MozDef Documentation

Release 0.0.1

Jeff Bryner, Anthony Verez

April 11, 2014

CHAPTER 1

Overview

1.1 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.

1.2 Goals

1.2.1 High level

- Provide a platform for use by defenders to rapidly discover and respond to security incidents.
- Automate interfaces to other systems like bunker, banhammer, mig
- 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

1.2.2 Technical

- Replace a SIEM
- Scalable, should be able to handle thousands of events/s, provide fast searching, alerting and correlations and handle interactions between teams of incident handlers.

MozDef aims to provide traditional SIEM functionality including:

- Accepts events/logs from your systems
- Stores the events/logs
- Facilitate searches

- Facilitate alerting
- Facilitate 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

1.3 Architecture

MozDef is based on open source technologies including:

- Nginx (http(s) based log input)
- Rabbit-MQ (message queue)
- UWSGI (supervisory control of python-based workers)
- bottle.py (simple python interface for web request handling)
- Elastic Search (scalable indexing and searching of JSON documents)
- Meteor (responsive framework for Node.js enabling real-time data sharing)
- Mongo DB (scalable data store, tightly integrated to Meteor)
- VERIS from verizon (open source taxonomy of security incident categorizations)
- d3 (javascript library for data driven documents)
- three.js (javascript library for 3d visualizations)
- Firefox (a snappy little web browser)

1.4 Status

MozDef is in early proof of concept phases at Mozilla where we are using it to replace our current SIEM.

1.5 Roadmap

Near term:

- Replace 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

Mid term:

- Repeatable installation guides
- Ready-made AMIs/downloadable ISOs
- Correlation through machine learning, AI
- 3D visualizations of threat actors
- Base integration into Mozilla's defense mechanisms for automation

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

2.1 Concept of operations

2.1.1 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.

2.1.2 Event Pipeline

The logical flow of events is:

```
                   +-----------+              +--------------+
                   |    MozDef +--------------+              |
+----------+       |    FrontEnd              | Elastic      |
| shipper  +-------+-----------+              | Search       |
++++++++++++                                  | cluster      |
++++++++++++                                  |              |
| shipper  +-------+-----------+              |              |
+----------+       |    MozDef ++-------------+              |
                   |    FrontEnd+             |              |
                   +-----------+              |              |
                                              +--------------+
```

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

2.1.3 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:

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

2.1.4 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.

2.1.5 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

Installation

For the Mozilla setup, please have a look at the MozDef Mana page.

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

3.1 Docker

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

3.1.1 Dockerfile

After installing docker, use this to build a new image:

```
cd docker && sudo make build
```

Running the container:

```
sudo make run
```

You're done! Now go to:

- http://127.0.0.1:3000 < meteor (main web interface)
- http://127.0.0.1:9090 < kibana
- http://127.0.0.1:9200 < elasticsearch
- http://127.0.0.1:9200/_plugin/marvel < marvel (monitoring for elasticsearch)
- http://127.0.0.1:8080 < loginput
- http://127.0.0.1:8081 < rest api

3.1.2 Known issues

- Marvel doesn't display node info: ' Oops! FacetPhaseExecutionException[Facet [fs.total.available_in_bytes]: failed to find mapping for fs.total.available_in_bytes]'

Marvel uses techniques to gather system info that are not compatible with docker. See https://groups.google.com/forum/#!topic/elasticsearch/dhpxaOuoZWI

Despite this issue, marvel runs fine.

- I don't see any data or dashboards in Kibana

We need to create some sample data, it's in our roadmap ;)

3.2 Elasticsearch nodes

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

3.2.1 ElasticSearch

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

3.2.2 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 development but you have to get a license for production.

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

```
bin/plugin -i elasticsearch/marvel/latest
sudo service elasticsearch restart
```

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

3.3 Web and Workers nodes

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

3.3.1 Python

We need to install a python2.7 virtualenv:

```
sudo yum install make zlib-devel bzip2-devel openssl-devel ncurses-devel sqlite-devel readline-devel
cd
wget http://python.org/ftp/python/2.7.6/Python-2.7.6.tgz
tar xvzf http://python.org/ftp/python/2.7.6/Python-2.7.6.tgz
./configure --prefix=/home/mozdef/python2.7 --enable-shared
make
make install

wget https://raw.github.com/pypa/pip/master/contrib/get-pip.py
export LD_LIBRARY_PATH=/home/netantho/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 2.7.6.

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 /home/mozdef/python2.7/lib/

3.3.2 RabbitMQ

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

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

```
yum install erlang
```

You can then install the rabbitmq server:

```
rpm --import http://www.rabbitmq.com/rabbitmq-signing-key-public.asc
yum install rabbitmq-server-3.2.4-1.noarch.rpm
```

To start rabbitmq at startup:

```
chkconfig rabbitmq-server on
```

3.3.3 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. 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
```

For meteor, in a terminal:

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

wget http://nodejs.org/dist/v0.10.26/node-v0.10.26.tar.gz
tar xvzf node-v0.10.26.tar.gz
cd node-v0.10.26
./configure
make
make install
```

Make sure you have meteorite/mrt:

```
npm install -g meteorite
```

Then from the meteor subdirectory of this git repository run:

```
mrt add iron-router
mrt add accounts-persona
```

You may want to edit the app/lib/settings.js file to properly point to your elastic search server:

```
elasticsearch={
  address:"http://servername:9200/",
  healthurl:"_cluster/health",
  docstatsurl:"_stats/docs"
}
```

Then start meteor with:

```
meteor
```

3.3.4 Nginx

We use nginx webserver.

You need to install nginx:

```
sudo yum install 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/centos/6/$basearch/
gpgcheck=0
enabled=1
```

3.3.5 UWSGI

We use uwsgi to interface python and nginx:

```
wget http://projects.unbit.it/downloads/uwsgi-2.0.2.tar.gz
~/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/

cd rest
# 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
```

3.3.6 Kibana

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

```
wget https://download.elasticsearch.org/kibana/kibana/kibana-3.0.0milestone5.tar.gz
tar xvzf kibana-3.0.0milestone5.tar.gz
mv kibana-3.0.0milestone5 kibana
# configure /etc/nginx/nginx.conf to target this folder
sudo service nginx reload
```

Import dashboards from *MozDef/kibana/dashboards* into the kibana webUI

CHAPTER 4

Usage

4.1 Web Interface

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

4.1.1 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.

4.1.2 Alerts

Alerts are generally implemented as Elastic Search searches, or realtime examination of the incoming message queues. MozDef provides a plugin interface to allow open access to event data for enrichment, hooks into other systems, etc.

4.1.3 Incident handling

4.2 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

4.2.1 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 centralized LDAP/persona) 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. 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. 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.

4.3 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.

4.3.1 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

4.3.2 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.

4.3.3 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 nsm,bro,intel
timestamp	Full date plus time timestamp of the event in ISO format including the timezone offset	2014-01-30T19:24:43+00:00

4.3.4 Details substructure (optional fields)

Field	Purpose	Used In	Sample Value
destinationipaddress	Destination IP of a network flow	NSM/Bro/Intel	8.8.8.8
destinationport	Destination port of a network flow	NSM/Bro/Intel	80
dn	Distinguished Name in LDAP, mean unique ID in the ldap hierarchy	event/ldap	john@example.org,o=org, dc=example
filedesc		NSM/Bro/Intel	
filemimetype		NSM/Bro/Intel	
fuid		NSM/Bro/Intel	
result	Result of an event, success or failure	event/ldap	LDAP_SUCCESS
seenindicator	Intel indicator that matched as seen by our system	NSM/Bro/Intel	evil.com/setup.exe
seenindicator_type	Type of intel indicator	NSM/Bro/Intel	HTTP::IN_URL
seenwhere	Where the intel indicator matched (which protocol, which field)	NSM/Bro/Intel	Intel::URL
source	Source of the connection	event/ldap	Mar 19 15:36:25 ldap1 slapd[31031]: conn=6633594 fd=49 ACCEPT from IP=10.54.70.109:23957 (IP=0.0.0.0:389) Mar 19 15:36:25 ldap1 slapd[31031]: conn=6633594 op=0 BIND
sourceipaddress	Source IP of a network flow	NSM/Bro/Intel event/ldap	8.8.8.8
sourceport	Source port of a network flow	NSM/Bro/Intel	42297
sources	Source feed	NSM/Bro/Intel	CIF - need-to-know
success	Auth success	event/ldap	True
uid	Bro connection uid	NSM/Bro/Intel	CZqhEs40odso1tFNx3

4.3.5 Examples

```
{
    "timestamp": "2014-02-14T11:48:19.035762739-05:00",
    "hostname": "fedbox",
    "processname": "/tmp/go-build278925522/command-line-arguments/_obj/exe/log_json",
    "processid": 3380,
    "severity": "INFO",
    "summary": "joe login failed",
    "category": "authentication",
    "source": "",
    "tags": [
        "MySystem",
        "Authentication"
    ],
    "details": {
        "user": "joe",
        "task": "access to admin page /admin_secret_radioactiv",
        "result": "10 authentication failures in a row"
    }
}
```

CHAPTER 5

Advanced Settings

CHAPTER 6

Public API

CHAPTER 7

Code

CHAPTER 8

Benchmarking

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

You probably want it to handle a lot of new messages per second, be able to have a fast reply when you search and have fast correlation. Therefore, we provide some benchmarking scripts for MozDef to help you determine the performance of your setup.

8.1 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 hit too much.

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

8.1.1 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

8.1.2 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

8.1.3 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

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 INSERTPOINT info (docker configuration)

CHAPTER 10

Indices and tables

- *genindex*
- *modindex*
- *search*

CHAPTER 11

License

`license`

CHAPTER 12

Contact

- opsec INSERTAT mozilla.com
- Jeff Bryner, jbryner INSERTAT mozilla.com @0x7eff
- Anthony Verez, averez INSERTAT mozilla.com @netantho