# MozDef Documentation

***Release***

**Mozilla**

**Sep 28, 2017**

# Contents


**1 Overview** **1**
- 1.1 Why? . . . 1
- 1.2 Goals . . . 1
- 1.3 Architecture . . . 2
- 1.4 Status . . . 3
- 1.5 Roadmap . . . 3

**2 Introduction** **5**
- 2.1 Concept of operations . . . 5

**3 Demo Instance** **7**

**4 Installation** **9**
- 4.1 Docker . . . 9
- 4.2 MozDef manual installation process on RedHat systems . . . 10
- 4.3 Elasticsearch nodes . . . 10
- 4.4 Web and Workers nodes . . . 10
- 4.5 Manual Installation . . . 15

**5 Screenshots** **19**
- 5.1 Health and Status . . . 19
- 5.2 Alerts . . . 20
- 5.3 Incident Handling . . . 21
- 5.4 d3 visualizations . . . 22
- 5.5 Geo location of Attackers . . . 23
- 5.6 3D interactive Attacker visualization . . . 23
- 5.7 3D interactive Attack visualization via Landmass . . . 24

**6 Usage** **25**
- 6.1 Web Interface . . . 25
- 6.2 Sending logs to MozDef . . . 25
- 6.3 JSON format . . . 28
- 6.4 Writing alerts . . . 30

**7 Advanced Settings** **31**
- 7.1 Conf files . . . 31

**8 Code** **35**
8.1 Plugins . . . . . 35

**9 Benchmarking** **39**
9.1 Elasticsearch . . . . . 39

**10 Contributors** **41**

**11 Indices and tables** **43**

**12 License** **45**

**13 Contact** **47**

CHAPTER 1

# Overview

## Why?

The inspiration for MozDef comes from the large arsenal of tools available to attackers. Suites like metasploit, armitage, lair, dradis and others are readily available to help attackers coordinate, share intelligence and finely tune their attacks in real time. Defenders are usually limited to wikis, ticketing systems and manual tracking databases attached to the end of a Security Information Event Management (SIEM) system.

The Mozilla Defense Platform (MozDef) seeks to automate the security incident handling process and facilitate the real-time activities of incident handlers.

## Goals

### High level

- Provide a platform for use by defenders to rapidly discover and respond to security incidents.
- Automate interfaces to other systems like MIG, flowspec, load balancers, etc
- Provide metrics for security events and incidents
- Facilitate real-time collaboration amongst incident handlers
- Facilitate repeatable, predictable processes for incident handling
- Go beyond traditional SIEM systems in automating incident handling, information sharing, workflow, metrics and response automation

### Technical

- Replace a Security Information and Event Management (SIEM)

- Scalable, should be able to handle thousands of events per second, provide fast searching, alerting, correlation and handle interactions between teams of incident handlers.

MozDef aims to provide traditional SIEM functionality including:

- Accepting events/logs from a variety of systems
- Storing events/logs
- Facilitating searches
- Facilitating alerting
- Facilitating log management (archiving,restoration)

It is non-traditional in that it:

- Accepts only JSON input
- Provides you open access to your data
- Integrates with a variety of log shippers including heka, logstash, beaver, nxlog and any shipper that can send JSON to either rabbit-mq or an HTTP endpoint.
- Provides easy python plugins to manipulate your data in transit
- Provides realtime access to teams of incident responders to allow each other to see their work simultaneously

# Architecture

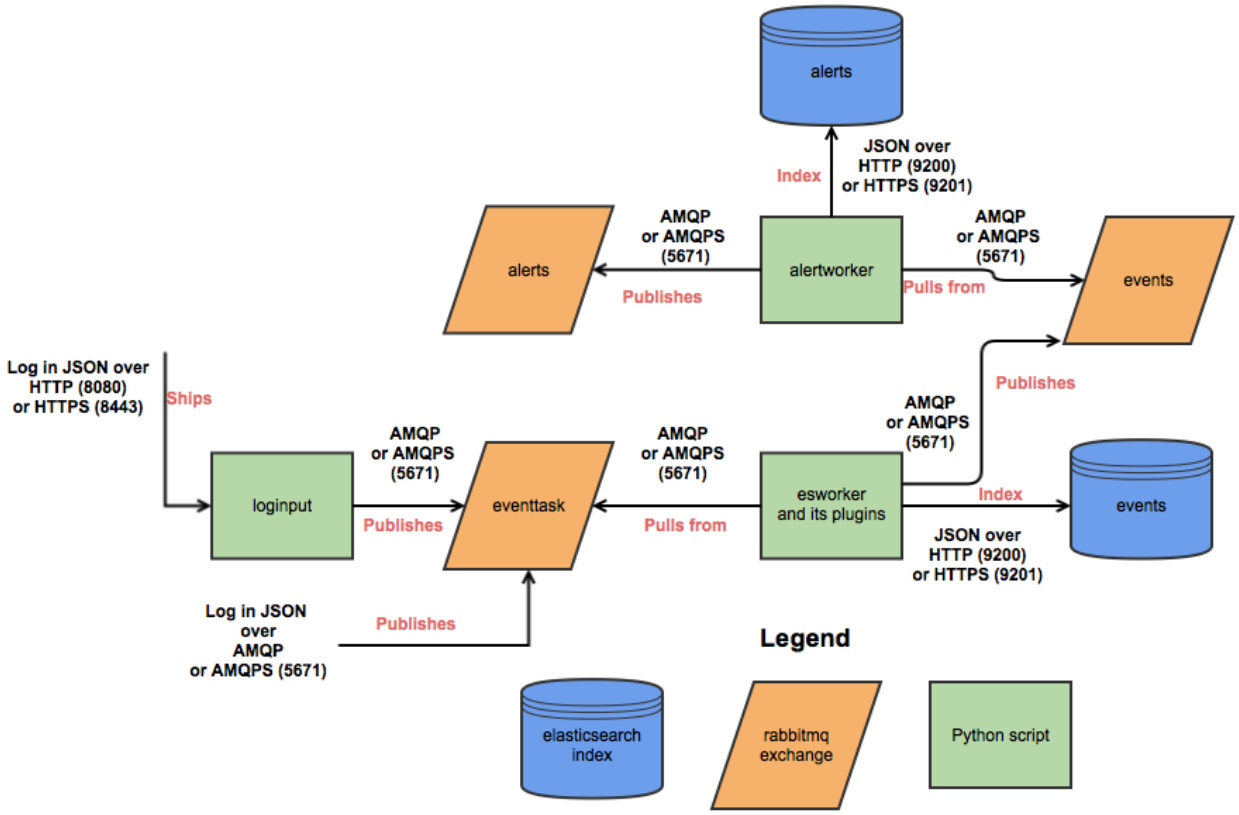
MozDef is based on open source technologies including:

- Nginx (http(s)-based log input)
- RabbitMQ (message queue and amqp(s)-based log input)
- uWSGI (supervisory control of python-based workers)
- bottle.py (simple python interface for web request handling)
- elasticsearch (scalable indexing and searching of JSON documents)
- Meteor (responsive framework for Node.js enabling real-time data sharing)
- MongoDB (scalable data store, tightly integrated to Meteor)
- VERIS from verizon (open source taxonomy of security incident categorizations)
- d3 (javascript library for data driven documents)
- dc.js (javascript wrapper for d3 providing common charts, graphs)
- three.js (javascript library for 3d visualizations)
- Firefox (a snappy little web browser)

## Frontend processing

Frontend processing for MozDef consists of receiving an event/log (in json) over HTTP(S) or AMQP(S), doing data transformation including normalization, adding metadata, etc. and pushing the data to elasticsearch.

Internally MozDef uses RabbitMQ to queue events that are still to be processed. The diagram below shows the interactions between the python scripts (controlled by uWSGI), the RabbitMQ exchanges and elasticsearch indices.




## Status

MozDef is in production at Mozilla where we are using it to process over 300 million events per day.

## Roadmap

Initial Release:

- Facilitate replacing base SIEM functionality including log input, event management, search, alerts, basic correlations
- Enhance the incident workflow UI to enable realtime collaboration
- Enable basic plug-ins to the event input stream for meta data, additional parsing, categorization and basic machine learning
- Support as many common event/log shippers as possible with repeatable recipies
- 3D visualizations of threat actors

Mid term:

- Repeatable installation guides
- Ready-made AMIs/downloadable ISOs
- Correlation through machine learning, AI

- Base integration into Mozilla's defense mechanisms for automation
- Fine tuning of interactions between meteor, mongo, dc.js
- Support a variety of authentication/authorization schemes/technologies
- Plain text version of attackers
- Enhanced search for alerts, events, attackers within the MozDef UI

Long term:

- Integration into common defense mechanisms used outside Mozilla
- Enhanced visualizations and interactions including alternative interfaces (myo, omnidirectional treadmills, oculus rift)

CHAPTER 2

# Introduction

## Concept of operations

### Event Management

From an event management point of view MozDef relies on Elastic Search for:

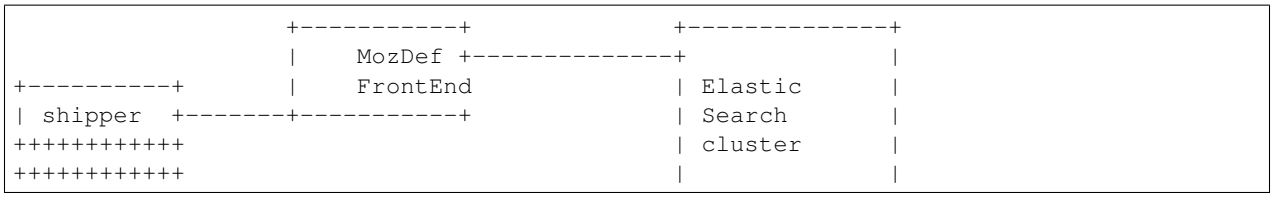
- event storage
- event archiving
- event indexing
- event searching

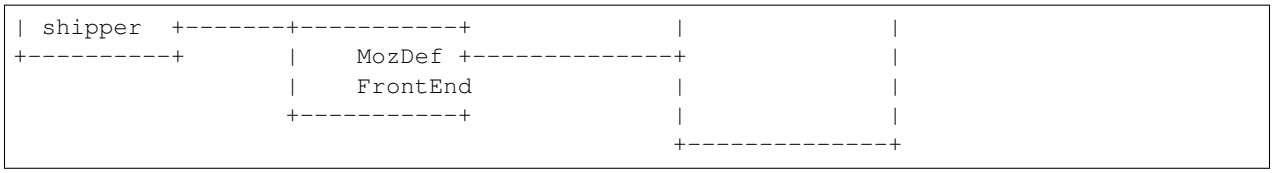
This means if you use MozDef for your log management you can use the features of Elastic Search to store millions of events, archive them to Amazon if needed, index the fields of your events, and search them using highly capable interfaces like Kibana.

MozDef differs from other log management solutions that use Elastic Search in that it does not allow your log shippers direct contact with Elastic Search itself. In order to provide advanced functionality like event correlation, aggregation and machine learning, MozDef inserts itself as a shim between your log shippers (rsyslog, syslog-ng, beaver, nxlog, heka, logstash) and Elastic Search. This means your log shippers interact with MozDef directly and MozDef handles translating their events as they make they're way to Elastic Search.

### Event Pipeline

The logical flow of events is:






Choose a shipper (logstash, nxlog, beaver, heka, rsyslog, etc) that can send JSON over http(s). MozDef uses nginx to provide http(s) endpoints that accept JSON posted over http. Each front end contains a Rabbit-MQ message queue server that accepts the event and sends it for further processing.

You can have as many front ends, shippers and cluster members as you with in any geographic organization that makes sense for your topology. Each front end runs a series of python workers hosted by uwsgi that perform:

- event normalization (i.e. translating between shippers to a common taxonomy of event data types and fields)
- event enrichment
- simple regex-based alerting
- machine learning on the real-time event stream

## Event Enrichment

To facilitate event correlation, MozDef allows you to write plugins to populate your event data with consistent meta-data customized for your environment. Through simple python plug-ins this allows you to accomplish a variety of event-related tasks like:

- further parse your events into more details
- geoIP tag your events
- correct fields not properly handled by log shippers
- tag all events involving key staff
- tag all events involving previous attackers or hits on a watchlist
- tap into your event stream for ancilary systems
- maintain 'last-seen' lists for assets, employees, attackers

## Event Correlation/Alerting

Correlation/Alerting is currently handled as a series of queries run periodically against the Elastic Search engine. This allows MozDef to make full use of the lucene query engine to group events together into summary alerts and to correlate across any data source accessible to python.

## Incident Handling

From an incident handling point of view MozDef offers the realtime responsiveness of Meteor in a web interface. This allows teams of incident responders the ability to see each others actions in realtime, no matter their physical location.

CHAPTER 3

# Demo Instance

Mozilla maintains a demo instance of MozDef that you can use try out the UI and get a feel for it in a live environment with test/random data.

Simply browse to http://demo.mozdef.com:3000 and login using any gmail or yahoo email address. No credentials/passwords are sent to the demo instance, though your email will be logged. If you'd prefer you can also use mozdef@mockmyid.com as a userID which will not prompt for any credentials.

CHAPTER 4

# Installation

The installation process has been tested on CentOS 6, RHEL 6 and Ubuntu 14.

## Docker

You can quickly install MozDef with an automated build generation using docker.

### Single Container

MozDef can run in a single docker container, which uses supervisord to handle executing all of the MozDef processes. In order to run a single container:

```
make single-build
make single-run
make single-stop # When you want to stop the container
```

You're done! Now go to:

- http://localhost < meteor (main web interface)
- http://localhost:9090/app/kibana < kibana
- http://localhost:9200 < elasticsearch
- http://localhost:8080 < loginput
- http://localhost:8081 < rest api

### Multiple Containers

Since MozDef consists of many processes running at once, we also support running MozDef with each process given it's own container. This can be useful during development, since you can turn off a single process to debug/troubleshoot while maintaining a functioning MozDef environment. In order to run in multiple containers:

```
make multiple-build
make multiple-run
make multiple-stop # When you want to stop the containers
```

You're done! Now go to:

- http://localhost < meteor (main web interface)
- http://localhost:9090/app/kibana < kibana
- http://localhost:9200 < elasticsearch
- http://localhost:8080 < loginput
- http://localhost:8081 < rest api

# MozDef manual installation process on RedHat systems

## Summary

**This section explains the manual installation process for the MozDef system.** git clone https://github.com/mozilla/MozDef.git

# Elasticsearch nodes

This section explains the manual installation process for Elasticsearch nodes (search and storage).

## ElasticSearch

Installation instructions are available on Elasticsearch website. You should prefer packages over archives if one is available for your distribution.

## Marvel plugin

Marvel is a monitoring plugin developed by Elasticsearch (the company).

WARNING: this plugin is NOT open source. At the time of writing, Marvel is free for 30 days. After which you can apply for a free basic license to continue using it for it's key monitoring features.

To install Marvel, on each of your elasticsearch node, from the Elasticsearch home directory:

```
sudo bin/plugin install license
sudo bin/plugin install marvel-agent
sudo service elasticsearch restart
```

You should now be able to access to Marvel at http://any-server-in-cluster:9200/_plugin/marvel

# Web and Workers nodes

This section explains the manual installation process for Web and Workers nodes.

## Python

Create a mozdef user:

```
adduser mozdef -d /opt/mozdef
```

We need to install a python2.7 virtualenv.

On Yum-based systems:

```
sudo yum install make zlib-devel bzip2-devel openssl-devel ncurses-devel sqlite-devel␣
↪readline-devel tk-devel pcre-devel gcc gcc-c++ mysql-devel
```

On APT-based systems:

```
sudo apt-get install make zlib1g-dev libbz2-dev libssl-dev libncurses5-dev libsqlite3-
↪dev libreadline-dev tk-dev libpcre3-dev libpcre++-dev build-essential g++␣
↪libmysqlclient-dev
```

Then:

```
su - mozdef
wget https://www.python.org/ftp/python/2.7.11/Python-2.7.11.tgz
tar xvzf Python-2.7.11.tgz
cd Python-2.7.11
./configure --prefix=/opt/mozdef/python2.7 --enable-shared
make
make install

cd /opt/mozdef

wget https://bootstrap.pypa.io/get-pip.py
export LD_LIBRARY_PATH=/opt/mozdef/python2.7/lib/
./python2.7/bin/python get-pip.py
./python2.7/bin/pip install virtualenv
mkdir ~/envs
cd ~/envs
~/python2.7/bin/virtualenv mozdef
source mozdef/bin/activate
pip install -r MozDef/requirements.txt
```

At this point when you launch python, It should tell you that you're using Python 2.7.11.

Whenever you launch a python script from now on, you should have your mozdef virtualenv actived and your LD_LIBRARY_PATH env variable should include /opt/mozdef/python2.7/lib/

## RabbitMQ

RabbitMQ is used on workers to have queues of events waiting to be inserted into the Elasticsearch cluster (storage).

RabbitMQ does provide a zero-dependency RPM that you can find for RedHat/CentOS here:: https://github.com/rabbitmq/erlang-rpm

For Debian/Ubuntu based distros you would need to install erlang separately.

To install it, first make sure you enabled EPEL repos. Then you need to install an Erlang environment.

If you prefer to install all the dependencies on a Red Hat based system you can do the following:: On Yum-based systems:

```
sudo yum install erlang
```

You can then install the rabbitmq server:

```
sudo rpm --import https://www.rabbitmq.com/rabbitmq-signing-key-public.asc
sudo yum install rabbitmq-server
```

To start rabbitmq at startup:

```
chkconfig rabbitmq-server on
```

On APT-based systems

```
sudo apt-get install rabbitmq-server
sudo invoke-rc.d rabbitmq-server start
```

## Meteor

Meteor is a javascript framework used for the realtime aspect of the web interface.

We first need to install Mongodb since it's the DB used by Meteor.

On Yum-based systems:

In /etc/yum.repo.d/mongo, add:

```
[mongodb]
name=MongoDB Repository
baseurl=http://downloads-distro.mongodb.org/repo/redhat/os/x86_64/
gpgcheck=0
enabled=1
```

Then you can install mongodb:

```
sudo yum install mongodb
```

On APT-based systems:

```
sudo apt-get install mongodb-server
```

For meteor, in a terminal:

```
curl https://install.meteor.com/ | sh

wget https://nodejs.org/dist/v4.7.0/node-v4.7.0.tar.gz
tar xvzf node-v4.7.0.tar.gz
cd node-v4.7.0
./configure
make
sudo make install
```

Then from the meteor subdirectory of this git repository (/opt/mozdef/MozDef/meteor) run:

```
meteor add iron-router
```

If you wish to use meteor as the authentication handler you'll also need to install the Accounts-Password pkg:

```
meteor add accounts-password
```

You may want to edit the app/lib/settings.js file to properly configure the URLs and Authentication The default setting will use Meteor Accounts, but you can just as easily install an external provider like Github, Google, Facebook or your own OIDC:

```
mozdef = {
  rootURL: "localhost",
  port: "443",
  rootAPI: "https://localhost:8444",
  kibanaURL: "https://localhost:9443/app/kibana#",
  enableBlockIP: true,
  enableClientAccountCreation: true,
  authenticationType: "meteor-password"
}
```

or for an OIDC implementation that passes a header to the nginx reverse proxy (for example using OpenResty with Lua and Auth0):

```
mozdef = {
  rootURL: "localhost",
  port: "443",
  rootAPI: "https://localhost:8444",
  kibanaURL: "https://localhost:9443/app/kibana#",
  enableBlockIP: true,
  enableClientAccountCreation: false,
  authenticationType: "OIDC"
}
```

Then start meteor with:

```
meteor
```

## Node

Alternatively you can run the meteor UI in 'deployment' mode using a native node installation.

First install node:

```
yum install bzip2 gcc gcc-c++ sqlite sqlite-devel
wget https://nodejs.org/dist/v4.7.0/node-v4.7.0.tar.gz
tar xvfz node-v4.7.0.tar.gz
cd node-v4.7.0
python configure
make
make install
```

Then bundle the meteor portion of mozdef:

```
cd <your meteor mozdef directory>
meteor bundle mozdef.tgz
```

You can then deploy the meteor UI for mozdef as necessary:

```
scp mozdef.tgz to your target host
tar -xvzf mozdef.tgz
```

This will create a 'bundle' directory with the entire UI code below that directory.

If you didn't update the settings.js before bundling the meteor installation, you will need to update the settings.js file to match your servername/port:

```
vim bundle/programs/server/app/app/lib/settings.js
```

If your development OS is different than your production OS you will also need to update the fibers node module:

```
cd bundle/programs/server/node_modules
rm -rf fibers
sudo npm install fibers@1.0.1
```

There are systemd unit files available in the systemd directory of the public repo you can use to start meteor using node. If you aren't using systemd, then run the mozdef UI via node manually:

```
export MONGO_URL=mongodb://mongoservername:3002/meteor
export ROOT_URL=http://meteorUIservername/
export PORT=443
node bundle/main.js
```

## Nginx

We use nginx webserver.

You need to install nginx:

```
sudo yum install nginx
```

On apt-get based system:

```
sudo apt-get nginx
```

If you don't have this package in your repos, before installing create */etc/yum.repos.d/nginx.repo* with the following content:

```
[nginx]
name=nginx repo
baseurl=http://nginx.org/packages/OS/OSRELEASE/$basearch/
gpgcheck=0
enabled=1
```

## UWSGI

We use uwsgi to interface python and nginx:

```
wget https://projects.unbit.it/downloads/uwsgi-2.0.12.tar.gz
tar zxvf uwsgi-2.0.12.tar.gz
cd uwsgi-2.0.12
~/python2.7/bin/python uwsgiconfig.py --build
~/python2.7/bin/python uwsgiconfig.py  --plugin plugins/python core
cp python_plugin.so ~/envs/mozdef/bin/
cp uwsgi ~/envs/mozdef/bin/

cp -r ~/MozDef/rest   ~/envs/mozdef/
```

```
cp -r ~/MozDef/loginput   ~/envs/mozdef/
mkdir ~/envs/mozdef/logs

cd ~/envs/mozdef/rest
# modify config file
vim index.conf
# modify uwsgi.ini
vim uwsgi.ini
uwsgi --ini uwsgi.ini

cd ../loginput
# modify uwsgi.ini
vim uwsgi.ini
uwsgi --ini uwsgi.ini

sudo cp nginx.conf /etc/nginx
# modify /etc/nginx/nginx.conf
sudo vim /etc/nginx/nginx.conf
sudo service nginx restart
```

### Kibana

Kibana is a webapp to visualize and search your Elasticsearch cluster data:

```
wget https://download.elastic.co/kibana/kibana/kibana-4.6.2-linux-x86_64.tar.gz
tar xvzf kibana-4.6.2-linux-x86_64.tar.gz
ln -s kibana-4.6.2 kibana
# configure /etc/nginx/nginx.conf to target this folder
sudo service nginx reload
```

To initialize elasticsearch indices and load some sample data:

```
cd examples/es-docs/
python inject.py
```

### Start Services

TO DO: Add in services like supervisord, and refer to systemd files.

Start the following services

cd ~/MozDef/mq ./esworker.py

cd ~/MozDef/alerts celery -A celeryconfig worker –loglevel=info –beat

cd ~/MozDef/examples/demo ./syncalerts.sh ./sampleevents.sh

## Manual Installation

*Use sudo whereever required*

**(Currently only for apt-based systems)**

1. Cloning repository

```
$ export MOZDEF_PATH=/opt/MozDef
$ git clone https://github.com/mozilla/MozDef.git $MOZDEF_PATH
```

2. Installing dependencies

```
# RabbitMQ
$ apt-get install -y rabbitmq-server
$ rabbitmq-plugins enable rabbitmq_management

# MongoDB
$ apt-get install -y mongodb

# NodeJS and NPM
$ curl -sL https://deb.nodesource.com/setup_0.12 | sudo bash -
$ apt-get install -y nodejs npm

# Nginx
$ apt-get install -y nginx-full
$ cp $MOZDEF_PATH/docker/conf/nginx.conf /etc/nginx/nginx.conf

# Libraries
$ apt-get install -y python2.7-dev python-pip curl supervisor wget libmysqlclient-
↪dev
$ pip install -U pip
```

3. Installing python libraries

```
$ pip install uwsgi celery virtualenv

$ export PATH_TO_VENV=$HOME/.mozdef_env
$ virtualenv $PATH_TO_VENV
$ source $PATH_TO_VENV/bin/activate

(.mozdef_env)$ pip install -r $MOZDEF_PATH/requirements.txt
```

4. Setting up uwsgi for rest and loginput

```
$ mkdir /var/log/mozdef
$ mkdir -p /run/uwsgi/apps/
$ touch /run/uwsgi/apps/loginput.socket
$ chmod 666 /run/uwsgi/apps/loginput.socket
$ touch /run/uwsgi/apps/rest.socket
$ chmod 666 /run/uwsgi/apps/rest.socket
```

5. Setting up local settings

```
$ cp $MOZDEF_PATH/docker/conf/supervisor.conf /etc/supervisor/conf.d/supervisor.
↪conf
$ cp $MOZDEF_PATH/docker/conf/settings.js $MOZDEF_PATH/meteor/app/lib/settings.js
$ cp $MOZDEF_PATH/docker/conf/config.py $MOZDEF_PATH/alerts/lib/config.py
$ cp $MOZDEF_PATH/docker/conf/sampleData2MozDef.conf $MOZDEF_PATH/examples/demo/
↪sampleData2MozDef.conf
$ cp $MOZDEF_PATH/docker/conf/mozdef.localloginenabled.css $MOZDEF_PATH/meteor/
↪public/css/mozdef.css
```

6. Installing Kibana

```
$ cd /tmp/
$ curl -L https://download.elastic.co/kibana/kibana/kibana-4.6.2-linux-x86_64.tar.
↪gz | tar -C /opt -xz
$ /bin/ln -s /opt/kibana-4.6.2 /opt/kibana
$ cp $MOZDEF_PATH/examples/kibana/dashboards/alert.js /opt/kibana/app/dashboards/
↪alert.js
$ cp $MOZDEF_PATH/examples/kibana/dashboards/event.js /opt/kibana/app/dashboards/
↪event.js
```

7. Installing Elasticsearch

```
For Red Hat based:
$ wget https://download.elastic.co/elasticsearch/release/org/elasticsearch/
↪distribution/rpm/elasticsearch/2.4.5/elasticsearch-2.4.5.rpm
For Debian based:
$ wget https://download.elastic.co/elasticsearch/release/org/elasticsearch/
↪distribution/deb/elasticsearch/2.4.5/elasticsearch-2.4.5.deb
# You can download and install any version of ELasticSearch > 2.x and < 5.x
```

8. Setting up Meteor

```
$ curl -L https://install.meteor.com/ | /bin/sh
$ cd $MOZDEF_PATH/meteor
$ meteor
```

9. Inserting some sample data

```
# Elasticsearch server should be running
$ service elasticsearch start
$ source $PATH_TO_VENV/bin/activate
(.mozdef_env)$ cd $MOZDEF_PATH/examples/es-docs && python inject.py
```

## Start Services

Start the following services

```
$ invoke-rc.d rabbitmq-server start

$ service elasticsearch start

$ service nginx start

$ uwsgi --socket /run/uwsgi/apps/loginput.socket --wsgi-file $MOZDEF_PATH/loginput/
↪index.py --buffer-size 32768 --master --listen 100 --uid root --pp $MOZDEF_PATH/
↪loginput --chmod-socket --logto /var/log/mozdef/uwsgi.loginput.log -H $PATH_TO_VENV

$ uwsgi --socket /run/uwsgi/apps/rest.socket --wsgi-file $MOZDEF_PATH/rest/index.py --
↪buffer-size 32768 --master --listen 100 --uid root --pp $MOZDEF_PATH/rest --chmod-
↪socket --logto /var/log/mozdef/uwsgi.rest.log -H $PATH_TO_VENV

$ cd $MOZDEF_PATH/mq && uwsgi --socket /run/uwsgi/apps/esworker.socket --
↪mule=esworker.py --mule=esworker.py --buffer-size 32768 --master --listen 100 --uid␣
↪root --pp $MOZDEF_PATH/mq --stats 127.0.0.1:9192  --logto /var/log/mozdef/uwsgi.
↪esworker.log --master-fifo /run/uwsgi/apps/esworker.fifo -H $PATH_TO_VENV

$ cd $MOZDEF_PATH/meteor && meteor run
```

```
# Activate the virtualenv to run background jobs
$ source $PATH_TO_VENV/bin/activate

(.mozdef_env)$ cd $MOZDEF_PATH/alerts && celery -A celeryconfig worker --
↪loglevel=info --beat
(.mozdef_env)$ cd $MOZDEF_PATH/examples/demo && ./healthjobs.sh
(.mozdef_env)$ cd $MOZDEF_PATH/examples/demo && ./sampleevents.sh
(.mozdef_env)$ cd $MOZDEF_PATH/examples/demo && ./syncalerts.sh
```

CHAPTER 5

# Screenshots

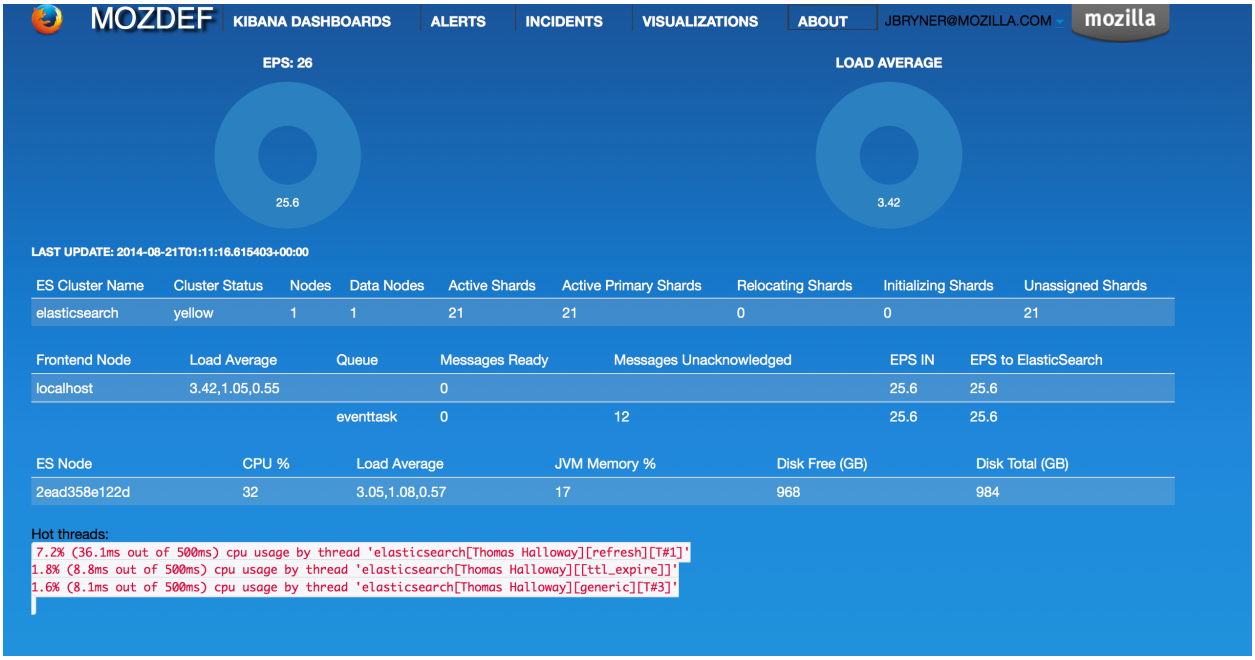
Here are a few screen captures of key portions of the MozDef user interface.

## Health and Status

MozDef includes an integrated health and status screen under the 'about' menu showing key performance indicators like events per second from rabbit-mq and elastic search cluster health.

You can have as many front-end processors running rabbit-mq as you like in whatever geographic distribution makes sense for your environment. The hot threads section shows you what your individual elastic search nodes are up to.

The entire display updates in real time as new information is retrieved.

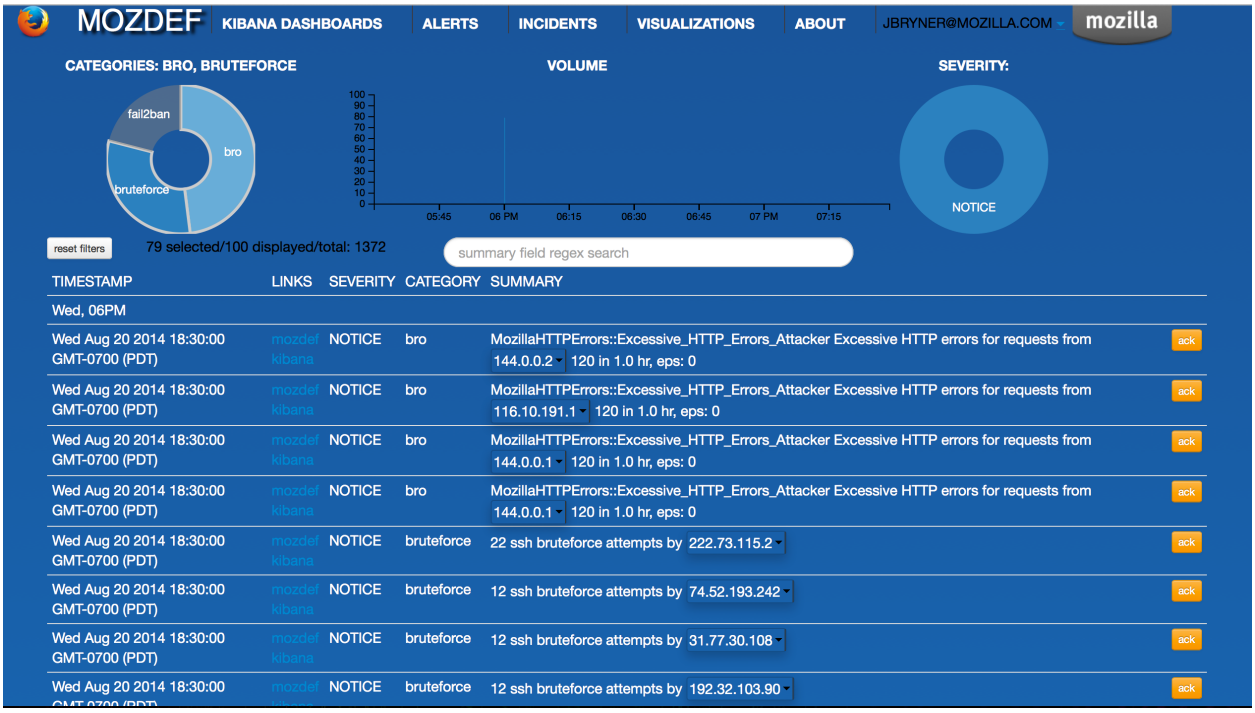



## Alerts

Alerts are simply python jobs run as celery tasks that query elastic search for either individual events, or correlate multiple events into an alert.

The alerts screen shows the latest 100 alerts and allows interactive filtering by category, severity, time frame and free-form regex.

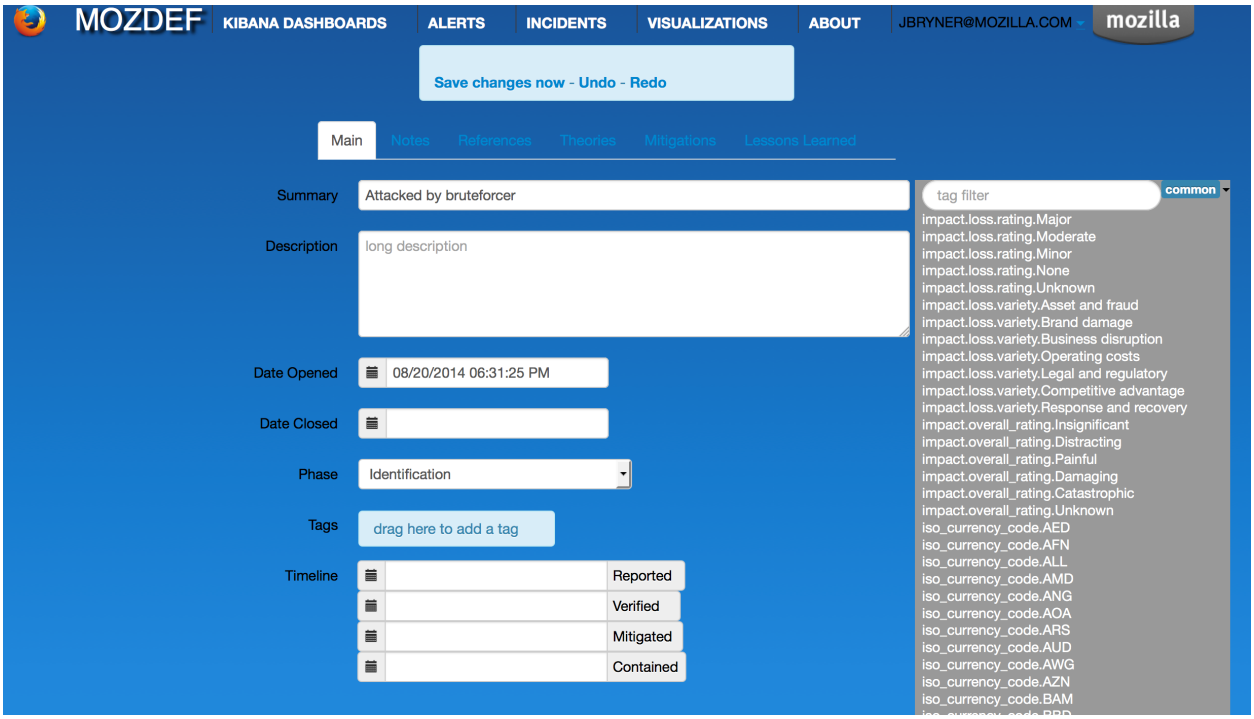
The display updates in real time as new alerts are received and any IP address in an alert is decorated with a menu allowing you to query whois, dshield, CIF, etc to get context on the item. If your facilities include blocking, you can also integrate that into the menu to allow you to block an IP directly from this screen.




## Incident Handling

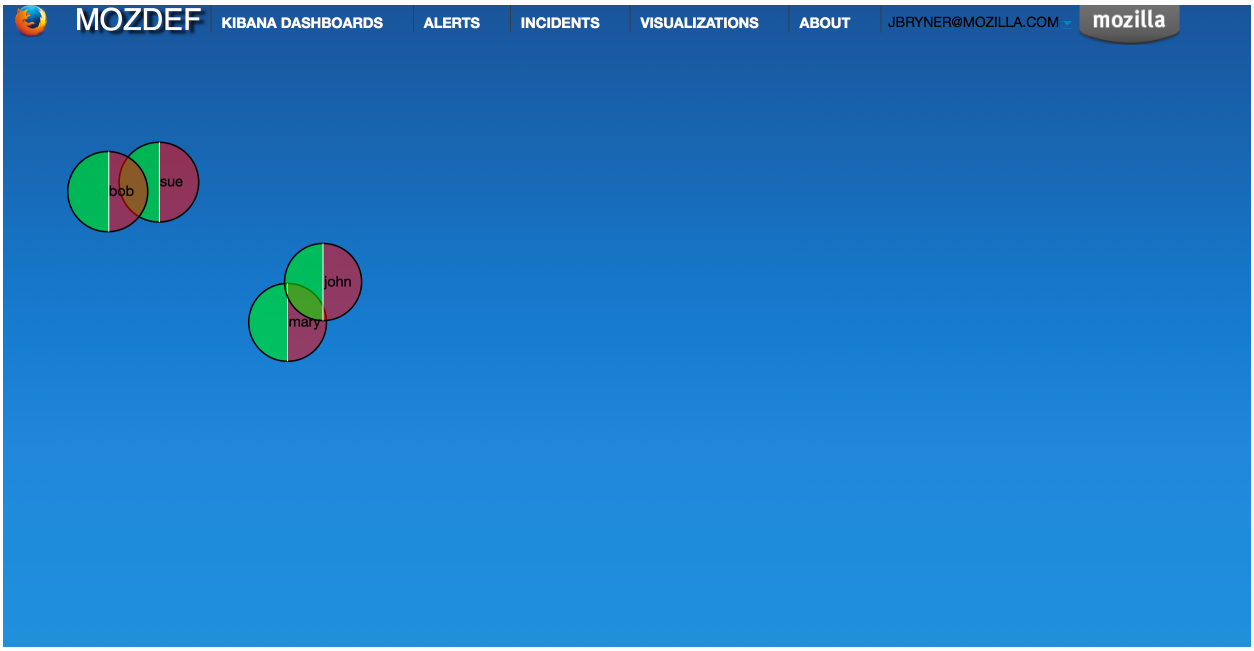
MozDef includes an integrated, real time incident handling facility that allows multiple responders to work collaboratively on a security incident. As they add information to the incident they are able to see each others changes as they happen, in real time.

MozDef includes integration into the VERIS classification system to quickly tag incidents with metadata by dragging tags onto the incident which allows you to aggregate metrics about your incidents.



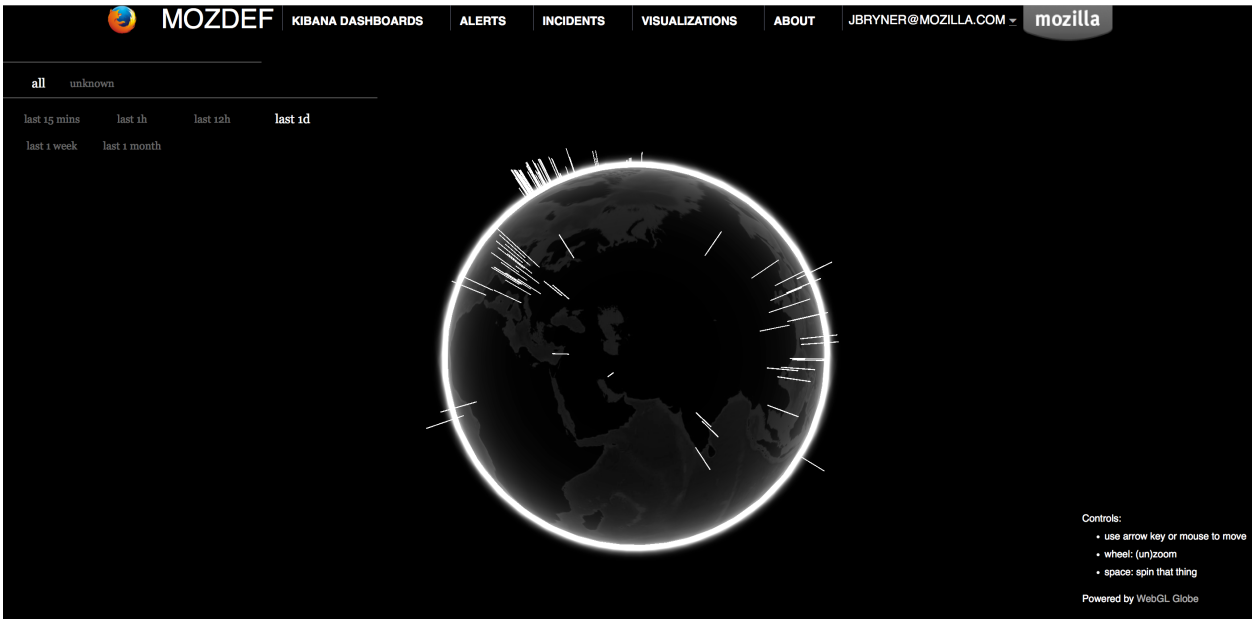

## d3 visualizations

The d3.js library is included in MozDef to allow you custom visualizations of your data. The is a sample visualization of login counts (success vs failed) that you can integrate into your central authentication directory for quick context into user activity.



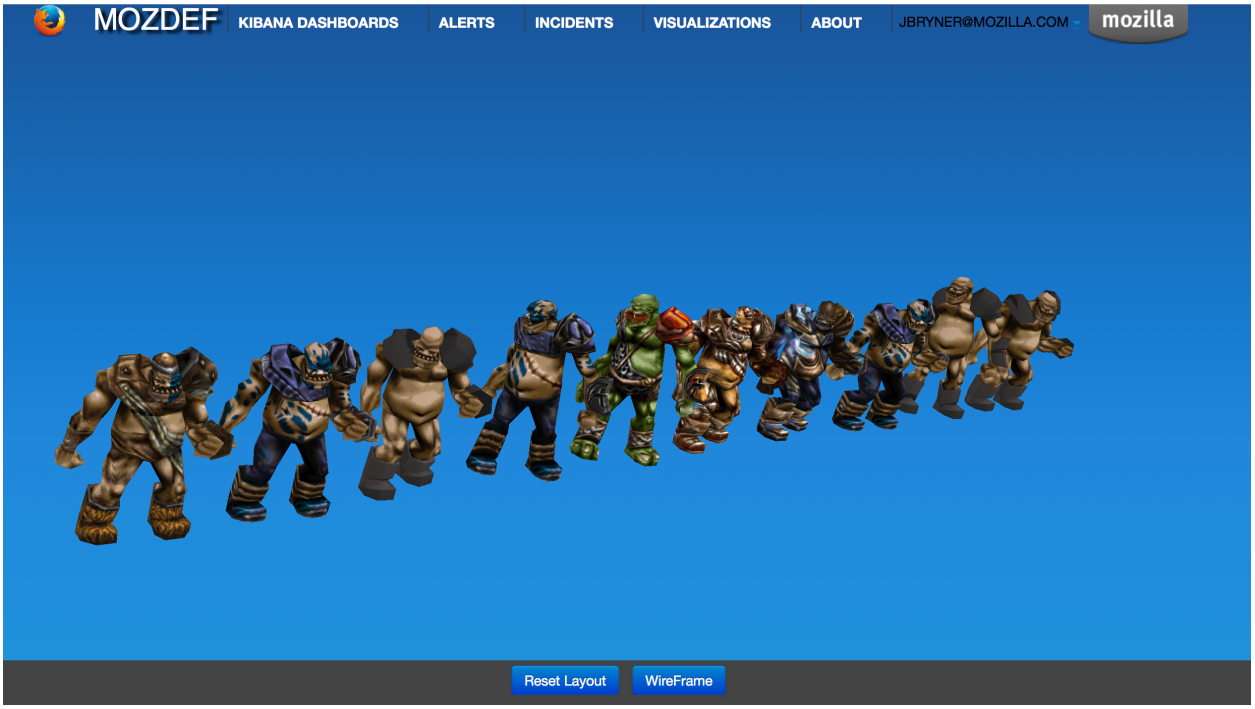
## Geo location of Attackers

MozDef includes the WebGL globe as a three.js visualization that geolocates attackers to give you quick, interactive context about threat actors.



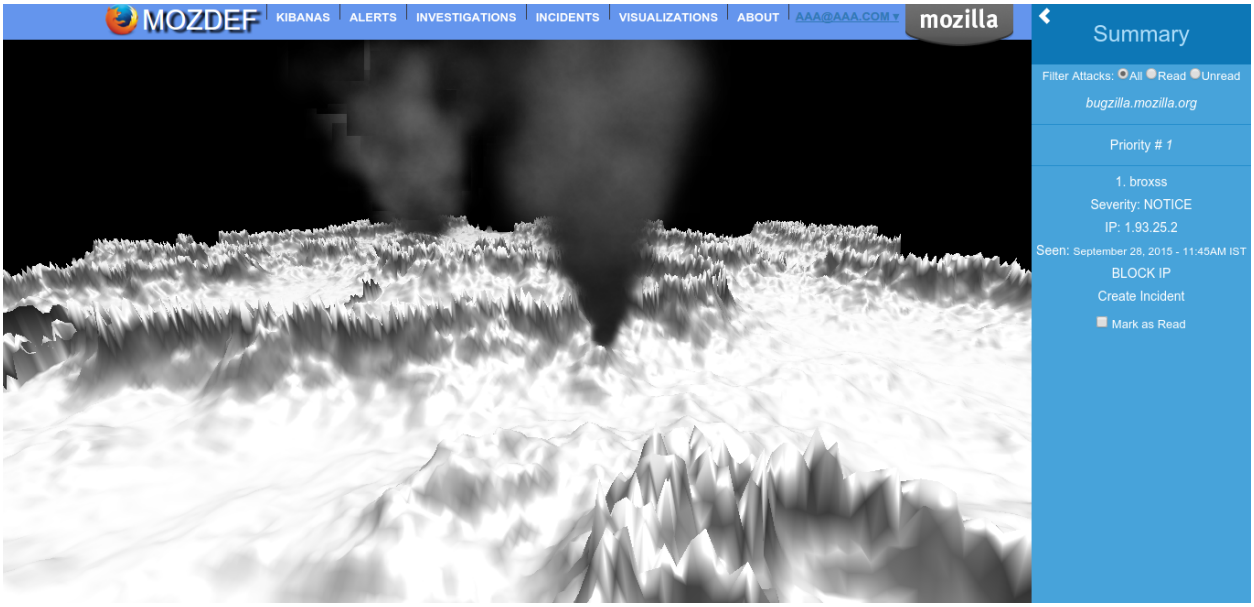

## 3D interactive Attacker visualization

MozDef correlates alerts and events into a 3D visual representation of attackers as ogres. You can use this to quickly filter attackers by category or timeframe and get easy access to recent alerts and events from attackers in 3D.



## 3D interactive Attack visualization via Landmass

MozDef has a service-oriented visualization where you will get see various animations on a landmass service wise. There are also options for handling attacks, and a sidebar which gives you detailed info into the attacks



CHAPTER 6

# Usage

## Web Interface

MozDef uses the Meteor framework for the web interface and bottle.py for the REST API. For authentication, MozDef supports local account creation. Meteor (the underlying UI framework) supports many authentication options including google, github, twitter, facebook, oath, native accounts, etc.

### Events visualizations

Since the backend of MozDef is Elastic Search, you get all the goodness of Kibana with little configuration. The MozDef UI is focused on incident handling and adding security-specific visualizations of SIEM data to help you weed through the noise.

### Alerts

Alerts are implemented as Elastic Search searches. MozDef provides a plugin interface to allow open access to event data for enrichment, hooks into other systems, etc.

### Incident handling

## Sending logs to MozDef

Events/Logs are accepted as json over http(s) with the POST or PUT methods or over rabbit-mq. Most modern log shippers support json output. MozDef is tested with support for:

- heka
- beaver
- nxlog

- logstash
- native python code
- AWS cloudtrail (via native python)

We have some configuration snippets

## What should I log?

If your program doesn't log anything it doesn't exist. If it logs everything that happens it becomes like the proverbial boy who cried wolf. There is a fine line between logging too little and too much but here is some guidance on key events that should be logged and in what detail.

| Event | Example | Rationale |
|---|---|---|
| Authentication Events | Failed/Success logins | Authentication is always an important event to log as it establishes traceability for later events and allows correlation of user actions across systems. |
| Authorization Events | Failed attempts to insert/update/delete a record or access a section of an application. | Once a user is authenticated they usually obtain certain permissions. Logging when a user's permissions do not allow them to perform a function helps troubleshooting and can also be helpful when investigating security events. |
| Account Lifecycle | Account creation/deletion/update | Adding, removing or changing accounts are often the first steps an attacker performs when entering a system. |
| Password/Key Events | Password changed, expired, reset. Key expired, changed, reset. | If your application takes on the responsibility of storing a user's password (instead of using a centralized source) it is important to note changes to a users credentials or crypto keys. |
| Account Activations | Account lock, unlock, disable, enable | If your application locks out users after failed login attempts or allows for accounts to be inactivated, logging these events can assist in troubleshooting access issues. |
| Application Exceptions | Invalid input, fatal errors, known bad things | If your application catches errors like invalid input attempts on web forms, failures of key components, etc creating a log record when these events occur can help in troubleshooting and tracking security patterns across applications. Full stack traces should be avoided however as the signal to noise ratio is often overwhelming.<br>It is also preferable to send a single event rather than a multitude of events if it is possible for your application to correlate a significant exception.<br>For example, some systems are notorious for sending a connection event with source IP, then sending an authentication event with a session ID then later sending an event for invalid input that doesn't include source IP or session ID or username. Correctly correlating these events across time is much more difficult than just logging all pieces of information if it is available. |

# JSON format

This section describes the structure JSON objects to be sent to MozDef. Using this standard ensures developers, admins, etc are configuring their application or system to be easily integrated into MozDef.

## Background

Mozilla used CEF as a logging standard for compatibility with Arcsight and for standardization across systems. While CEF is an admirable standard, MozDef prefers JSON logging for the following reasons:

- Every development language can create a JSON structure
- JSON is easily parsed by computers/programs which are the primary consumer of logs
- CEF is primarily used by Arcsight and rarely seen outside that platform and doesn't offer the extensibility of JSON
- A wide variety of log shippers (heka, logstash, fluentd, nxlog, beaver) are readily available to meet almost any need to transport logs as JSON.
- JSON is already the standard for cloud platforms like amazon's cloudtrail logging

## Description

As there is no common RFC-style standard for json logs, we prefer the following structure adapted from a combination of the graylog GELF and logstash specifications.

Note all fields are lowercase to avoid one program sending sourceIP, another sending sourceIp, another sending SourceIPAddress, etc. Since the backend for MozDef is elasticsearch and fields are case-sensitive this will allow for easy compatibility and reduce potential confusion for those attempting to use the data. MozDef will perform some translation of fields to a common schema but this is intended to allow the use of heka, nxlog, beaver and retain compatible logs.

## Mandatory Fields

| Field | Purpose | Sample Value |
|---|---|---|
| category | General category/type of event matching the 'what should I log' section below | Authentication, Authorization, Account Creation, Shutdown, Startup, Account Deletion, Account Unlock, brointel, bronotice |
| details | Additional, event-specific fields that you would like included with the event. Please completely spell out a field rather an abbreviate: i.e. sourceipaddress instead of srcip. | "dn": "john@example.com,o=com, dc=example", "facility": "daemon" |
| hostname | The fully qualified domain name of the host sending the message | server1.example.com |
| processid | The PID of the process sending the log | 1234 |
| processname | The name of the process sending the log | myprogram.py |
| severity | RFC5424 severity level of the event in all caps: DEBUG, INFO, NOTICE, WARNING, ERROR, CRITICAL, ALERT, EMERGENCY | INFO |
| source | Source of the event (file name, system name, component name) | /var/log/syslog/2014.01.02.log |
| summary | Short human-readable version of the event suitable for IRC, SMS, etc. | john login attempts over threshold, account locked |
| tags | An array or list of any tags you would like applied to the event | vpn, audit<br>nsm,bro,intel |
| timestamp | Full date plus time timestamp of the event in ISO format including the timezone offset | 2014-01-30T19:24:43+06:00 |
| utctimestamp | Full UTC date plus time timestamp of the event in ISO format including the timezone offset | 2014-01-30T13:24:43+00:00 |
| receivedtimestamp | Full UTC date plus time timestamp of the event in ISO format when mozdef received the event | 2014-01-30T13:24:43+00:00 |

## Details substructure (mandatory if such data is sent, otherwise optional)

| Field | Purpose | Sample Value |
|---|---|---|
| destinationipaddress | Destination IP of a network flow | 8.8.8.8 |
| destinationport | Destination port of a network flow | 80 |
| sourceipaddress | Source IP of a network flow | 8.8.8.8 |
| sourceport | Source port of a network flow | 42297 |
| sourceuri | Source URI such as a referer | https://www.mozilla.org/ |
| destinationuri | Destination URI as in "wget this URI" | https://www.mozilla.org/ |
| error | Action resulted in an error or failure | true/false |
| username | Username, email, login, etc. | kang@mozilla.com |
| useragent | Program agent string | curl/1.76 (Windows; 5.1) |

### Examples

```
{
    "timestamp": "2014-02-14T11:48:19.035762739-05:00",
    "hostname": "somemachine.in.your.company.com",
    "processname": "/path/to/your/program.exe",
    "processid": 3380,
    "severity": "INFO",
    "summary": "joe login failed",
    "category": "authentication",
    "source": "ldap",
    "tags": [
        "ldap",
        "adminAccess",
        "failure"
    ],
    "details": {
        "username": "joe",
        "task": "access to admin page /admin_secret_radioactiv",
        "result": "10 authentication failures in a row"
    }
}
```

## Writing alerts

Alerts allow you to create notifications based on events stored in elasticsearch. You would usually try to aggregate and correlate events that are the most severe and on which you have response capability. Alerts are stored in the alerts folder.

There are two types of alerts:

- simple alerts that consider events on at a time. For example you may want to get an alert everytime a single LDAP modification is detected.
- aggregation alerts allow you to aggregate events on the field of your choice. For example you may want to alert when more than 3 login attempts failed for the same username.

You'll find documented examples in the alerts folder.

Once you've written your alert, you need to configure it in celery to be launched periodically. If you have a `AlertBruteforceSsh` class in a `alerts/bruteforce_ssh.py` file for example, in `alerts/lib/config` you can configure the task to run every minute:

```
ALERTS = {
        'bruteforce_ssh.AlertBruteforceSsh': crontab(minute='*/1'),
}
```

CHAPTER 7

# Advanced Settings

## Conf files

MozDef python scripts in almost all cases expect to be given a -c path/to/file.conf command line option to specify configuration/run time options.

These files all follow the same format:

```
[options]
setting1=value1
setting2=value2
```

All programs do their best to set reasonable, sane defaults and most will run fine without a conf file. By default programname.py will look for programname.conf as it's configuration file so if you follow that convention you don't even need to specify the -c path/to/file.conf option.

### Special Config Items

Here are some tips for some key settings:

```
[options]
esservers=http://server1:9200,http://server2:9200,http://server3:9200
```

is how you can specify servers in your elastic search cluster.

```
[options]
backup_indices = intelligence,.kibana,alerts,events,complianceitems,.jsp,.marvel-
↪kibana,vulnerabilities
backup_dobackup = 1,1,1,1,1,1,1,1
backup_rotation = none,none,monthly,daily,none,none,none,none
backup_pruning = 0,0,0,20,0,0,0,0
```

is how you would configure the backupSnapshot.py and pruneIndexes.py programs to backup selected elastic search indexes, rotate selected indexes and prune certain indexes at selected intervals. In the case above we are backing up all indexes mentioned, rotating alerts monthly, rotating events daily and pruning events indices after 20 days.

```
[options]
autocategorize = True
categorymapping = [{"bruteforce":"bruteforcer"},{"nothing":"nothing"}]
```

is how you would configure collectAttackers.py to do autocategoization of attackers that it discovers and specify a list of mappings matching alert categories to attacker category.

## Myo with TLS/SSL

MozDef supports the Myo armband to allow you to navigate the attackers scene using gestures. This works fine if meteor is hosted using http WITHOUT TLS/SSL as the browser will allow you to connect to the server and to the Myo connect which runs a local webserver at http://127.0.0.1:10138 by default. The browser makes a websocket connection to Myo connect and everyone is happy.

When hosting MozDef/Meteor on a TLS/SSL-enabled server things go south quickly. The browser doesn't like (or permit) a https:// hosted page from accessing a plain text websocket resource such as ws://127.0.0.1:10138.

Luckily you can use nginx to work around this.

On you local workstation you can setup a nginx reverse proxy to allow the browser to do TLS/SSL connections, and use nginx to redirect that 127.0.0.1 traffic from TLS to plain text Myo. Here's some configs:

First in mozdef you need to add a myoURL option to settings.js:

```
mozdef = {
  rootURL: "http://yourserver",
  port: "3000",
  rootAPI: "https://yourserver:8444/",
  enableBlockIP: true,
  kibanaURL: "http://yourkibanaserver:9090",
  myoURL: "wss://127.0.0.1:8444/myo/"
}
```

This tells MozDef to initialize Myo using a local TLS connection to port 8444.

Now install nginx and set a nginx.conf file like so:

```
http {
    include       mime.types;
    default_type  application/octet-stream;
    ssl_session_cache   shared:SSL:10m;
    ssl_session_timeout 10m;
    ssl_certificate /path/to/localhost.crt;
    ssl_certificate_key /path/to/localhost.key;

    sendfile        on;
    keepalive_timeout  65;

    proxy_headers_hash_max_size 51200;
    proxy_headers_hash_bucket_size 6400;
    ##ssl version of myo connect##
    server{
        listen  *:8444 ssl;
        #access_log /dev/null main;
```

```
        location /{
            proxy_pass http://127.0.0.1:10138;
            proxy_read_timeout 90;
            # WebSocket support (nginx 1.4)
            proxy_http_version 1.1;
            proxy_set_header Upgrade $http_upgrade;
            proxy_set_header Connection "upgrade";
            proxy_redirect     default;
        }
    }
}
```

You'll need a SSL certificate that your browser trusts, you can issue a self-signed one and accept it by just browsing to https://127.0.0.1:8443 and accept the cert if necessary.

Start up MozDef, start up your Myo and enjoy!

CHAPTER 8

# Code

## Plugins

Plugins are supported in several places: Event Processing and the REST api.

### Event Processing

The front-end event processing portion of MozDef supports python plugins to allow customization of the input chain. Plugins are simple python modules than can register for events with a priority, so they only see events with certain dictionary items/values and will get them in a predefined order.

To create a plugin, make a python class that presents a registration dictionary and a priority as follows:

```
class message(object):
    def __init__(self):
        '''register our criteria for being passed a message
           as a list of lower case strings or values to match with an event's␣
↪dictionary of keys or values
           set the priority if you have a preference for order of plugins to run.
           0 goes first, 100 is assumed/default if not sent
        '''
        self.registration = ['sourceipaddress', 'destinationipaddress']
        self.priority = 20
```

#### Message Processing

To process a message, define an onMessage function within your class as follows:

```
def onMessage(self, message, metadata):
    #do something interesting with the message or metadata
    return (message, metadata)
```

The plugin will receive a copy of the incoming event as a python dictionary in the 'message' variable. The plugin can do whatever it wants with this dictionary and return it to MozDef. Plugins will be called in priority order 0 to 100 if the incoming event matches their registration criteria. i.e. If you register for sourceipaddress you will only get events containing the sourceipaddress field.

If you return the message as None (i.e. message=None) the message will be dropped and not be processed any further. If you modify the metadata the new values will be used when the message is posted to elastic search. You can use this to assign custom document types, set static document _id values, etc.

### Plugin Registration

Simply place the .py file in the plugins directory where the esworker.py is located, restart the esworker.py process and it will recognize the plugin and pass it events as it sees them.

## REST Plugins

The REST API for MozDef also supports python plugins which allow you to customize your handling of API calls to suit your environment. Plugins are simple python modules than can register for REST endpoints with a priority, so they only see calls for that endpoint and will get them in a predefined order.

To create a REST API plugin simply create a python class that presents a registration dictionary and priority as follows:

```
class message(object):
    def __init__(self):
        '''register our criteria for being passed a message
           as a list of lower case strings to match with an rest endpoint
           (i.e. blockip matches /blockip)
           set the priority if you have a preference for order of plugins
           0 goes first, 100 is assumed/default if not sent

           Plugins will register in Meteor with attributes:
           name: (as below)
           description: (as below)
           priority: (as below)
           file: "plugins.filename" where filename.py is the plugin code.

           Plugin gets sent main rest options as:
           self.restoptions
           self.restoptions['configfile'] will be the .conf file
           used by the restapi's index.py file.

        '''

        self.registration = ['blockip']
        self.priority = 10
        self.name = "Banhammer"
        self.description = "BGP Blackhole"
```

The registration is the REST endpoint for which your plugin will receive a copy of the request/response objects to use or modify. The priority allows you to order your plugins if needed so that they operate on data in a defined pattern. The name and description are passed to the Meteor UI for use in dialog boxes, etc so the user can make choices when needed to include/exclude plugins. For example the /blockip endpoint allows you to register multiple methods of blocking an IP to match your environment: firewalls, BGP tables, DNS blackholes can all be independently implemented and chosen by the user at run time.

### Message Processing

To process a message, define an onMessage function within your class as follows:

```
def onMessage(self, request, response):
    '''
    request: http://bottlepy.org/docs/dev/api.html#the-request-object
    response: http://bottlepy.org/docs/dev/api.html#the-response-object

    '''
    response.headers['X-PLUGIN'] = self.description
```

It's a good idea to add your plugin to the response headers if it acts on a message to facilitate troubleshooting. Other than that, you are free to perform whatever processing you need within the plugin being sure to return the request, response object once done:

```
return (request, response)
```

### Plugin Registration

Simply place the .py file in the rest/plugins directory, restart the REST API process and it will recognize the plugin and pass it events as it sees them.

CHAPTER 9

# Benchmarking

Performance is important for a SIEM because it's where you want to store, search and analyze all your security events.

You will want it to handle a significant number of new events per second, be able to search quickly and perform fast correlation. Therefore, we provide some benchmarking scripts for MozDef to help you determine the performance of your setup. Performance tuning of elastic search can be complex and we highly recommend spending time tuning your environment.

## Elasticsearch

Elasticsearch is the main backend component of MozDef. We strongly recommend you to have a 3+ nodes cluster to allow recovery and load balancing. During our tests, Elasticsearch recovered well after being pushed to the limits of hardware, loosing and regaining nodes, and a variety of valid/invalid data. We provide the following scripts for you to use to test your own implementation.

The scripts for Elasticsearch benchmarking are in *benchmarking/es/*. They use nodejs to allow asynchronous HTTP requests.

### insert_simple.js

*insert_simple.js* sends indexing requests with 1 log/request.

Usage: *node ./insert_simple.js <processes> <totalInserts> <host1> [host2] [host3] [...]*

- *processes*: Number of processes to spawn
- *totalInserts*: Number of inserts to perform, please note after a certain number node will slow down. You want to have a lower number if you are in this case.
- *host1*, *host2*, *host3*, etc: Elasticsearch hosts to which you want to send the HTTP requests

## insert_bulk.js

*insert_bulk.js* sends bulk indexing requests (several logs/request).

Usage: *node ./insert_bulk.js <processes> <insertsPerQuery> <totalInserts> <host1> [host2] [host3] [...]*

- *processes*: Number of processes to spawn
- *insertsPerQuery*: Number of logs per request
- *totalInserts*: Number of inserts to perform, please note after a certain number node will slow down. You want to have a lower number if you are in this case.
- *host1*, *host2*, *host3*, etc: Elasticsearch hosts to which you want to send the HTTP requests

## search_all_fulltext.js

*search_all_fulltext.js* performs search on all indices, all fields in fulltext. It's very stupid.

Usage: *node ./search_all_fulltext.js <processes> <totalSearches> <host1> [host2] [host3] [...]*

- *processes*: Number of processes to spawn
- *totalSearches*: Number of search requests to perform, please note after a certain number node will slow down. You want to have a lower number if you are in this case.
- *host1*, *host2*, *host3*, etc: Elasticsearch hosts to which you want to send the HTTP requests

CHAPTER 10

# Contributors

Here is the list of the awesome contributors helping us or that have helped us in the past:

- Yohann Lepage (@2xyo) yohann INSERTAT lepage INSERTDOT info (docker configuration)
- Björn Arnelid bjorn.arnelid INSERTAT gmail INSERTDOT com

CHAPTER 11

# Indices and tables

- genindex
- modindex
- search

CHAPTER 12

# License

license

CHAPTER 13

# Contact

- mozdef INSERTAT mozilla.com
- #mozdef
- https://lists.mozilla.org/listinfo/dev-mozdef