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. MEAN is a javascript full stack framework that uses MongoDB, Express.js, AngularJS and NodeJS. Because the applications are exposed on the Internet, these applications 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 solve the effects of the attacks. Recovery aims 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 an evolution of the Rectify tool furthering its scope, giving the tool the ability to recover web applications written with the MEAN stack. MEAN is a full-stack javascript framework that helps in the building of fast, robust and maintainable web applications using MongoDB, Express, AngularJS and Node.js. We propose RectifyPlus, an evolution of Rectify capable to recover applications written with MEAN full-stack javascript framework from intrusions. 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 content management system (CMS) MEANie and the results show that RectifyPlus is able to recover from intrusions effects whilst preserving the valid application data.

Keywords: Rollback, Recovery, Intrusion Removal, Intrusion recovery, MEAN web applications, AngularJS, MongoDB

1. Introduction

Cloud computing is transforming the IT industry [2], making software even more attractive as a service and shaping the way 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 costumers of information technology. Cloud computing services provide the resources that costumers 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 application 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.

Many customers and companies are migrating their applications and valuable information to cloud environments. The advantages of cloud computing are making the number of critical and complex applications in the cloud to increase rapidly. 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 [14] 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 most likely discard legitimate state modifications that occurred after the intrusion.

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 [4, 13] 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 persists in the new state.

The approach followed in this work consists in evolving the Rectify tool [10] 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 100 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. The goal of this work is to generalize Rectify to recover more web applications, in particular those that were developed with recent frameworks, like MEAN. Rectify can recover web applications that uses HTTP with parameters as communication protocol. This tool uses a route, i.e., an URL pattern such as www.app.com/posts/title, to build its knowledge base. Our goal is to improve this tool making it able to recover web applications that uses as communication the REST API with JSON objects and with the NoSQL database MongoDB as the database of the web application.

In this work, we propose an improvement of Rectify [10], increasing its scope of action. We propose, to the best of our knowledge, the first intrusion recovery system for PaaS applications that, as Rectify does not require changes in the application code, and can recover applications that use the REST API with JSON objects as their communication system.

2. Background

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.

2.1. 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 [11]. 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 [11].

**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.

2.2. 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.

Normally, if any HTTP method requires arguments, those are sent in the URL as part of the URL path (/resource/parameter=value) or as a query argument (/resource?parameter=value). The communication between an AngularJS application and the back-end relies on the HTTP protocol to transfer data through JSON objects. Normally, REST (Representational State Transfer) 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" }
```

2.3. MEAN stack
MEAN is a full JavaScript stack and stands for MongoDB, Express.js, AngularJS, and Node.js [7]. 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.

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 [12]. 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.

Express is a minimal and flexible Node.js web application framework that allows the creation of web servers in a easy way. Express provides request routing, a static file server, view engine integration, and many modules developed by the developer community. [7]. In the MEAN stack Express.js is normally used to provide REST Endpoints that are used by AngularJS application to get data.

Node.js is a framework used to create scalable network applications [7] built on Chrome's V8 JavaScript engine. Node support server-side JavaScript development by having libraries for, or compatible, with
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) [7].

Figure 1: Overview of MEAN stack

Figure 1 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.

2.4. Machine Learning
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 thesis we only use the classification, more precisely, the naïve Bayes classifier.

Naïve Bayes[16, 8] is one of the most efficient and effective supervised machine learning algorithm. The goal is to accurately predict the label of test instances by using training instances that include label information to train the model. The naïve Bayes goal is to learn from a set of instances and, given an instance of the same type, predict the label of that instance.

3. RectifyPlus
In this section we give an overview of the system architecture of a Platform-as-a-Service and RectifyPlus, the extended version of Rectify. Section 4 and section 5 give an overview of the PaaS and RectifyPlus architecture respectively. Section 6 describes the operation phases of the tool, as required to perform intrusion recovery.

4. PaaS Architecture
Platform as a Service is one of the three original cloud computing models, alongside IaaS and SaaS [11]. 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 2: Typical PaaS Architecture. Adapted from [10].

5. Architecture
Figure 3 is the representation of the RectifyPlus system architecture. RectifyPlus itself is the set of components greyed out. In the figure, the protected application hosted in container 1 is the web application that can be recovered by RectifyPlus in case of an intrusion. 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 the 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.

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.

6. 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 6.1 describes the normal activity of RectifyPlus. Subsection 6.2 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 6.3 describes the recovery phase in which malicious HTTP requests are removed from the protected application database.

### 6.1. 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 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.

<table>
<thead>
<tr>
<th>HTTP Request</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>id</em></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: Stored HTTP Request example*

### 6.2. 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.

6.2.1 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 website.

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.

6.2.2 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 [6], 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) [6] containing the information of the HTTP requests of the signature record collection. Figure 6 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/instances to this file respecting its structure.

<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>Nr. of Statements</td>
</tr>
<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>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>
</tbody>
</table>

Figure 5: Signature Record example

The Knowledge Base contains signature records which consist in associations between HTTP requests and a set of database statements. Figure 5 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.

Weka uses this file to train the classification model. The classification model used by RectifyPlus is the naïve Bayes classifier[15, 3] because it accomplished 100% accuracy our experiments. 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 the 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 7 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 {"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.

6.3. Recovery Phase

Figure 8 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 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.

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 clas-
sification model. After filtering that file the classification model predicts 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 statements. 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 values. These statements can now be compared with the ones stored in the DB log.

Figure 9 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 (a capped collection works as a circular array that replaces older entries with new ones) 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 statement.

To recover the protected application from the malicious HTTP request RectifyPlus uses the NoSQL Undo tool. NoSQL Undo [9] is a recovery approach and tool that allows database administrators to automatically remove the effect of faulty operations. RectifyPlus gives the malicious database statements found to the NoSQL Undo to recover the protected application.

7. Results & discussion

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. 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 is a content management system that allows the creation of posts, pages and redirects.

7.1. 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. Due to this, the performance will be better without the HTTP proxy and worst with the HTTP proxy.

The performance overhead experience was done using JMeter [5]. 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 [1] chrome extension to capture the HTTP 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, interact with the application covering all the possible routes of the application. The BlazeMeter stored all those HTTP requests in a file that JMeter was 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 10 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.

The results show that without RectifyPlus proxy in-
Figure 9: Data flow diagram with the tasks performed to identify the database statements issued by a malicious HTTP request. Diagram adapted from [10].

Figure 10: Performance overhead of using RectifyPlus with MEANie application measured in requests per second.

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.

7.2. Space Overhead
Over time the space occupied by the RectifyPlus logs will grow. With this experiment we wanted to know 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 7.1 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.

7.3. 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 11: Total time to recover the protected application
Figure 11 shows the average time to recover each set of requests. The time to recover grows linearly with the number of operations to undo. RectifyPlus recovered 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.

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 request and 23 minutes to undo 1,000 requests.

8. Conclusions
This work presented RectifyPlus, a black-box intrusion recovery system for MEAN 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. The naive 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, 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.

References