerabilities are fixed by configuration adjustments or by applying security patches on the system, i.e., by uploading code developed to resolve that specific problem in an existing software.

The last phase consists in the removal of the intrusion effects from the system. Intrusions affect the application integrity, confidentiality and/or availability. This work focuses on removing the intrusions effects restoring the integrity of the application. Intrusion removal aims to recover a system that has been modified in a malicious or non-malicious way to a state where the effects of those modifications are not in the system.

### 3.2 Recovery at Application Level

**Undo for Operators** [21]: is a tool that allows system operators to recover from their own mistakes, from unanticipated software problems, and from intentional or accidental data corruption. This tool was
called Operator Undo and is based on three fundamental steps that the authors refer to as the “Three R’s: Rewind, Repair, and Replay” [22]. In the Rewind step all the system is physically rolled back to a point before any damage occurred. In the Repair step, the operator alters the rolled-back system to prevent the problem from reoccurring. In the Replay step, the repaired system is rolled forward to the present by replaying portions of the previously-rewound time-line in the context of the repaired system.

There is a time-travel storage layer that provides the ability to physically roll the system’s hard state back to a prior point in time. Undo for operators proposes a proxy interposed between the application and its users. This proxy intercepts the incoming user request stream to record the system timeline and injects its own requests to effect replay. The time-travel storage is below the service application’s operating system and the proxy is above the service application. The undo manager is responsible for coordinating the proxy and the time-travel storage layer, which maintains a history of user interactions comprising the system timeline. The communication between the Undo Proxy and the Undo manager are done with verbs. A verb is an encapsulation of a user interaction with the system, i.e., a record of an event that causes state in the service to be changed or externalized.

In the recovery phase, the operator determines the corrupted verbs and fixes their order by adding, deleting or changing verbs. After, the application is rewind, i.e., a system-wide snapshot is loaded to remove any corrupted data and the operator patch the software flaws of the application. Finally, all legitimate requests started after the intrusion are re-sent by the proxy to rebuild the application state. This tool was designed to help on servers dependability and does not fit the needs of web application recovery.

Data Recovery for Web Applications [23]: This work presents the design of a generic data recovery system for web applications that store their persistent data in a database tier. This system has three main components. The first one is the monitoring component that operates during run-time. This component it is used to track and correlate request across the application, allowing request-level data recovery. The monitors log sufficient information to allow mapping each request to database transactions, and transactions to specific tables and rows that were modified. The other two components are used after the corruption is detected. The first one is the analysis component that helps determine data corruption and loss related activities. This step is performed in a sandbox environment. The analysis component uses the data collected to derive three types of data dependencies, at the database, program and client level:

- **Database dependencies** - These are generated at the row field granularity based on the database rows or fields accessed by the application logic. These dependencies help correlate different requests based on the database operations performed by the requests. After corruption is detected, the administrator uses the tool to identify one or more initial requests that trigger the vulnerability
in the application. Then the analysis component generates dependent requests using a method similar to read-set templates. This method creates a read-set template for each query, and then materializes the rows read by the query based on the parameters passed to the query.

- **Application Dependencies** - The database dependencies can generate many false dependencies. This tool uses dynamic tainting to track application-logic dependencies within a request to prune this false dependencies from the dependency graph. This is done by tainting the initial request(s) in the dependency graph and replaying them. It then replays requests that have incoming edges in the dependency graph and uses tainting to prune outgoing edges that are created by untainted queries.

- **Client Dependencies** - The analysis component uses client-side dependencies across requests, such as login sessions and user accounts. This can help provide different starting points for the analysis by tainting all modifications from a specific user: an administrator might know that the data corruption started with a specific user.

The last component is the Recovery component. This component provides tools that simplify the recovery process. After the analysis component generates the set of tainted request, this component uses the information in the database log and the request and transaction mappings to generate compensating transactions. This tool does not replay application requests.

This tool implementation required modifications to existing software such as MySQL for the DB Monitor component, PHP interpreter for the Application-logic Monitor and Analysis components and JSQL-Parser for the Query Rewriter.

**WARP [6]**: is a system that helps users and administrators of web applications recover from intrusions while preserving legitimate user changes. The WARP system is built on the ideas from Retro, a tool that helps repairing from intrusions on operating systems. The WARP system aim is to recovery from vulnerabilities using security patches given by the administrator. The first step is to determine which runs of the application code may have been affected by a bug. WARP then applies the security patch and considers re-executing all potentially affected runs of the application. In order to re-execute the application, WARP records sufficient information during the original execution about all of the inputs to the application. WARP records and replays the original return values from non-deterministic function calls to minimize the chance that the application re-executes differently. When WARP re-executes the application code for the attacker's initial request, the newly patched application will behave differently, and then issued an SQL query to store the results in the database. WARP first rolls back the database to its state before the attack took place. After the database has been rolled back, and the new query has executed, WARP must determine what other parts of the system were affected by this changed query. To do this, during the original execution WARP records all SQL queries, along with their results.
WARP provides a browser extension that records all events for each open page in the browser (such as HTTP requests and user input) and uploads this information to the server. If WARP determines that the browser may have been affected by an attack, it starts a clone of the browser on the server, and re-executes the original input on the repaired page, without involving the user. If any conflict arise, WARP signals the conflict and asks the user (or administrator) to resolve it. The current design cannot re-execute web-applications involving multiple web servers, since the event logs for each application's frame would be uploaded to a different web server.

Shuttle [3]: is the first intrusion recovery service for PaaS. It is a service provided by Cloud service providers (CSPs) and assumes a client-server model in which clients communicate with the servers in the cloud using HTTP/HTTPS. For each application deployed in the Paas system, Shuttle records the requests issued by clients and creates periodic snapshots of the application database. After detection of the intrusion, Shuttle restores the application database loading the snapshot that precedes the intrusion and replays only legitimate requests. Requests are replayed asynchronously and, whenever possible, concurrently. The recovery process is deterministic because the accesses to each data item are performed in the original order of execution. Shuttle can operate in one of two phases: normal execution and recovery. During normal execution, Shuttle records the data required to recover the application's state: it does periodic database snapshots, logs user requests and database accesses. During the recovery phase, Shuttle creates a branch of the system execution in which it loads a snapshot, which contains the application state before the intrusion. It replays, in the new branch, the legitimate requests logged during normal execution, performing either selective or full replay. Full replay consists in replaying every request done after a snapshot. This approach is adequate for intrusions detected reasonably early after they happen. Selective replay re-executes only part of the requests, requiring that clients provide a set of malicious requests. This set is used to deduce the set of tainted requests. After the tainted requests are deduced, selective replay get the requests needed to obtain the values read by them. Next, shuttle recovery process determine the replay order sorted in start-end order. Finally, the process replay the requests. Shuttle is a recovery approach provided as a service integrated in a PaaS system and works with NoSQL databases.

Rectify [2]: is a tool for recovering from intrusions in PaaS applications without modifying the applications source code. These applications are typically web applications, i.e., software that runs in web servers, backed by databases, and that communicates with browsers. The application is considered a black-box and a machine learning scheme is used to automatically associate database statements to the request that generated them. Rectify uses supervised machine learning to find relations between faulty HTTP request and the corresponding database statements. In the Rectify learning phase the administrator needs to provide samples, allowing Rectify to learn that a specific HTTP request will generate
a certain kind of database statement. All the information gathered during the learning phase is captured and stored in the knowledge base. Each example in the knowledge base is identified by an application route. A route is a URL pattern that is mapped to a resource of the web application. Rectify is assisted by a web crawler to discover all routes. In order to identify the database statements issued by a malicious HTTP request, Rectify needs to solve two classification problems: signature matching - identifying the signature record, and DB statements matching - finding in the DB log the actual statements that were created by the malicious HTTP request.

**Signature Matching** - In this step the malicious HTTP request is parsed in order to extract its relevant parts (method, URL, parameters, etc). Using those parts, the classification algorithm is executed to find the corresponding signature record.

**DB Statements Matching** - Using the signature record from the signature matching, it is possible to find the corresponding database statements issued by the malicious request. First, the algorithm gets all the database statements of the signature record. Then Rectify calculates generic statements taking as an example the DB statements of the signature record and the parameter values from the malicious HTTP request. This generic statements should be as close as possible to the malicious DB statements, allowing a machine learning algorithm to identify them from the DB log.

Rectify removes the effects of an incorrect statement from the database by calculating a set of database statements, called compensation operations that undo what the malicious statement corrupted. We know that, in a simplistic scenario, in order to undo an insert it is necessary to execute a delete; to undo an update it is necessary to update the record back to its previous value. However, this problem becomes more difficult to solve in relational databases. In this kind of database, it is not recommended to remove a record that is related with other records because of the referential integrity constraints. In order to deal with this problem, an algorithm was used to calculate a graph of dependencies which will be used to execute the compensation operations. The algorithm to calculate and execute the compensation operations is the two pass repair algorithm [24] modified to allow undoing single statements.

### 3.3 Comparison

Table 3.1 presents a summary of the recovery phase of the services presented above. In this phase the intrusion services remove the intrusions effects and recover a consistent state. There are three methods to remove intrusion effects: snapshot loading, transaction compensation, and row versioning. The first one uses a state snapshot prior to the attack to recover a state with no intrusion effects. After the state recover, the studied works (Undo for Operators [21] and Shuttle [3]) replay all legitimate requests
<table>
<thead>
<tr>
<th>System</th>
<th>State Recovery</th>
<th>Effect removal</th>
<th>Replay phase</th>
<th>Runtime Recovery</th>
<th>Externally consistent</th>
</tr>
</thead>
<tbody>
<tr>
<td>Undo for Operators [21]</td>
<td>Snapshot</td>
<td>Load snapshot</td>
<td>Replay all requests sorted by application semantics</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Goel et al. [23]</td>
<td>Compensating transactions</td>
<td>Compensate tainted transactions</td>
<td>No</td>
<td></td>
<td></td>
</tr>
<tr>
<td>WARP [6]</td>
<td>Row versioning</td>
<td>Load previous entry version</td>
<td>Tainting via replay</td>
<td></td>
<td>✓ ✓</td>
</tr>
<tr>
<td>Shuttle [3]</td>
<td>Snapshot</td>
<td>Load snapshot</td>
<td>Replay tainted requests sorting by original execution read/write sets</td>
<td>✓</td>
<td>✓ ✓</td>
</tr>
<tr>
<td>Rectify [2]</td>
<td>Compensation transactions</td>
<td>Compensate tainted transactions</td>
<td>No</td>
<td></td>
<td>✓ ✓</td>
</tr>
</tbody>
</table>

Table 3.1: State recovery extended from Shuttle [3]

to recover all the changes in the state after the intrusion. The second method uses compensation to recover a state free of the intrusion effects [2, 23]. This is done by calculating and executing a set of database statements that undo what the malicious statement corrupted, e.g., to undo an insert it is necessary to execute a delete. The last method is the row versioning. This is a per-entry and per-write snapshot which allows the reading of previous versions without replay the actions after the snapshot. The intrusion effects are removed by loading a previous entry version of the rows in the database that were affected.

Rectify [2] is the only recovery tool studied that does not require source code modifications. In a PaaS system, where any type of web application can be deployed, an approach that does not require source code modifications is better because the web application can use this tool as a service provided by the cloud computing providers.
4
RectifyPlus

Contents

4.1 PaaS Architecture ................................................................. 31
4.2 Architecture ................................................................. 32
4.3 RectifyPlus Phases ................................................................. 33
4.4 Summary ................................................................. 40
In this chapter we give an overview of the system architecture of a Platform-as-a-Service (PaaS) and RectifyPlus, the extended version of Rectify, in Sections 4.1 and 4.2, respectively. Section 4.3 describes the operation phases of the tool, as required to perform intrusion recovery.

### 4.1 PaaS Architecture

Platform as a Service is one of the three original cloud computing models, alongside IaaS and SaaS [8]. Platform as a Service is a complete development and deployment environment in the cloud with resources on a pay-as-you-go basis. PaaS applications are typically web applications, i.e, software that runs in web servers, backed by databases and uses the HTTP protocol to communicate via browsers.

![Figure 4.1: Typical PaaS Architecture. Adapted from [2].](image)

Figure 4.1 presents the architecture of a PaaS platform. The *application delivery controller* (ADC) is responsible for the load balancing of the application, i.e, it is responsible for directing client requests to the appropriate server based on the server load and the application being accessed. Applications are deployed in a virtualized environment called *container*. Containers offer a logical packaging mechanism in which applications can be abstracted from the environment in which they actually run and are logically isolated from other containers using mechanisms such as *cgroups* (a Linux kernel feature that isolates computing resource usage). Containers provide APIs for applications to access the functionality
provided by the PaaS environment. An important one is the API for accessing the data layers, i.e., the database access and replication middleware.

4.2 Architecture

RectifyPlus system architecture is represented in figure 4.2. RectifyPlus itself is the set of components greyed out. The protected application hosted in container 1 is the web application that can be recovered by RectifyPlus in case of an intrusion.

Figure 4.2: RectifyPlus system architecture. RectifyPlus itself is the set of components greyed out. Adapted from [2].

In the figure, the arrows represent the interactions between RectifyPlus and the protected application. Arrow (a) represents the communication between the application delivery controller and the RectifyPlus HTTP proxy and the communication between the system administrator and the admin console, where all the functions are available. Arrow (b) represents the communication between the RectifyPlus HTTP proxy and the protected application. Normal users, without being transparent, communicate and interact with the protected application via RectifyPlus HTTP proxy and the protected application respond to those normal users via RectifyPlus proxy too. To be able to perform recovery, it is necessary to perform logging in the entrypoint of requests, the HTTP proxy.
RectifyPlus has four main components:

- **HTTP Proxy**: RectifyPlus uses a proxy to intercept and store HTTP requests issued to the protected application. This proxy is responsible for creating signature records (examples of HTTP requests and its corresponding database statements). To build this signature records, the proxy intercepts the HTTP requests issued by the administrator and finds, through the OpLog collection, the correspondent database statements.

- **Administrator Console**: RectifyPlus provides a shell interface to be used by the system administrator. This interface allows the administrator to configure RectifyPlus.

- **HTTP Log**: is a database in which RectifyPlus stores the HTTP requests intercepted by the proxy during the normal phase, i.e., while the application is available for the users. This log is used by RectifyPlus during the recovery phase.

- **Knowledge Base**: is a database in which RectifyPlus stores the signature records created during the learning phase.

## 4.3 RectifyPlus Phases

This section describes the execution phases of RectifyPlus. A phase is a runtime functionality of RectifyPlus that is controlled by the system administrator. These phases allow later to recover the protected application. The major distinction is between learning, normal and recovery phases. The subsection 4.3.1 describes the normal activity of RectifyPlus. Subsection 4.3.2 describes the creation process of the signature records and the creation of the classification model used by RectifyPlus to find the requests to recover from. Finally, subsection 4.3.3 describes the recovery phase in which malicious HTTP requests are removed from the protected application database.

### 4.3.1 Normal Phase

In the normal phase RectifyPlus intercepts the HTTP requests issued to the protected application and stores the ones that make changes to the protected application database. RectifyPlus uses a proxy to intercept the HTTP requests issued to the protected application. RectifyPlus only needs to store the HTTP requests that modify the application database the others do not have influence in the application database. In order to identify which HTTP requests modify the state of the protected application, RectifyPlus observes the header and the data of the HTTP requests and, if the HTTP request method is GET or POST with arguments or DELETE, then it makes changes to the protected application database. The information about those HTTP requests are store in a MongoDB database to be later used in the
learning phase. Figure 4.3 shows how data is extracted from the HTTP request by the RectifyPlus proxy and stored in the database. In this example, we show an HTTP request with POST method that issues the creation of a new post in the /api/posts URI with $t$ as the title of the post and $s$ as the slug of the post. The slug is a user friendly and URL valid name of a post, and its objective is to create a permalink for each post that can be used in the URL structure. Slugs are usually created automatically from the post title.

<table>
<thead>
<tr>
<th>HTTP Request</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
</tr>
<tr>
<td>method</td>
</tr>
<tr>
<td>uri</td>
</tr>
<tr>
<td>numberArgs</td>
</tr>
<tr>
<td>args</td>
</tr>
<tr>
<td>values</td>
</tr>
</tbody>
</table>

Figure 4.3: Stored HTTP Request example

### 4.3.2 Learning Phase

RectifyPlus uses supervised machine learning to predict and find relations between faulty HTTP requests and the corresponding database statements. For the RectifyPlus the protected application is a black-box application, so it intercepts the HTTP requests and finds the relation between them and the database statements without requiring modifications to the protected application source code. The purpose of the learning phase is to train a classification model capable of predicting the correct type of faulty HTTP request that will be used later on during the recovery phase. To create this model the RectifyPlus, with the help of the administrator, intercepts all HTTP requests and stores them with their corresponding database statements in the signature records database. Each signature record allows RectifyPlus to learn that a specific HTTP request will generate a certain set of database statements.

#### 4.3.2.A Signature Record

A signature record is identified by an application route. A route is a URL pattern that is mapped to a resource of the web application. For example, the URL www.meanie.com/posts points to a web page that displays the list of posts of that web site.

All signature records are stored in the Knowledge Base. The loading of the knowledge base is done by setting the operation mode of RectifyPlus to learning and executing all the routes available in the protected application. To do this, the administrator has to manually replicate all the possible interactions with the protected application. RectifyPlus, when in learning mode, forces a delay between HTTP requests so that there is no interference between two HTTP requests when it captures the corresponding
database statements. If there were HTTP requests being executed simultaneously, it would be hard to know to which request corresponded the database statements. When RectifyPlus proxy intercepts an HTTP request it searches the OpLog of the database for the corresponding database statements and stores the signature record for that HTTP request on the knowledge base database. If there is a route that was not learned by RectifyPlus it cannot be undone later, since there is no information in the Knowledge Base that allows such requests to be recovered/undone.

The Knowledge Base contains signature records which consist in associations between HTTP requests and a set of database statements. Figure 4.4 presents an example of a signature record of an HTTP request to insert a new post in a web page. The record is divided into HTTP and NoSQL parts. The HTTP part contains all the information about the request issued to the protected application. The NoSQL part contains the information about all database statements issued by the HTTP request that modifies the database. In this example, the HTTP request causes the execution of two NoSQL statements.

<table>
<thead>
<tr>
<th>Feature</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>HTTP</td>
<td>Method</td>
</tr>
<tr>
<td></td>
<td>URI</td>
</tr>
<tr>
<td></td>
<td>Nr. of parameters</td>
</tr>
<tr>
<td></td>
<td>Parameters</td>
</tr>
<tr>
<td></td>
<td>Values</td>
</tr>
<tr>
<td></td>
<td>POST</td>
</tr>
<tr>
<td></td>
<td>/api/posts</td>
</tr>
<tr>
<td></td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>[title,summary]</td>
</tr>
<tr>
<td></td>
<td>[t,s]</td>
</tr>
<tr>
<td>Nr. of Statements</td>
<td>2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Feature</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>NoSQL1</td>
<td>Type</td>
</tr>
<tr>
<td></td>
<td>Nr. of columns</td>
</tr>
<tr>
<td></td>
<td>Columns</td>
</tr>
<tr>
<td></td>
<td>Values</td>
</tr>
<tr>
<td></td>
<td>Namespace</td>
</tr>
<tr>
<td></td>
<td>i</td>
</tr>
<tr>
<td></td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>{id, title, summary}</td>
</tr>
<tr>
<td></td>
<td>{1,t,s}</td>
</tr>
<tr>
<td></td>
<td>meanie.posts</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Feature</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>NoSQL2</td>
<td>Type</td>
</tr>
<tr>
<td></td>
<td>Nr. of columns</td>
</tr>
<tr>
<td></td>
<td>Columns</td>
</tr>
<tr>
<td></td>
<td>Values</td>
</tr>
<tr>
<td></td>
<td>Namespace</td>
</tr>
<tr>
<td></td>
<td>u</td>
</tr>
<tr>
<td></td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>{expires}</td>
</tr>
<tr>
<td></td>
<td>{timestamp}</td>
</tr>
<tr>
<td></td>
<td>meanie.sessions</td>
</tr>
</tbody>
</table>

**Figure 4.4:** Signature Record example

4.3.2.B Classification

In order to identify the database statements issued by a malicious HTTP request RectifyPlus needs to solve a classification problem. This classification problem consists in identifying the signature record (class) that corresponds to the malicious HTTP request in order to find the database statements issued by such request. This is done by using Weka [16], a workbench for machine learning. Weka allows RectifyPlus to create a classification model capable of predicting the class of an HTTP request.

RectifyPlus starts by creating an ARFF file (file format used by Weka to store the instances –signature records – and its attributes) [16] containing the information of the HTTP requests of the signature record collection. Figure 4.5 shows the header of the ARFF file used by RectifyPlus to create the classification
model. This file contains an header that specifies its structure. RectifyPlus appends new records/in-
stances to this file respecting its structure.

```plaintext
@relation signRecords

@attribute method string
@attribute uri string
@attribute numberOfParams numeric
@attribute parameters string
@attribute values string
@attribute class string
```

Figure 4.5: Example of an ARFF file header created by RectifyPlus

Weka uses this file to train the classification model. The classification model used by RectifyPlus
is the naïve Bayes classifier [25, 26] because it accomplished 100% accuracy our experiments. This
was possible because the requests done to the protected application are not complex. Usually, 100%
accuracy in machine learning, is not easy to achieve, but the lack of complexity behind the protected
application logic and requests it was possible for us to achieve the 100% accuracy with the naïve Bayes
classifier. In order to achieve this level of accuracy some data preparation was required as well as
providing the classifier with enough signature records so it could learn the different patterns present in
each request.

To be able to use the naïve Bayes classifier more efficiently and to reach 100% accuracy RectifyPlus
uses preprocessing to filter and prepare the data. RectifyPlus uses three unsupervised filters to process
the data from the ARFF file:

- **Remove**: Used to remove the *values* attribute from the data ARFF file. The values attribute
come from the user input and can be very different from each HTTP request. To achieve 100% of
accuracy in the classification model it is needed to have a fixed number of different values to each
attribute, and, taking into account that the values attribute comes from user input, it can have an
infinite number of different values. Because of this RectifyPlus removes the values attribute from
que ARFF file.

- **StringToNominal**: Weka does not allow the use of naïve Bayes classifier with string attributes,
because of this RectifyPlus uses the StringToNominal filter to transform the method, parameters
and class string attributes to nominal attributes. This means that Weka gathers all possible values
for each attribute and fixes those values making a list of a fixed number of possible values to each attribute. Figure 4.6 shows the result of the usage of this filter on an ARFF file with string attributes.

As shown, after the usage of the filter the string attributes are replaced by the possible values from the data. If there were another data line with a DELETE method the attribute method would be replaced by the string \{ \text{'POST'; 'DELETE'} \}.

- **StringToWordVector**: The greater the difference between the values of the attributes from different data classes the higher the accuracy of the classification model. To prevent cases where the classification model can not be accurate when classifying an HTTP request, RectifyPlus uses the StringToWordVector filter to preprocess the URI attribute. This filter separates the URI string by the slash and counts the occurrence of the words in that string, i.e, converts the uri string attribute into a set of numeric attributes representing word occurrence information from the text contained in the strings.

After preprocessing, RectifyPlus stores the new file containing all the changes produced by the filters. RectifyPlus uses the 10-fold cross validation to validate the naïve Bayes classifier. The 10-fold cross validation divides the train set in 10 parts and uses each part for training and test on the other 9 parts. The model and the filters are saved to use later on the Recovery Phase to classify the malicious HTTP request.

4.3.3 Recovery Phase

To recover the protected application it is necessary the help of the application administrator to select the malicious HTTP request that needs to be undone, referred as malicious HTTP request through the rest of the document. RectifyPlus, when in recovery mode, displays the list of the HTTP requests intercepted by the RectifyPlus proxy during the normal phase. To do this, RectifyPlus accesses the HTTP log and displays the first 10 HTTP requests. The administrator can roll through the HTTP request until it reaches the last one.

Figure 4.7 shows the generic data flow diagram of the recovery phase performed by RectifyPlus to undo the malicious HTTP request from the protected application. In the figure, the system administrator...
consults the HTTP Log for malicious requests. These are used in the Signature Matching function to classify them to the corresponding signature record. With the signature record, RectifyPlus fills the generic DB statements with the values from the HTTP request. These generic DB statements can now be compared with the ones stored in the OpLog. Finally, RectifyPlus, with the aid of NoSQL Undo, executed compensating statements in the protected application database.

To recover the protected application RectifyPlus has to find the database statements issued by the malicious HTTP request. To do this, RectifyPlus uses the previously trained classification model to classify the malicious HTTP request. To use the classification model to identify the malicious HTTP request, RectifyPlus applies the filters used in the creation of the classification model to the malicious HTTP request. The malicious HTTP request is then parsed and the parsed information is saved in an ARFF file with the same header of the train set used to create the classification model. After filtering that file the classification model predict the label of the malicious HTTP request. RectifyPlus uses the label of the malicious HTTP request to run through the signature records database and find a signature record with the same label. This is done to get the database statements issued by the malicious request. Since a signature record is a set composed by an HTTP request and its issued database statements, with the label of the malicious HTTP requests RectifyPlus can get a set of generic database statements for that given label.

RectifyPlus uses the information from the malicious HTTP request to fill the generic database state-
ments. So, to have the database statements as similar as possible with the real malicious database statements, RectifyPlus replaces the generic database statements values with the malicious HTTP request parameters.

Figure 4.8: Data flow diagram with the tasks performed to identify the database statements issued by a malicious HTTP request. Diagram adapted from [2].

Figure 4.8 shows the tasks performed to identify the database statements issued by a malicious HTTP request. In the figure, the system administrator provides a malicious HTTP request, which is then parsed and classified. After classification, RectifyPlus can obtain a list of generic database statements that can be filled with the values from the HTTP requests. These statements can now be compared with the ones stored in the DB log.

RectifyPlus takes advantages of the MongoDB OpLog collections. The OpLog is a capped collection\(^1\) that keeps a rolling record of all operations that modify the data stored in the database. RectifyPlus filters the OpLog collection by the malicious HTTP request and gets a sub collection with the database statements that occurred around the same time that the malicious HTTP request was issued. With this, RectifyPlus has access to a more selected sub collection and do not have to search for the malicious database statements on the OpLog collection.

RectifyPlus uses the sub collection filtered from the OpLog collection to match the generic database statements filled with the data from the malicious HTTP request and the malicious database statements. To do this, RectifyPlus parses the records present in the sub collection and compares the values present in it with the generic database statements filled with the data from the malicious HTTP request. If the values are the same RectifyPlus marks that record as a malicious database statements.

To recover the protected application from the malicious HTTP request RectifyPlus uses the NoSQL Undo tool. NoSQL Undo [27] is a recovery approach and tool that allows database administrators to au-

\(^1\)A capped collection works as a circular array that replaces older entries with new ones
automatically remove the effect of faulty operations. RectifyPlus gives the malicious database statements found to the NoSQL Undo to recover the protected application.

4.4 Summary

In this chapter we provided an overview of RectifyPlus, describing its architecture and operation in the three phases: Normal, Learning and Recovery. The system is evaluated in the next chapter.
5

Evaluation

Contents

5.1 Performance Overhead .................................................. 43
5.2 Data Storage Space Overhead .......................................... 44
5.3 Total Time to Recover .................................................... 45
5.4 Summary ................................................................. 46
Our evaluation aims to answer the following questions: (a) What is the cost, in terms of performance, of using RectifyPlus? (b) How much space does RectifyPlus require to store the HTTP requests log and the knowledge base? (c) How much time does it take to recover a web application?

To evaluate RectifyPlus we used a virtual machine in VMWare Workstation\(^1\). The virtual machine was configured with 8GB of memory, 2 processors each one with 2 cores and 160GB of hard disk space. The virtual machine was running Ubuntu 18.04.1 version.

During the development and the experimental evaluation we used the MEANie CMS (Content Management System) and blogging platform. MEANie was built with the MEAN stack, i.e, with MongoDB, Express, AngularJS and Node.js. MEANie\(^2\) is a content management system that allows the creation of posts, pages and redirects.

### 5.1 Performance Overhead

RectifyPlus uses one proxy to intercept and store HTTP requests in the HTTP Log. The HTTP proxy causes an overhead to the system because it adds an extra step in the normal operation of the protected application.

The performance overhead experiment was done using the JMeter \([28]\) load-testing tool. It allows the creation of a test plan composed of user requests to later run it and issue those requests to a targeted application. We used the BlazeMeter \([29]\) Chrome extension to capture the requests issued to the protected application. To capture the HTTP requests with BlazeMeter we started the MEANie application and, with the BlazeMeter in record mode, interacted with the application covering all the possible routes of the application. The BlazeMeter stored all those HTTP requests in a file that JMeter was later capable to read.

The experiments consisted in a user intensive simulation using JMeter with 500 concurrent users. Each user issued a set of 38 requests to the application in which 7 of them where requests that caused modifications to the protected application’s database. Each user repeated 38 requests 8 times. In short, the tests where done mocking 500 users simultaneously issuing 38 request to the protected application and doing this 8 times. In total 152 000 requests were issued to the protected application by all 500 users.

Figure 5.1 shows the performance overhead of using RectifyPlus with MEANie as the protected application measured in requests per second. The first bar represents the requests per second issued to the MEANie application through the RectifyPlus proxy and the second bar represents the requests per second issued to the MEANie application without the RectifyPlus.

\(^1\)https://www.vmware.com/products/workstation-pro.html  
\(^2\)https://jasonwatmore.com/post/2016/10/29/meanie-mean-stack-blogging-platform
Figure 5.1: Performance overhead of using RectifyPlus with MEANie application measured in requests per second

The results show that without RectifyPlus proxy interception of HTTP requests the application receive an average of 52.4 requests per second and with RectifyPlus proxy intercepting the HTTP requests the application receive an average of 46.6 requests per second. This means that there is a performance degradation around 12% due to the use of the RectifyPlus proxy. This is expected because every HTTP requests issued to the protected application is intercepted by the RectifyPlus proxy and stored in the HTTP log database. We consider the present performance overhead is acceptable for many applications given the benefit provided by RectifyPlus. This values may improve if machines with better computing power are used along with optimized handling of input and output (I/O).

5.2 Data Storage Space Overhead

Over time the space occupied by the RetifyPlus logs will grow. With this experiment we wanted to measure how much space RectifyPlus logs require to store the HTTP requests issued to the protected application. RectifyPlus creates and updates only one log. After the experiments presented in section 5.1 we checked how much space the HTTP log created by RectifyPlus were taking in the database.

These experiments showed that the HTTP Log occupied 2.3MB of space after 152 000 HTTP requests were issued to the protected application MEANie of which only 28 000 HTTP requests caused modifications to the application’s database. The average log size per record is 63 bytes. Using this average we can extrapolate the storage size required for logs in applications serving more requests.

44
5.3 Total Time to Recover

The Total Time to Recover (TTR) is the time that RectifyPlus needs to recover the protected application since the instant that the system administrator selects the faulty http request and the instant that the compensation statements issued to the protected application database are completed. In practice this time was recorded by getting the time when the system administrator clicks to undo the selected malicious HTTP request and getting the time when the NoSQL Undo recover from the malicious database statements and subtracting those two times.

This experiments goal is to know the time that RectifyPlus needs to recover the protected application from a set of malicious HTTP requests. To do this we calculated the total time it took to recover from a series of malicious HTTP request from a single request to 1000 in intervals of 100. This experiment was done ten times for each set of malicious HTTP request.

![Figure 5.2: Total time to recover the protected application](image)

Figure 5.2 shows the average time to recover each set of requests. The time to recover grows linearly with the number of operations to undo. We choose a boxplot graph to represent the total time to recover data. A boxplot graph gives a good indication of how the values in the data are spread out. Boxplots is a way of displaying the distribution of data based on five components: Minimum: the smallest number represented in the dataset; First quartile: the middle number between the smallest number and the median; Median: the middle of the dataset; Third quartile: the middle number between the median and the highest number; Maximum: the highest number represented in the dataset.

Figure 5.2 shows that RectifyPlus recover the protected application from 1 faulty request in around 2
seconds while undoing 1,000 requests took around 23 minutes. The time to recover grows linearly with the number of operations to undo.

5.4 Summary

This chapter explained the methodology used to evaluate RectifyPlus. It showed that RectifyPlus imposes a 12% degradation in the performance of the protected application when used. Because RectifyPlus only needs to store the HTTP requests that modify the protected application database it adds a small space overhead (2.3MB to store 28,000 HTTP requests) to the application and it takes 2 seconds to undo 1 faulty request and 23 minutes to undo 1,000 requests.
Conclusion
More applications and data are being migrated to the cloud to take advantage of the 'computing as a service' models. However, applications are accessed through the public Internet and executed in hardware shared with other cloud customers and, as such, they are more exposed to attacks.

When the cloud defenses are crossed by an attacker, the applications suffer intrusions that can corrupt their state.

Nowadays to recover an application from a malicious actor, the system administrator searches the HTTP request to recover and performs manual recovery of records. This work presented RectifyPlus, a black-box intrusion recovery system for MEAN applications, a relevant subset of modern web applications. RectifyPlus is a solution to recover attacked web applications written using the MEAN stack that does not require modifications to the protected application source code and that can be performed by a system administrator. The RectifyPlus approach uses machine learning classification techniques to correlate HTTP requests to the database statements issued by them. Of the tested classification algorithms, the naïve Bayes classification algorithm showed 100% accuracy in our classification problem, being that the reason that we choose to use it as our machine learning classification algorithm.

Our evaluation showed that RectifyPlus imposes a 12% degradation in the performance overhead of the protected application. Given the benefit provided by RectifyPlus we consider that those 12% degradation is an acceptable tradeoff. RectifyPlus used 2.3MB to store 28,000 HTTP requests (an average of 63 bytes per request), and that is the only used space in a database that RectifyPlus uses for it to function. The oplog database is a MongoDB database and the space required to store the signature records is insignificant because it is needed a small amount of signature records to the classification model achieve 100% accuracy. RectifyPlus takes around 2 seconds to recover the protected application from 1 request and 23 minutes to undo 1,000 requests.

RectifyPlus allows for different future work directions: improve its interface and apply it to more applications.

The system evaluation was done with only one protected application. MEANie is a simple application without complex requests and with a low list of features (posts, pages and redirects).

For future work, it would be important to build a dedicated interface to the RectifyPlus tool. This interface should help the system administrator finding the faulty HTTP request faster and should enable interacting with the RectifyPlus without changing its source code.

RectifyPlus shell interface has a few limitations to the system administrator. The server proxy port and the protected application address have to be written in the source code of the RectifyPlus and the search for the faulty HTTP request by the administrator is done by iterating over the list of all the HTTP requests present in the HTTP log. The administrator can only see 10 HTTP requests at a time and he can not filter the HTTP request to find the faulty one faster. In a list with thousands of HTTP requests, as it is done now, the administrator will need to spend a lot of time iterating over the list until he finds the
HTTP request he wants to recover the system from.

Testing the RectifyPlus with other applications built using the MEAN full-stack would be important too to conclude how RectifyPlus reacts when it has to log more complex HTTP requests and recover applications with more features. It would be important too to realise, with an application with more routes that MEANie, how many signature records RectifyPlus would need to achieve 100% accuracy in the machine learning classification model.
Bibliography


