Security automation simplified

An intro to DIY security automation

Moses Schwartz, Security Automation Engineer
moses@box.com | @mosesschwartz

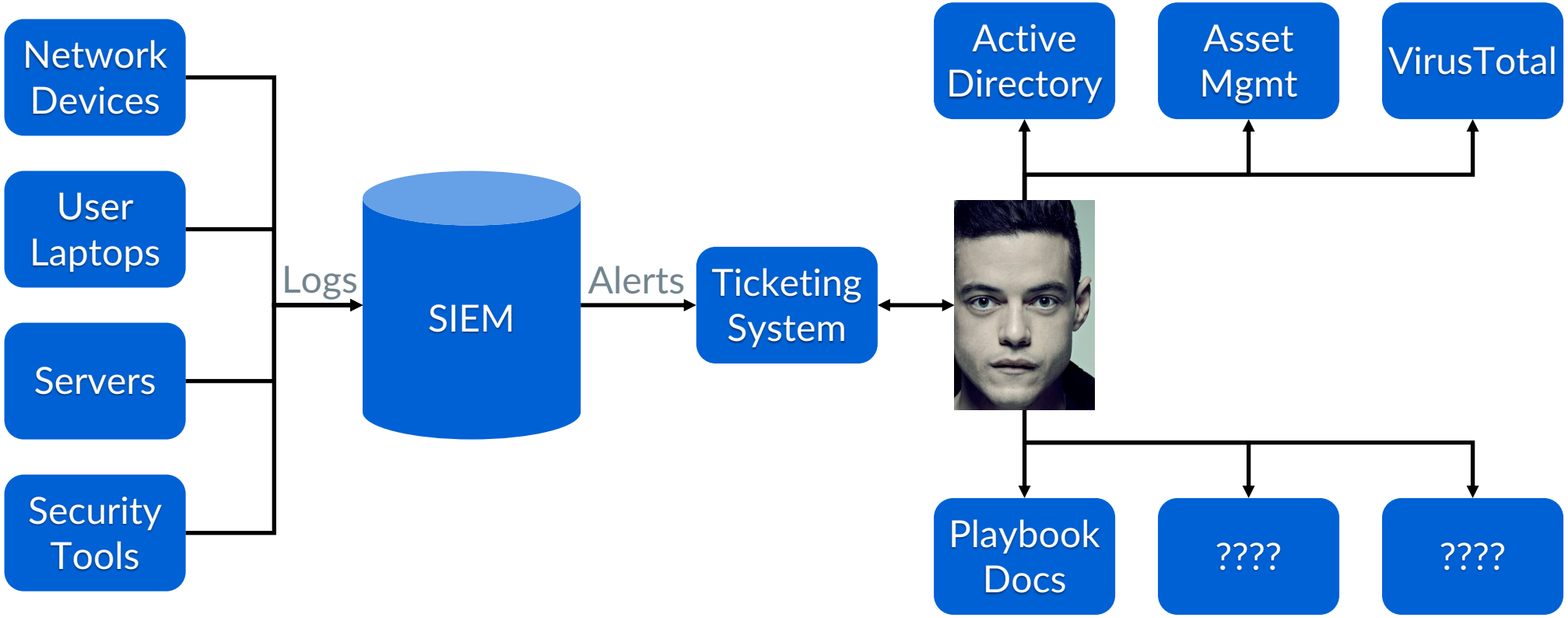
Tristan Waldear, Security Automation Engineer
twaldear@box.com

BSidesSF 2019

Submit questions at **https://sli.do** event code **#BSidesSF2019**

Incident response / security monitoring infrastructure

Before automation



Automation approaches

Building an automation/orchestration solution for every environment is a huge undertaking

Building one for *your* environment is a much more tractable problem

Centralized: Logic mostly in one place, often with some sort of workflow or orchestration engine

- Examples: Most commercial offerings, building everything in Jenkins, StackStorm, or other platform
- Downsides: single point of failure, major effort to build out, major learning curve

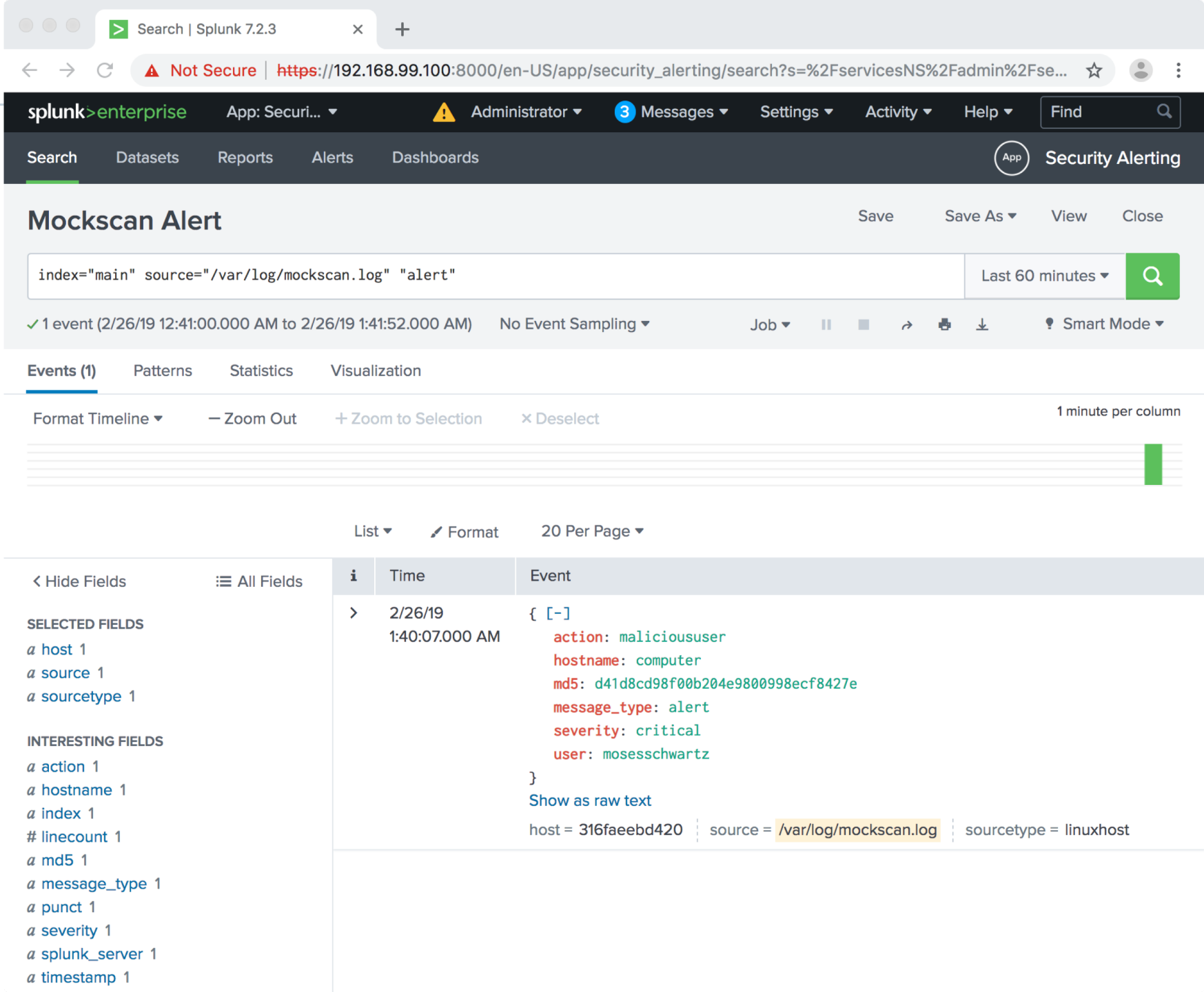
Distributed: Bits of automation from many different tools to make your life better

- Examples: 1:1 pairings using web hooks between tool APIs
- Downsides: gets complicated, hard to debug, hard to piece together complex workflows

Splunk

Alert development

Build a search query that matches the condition you want to alert on



Splunk

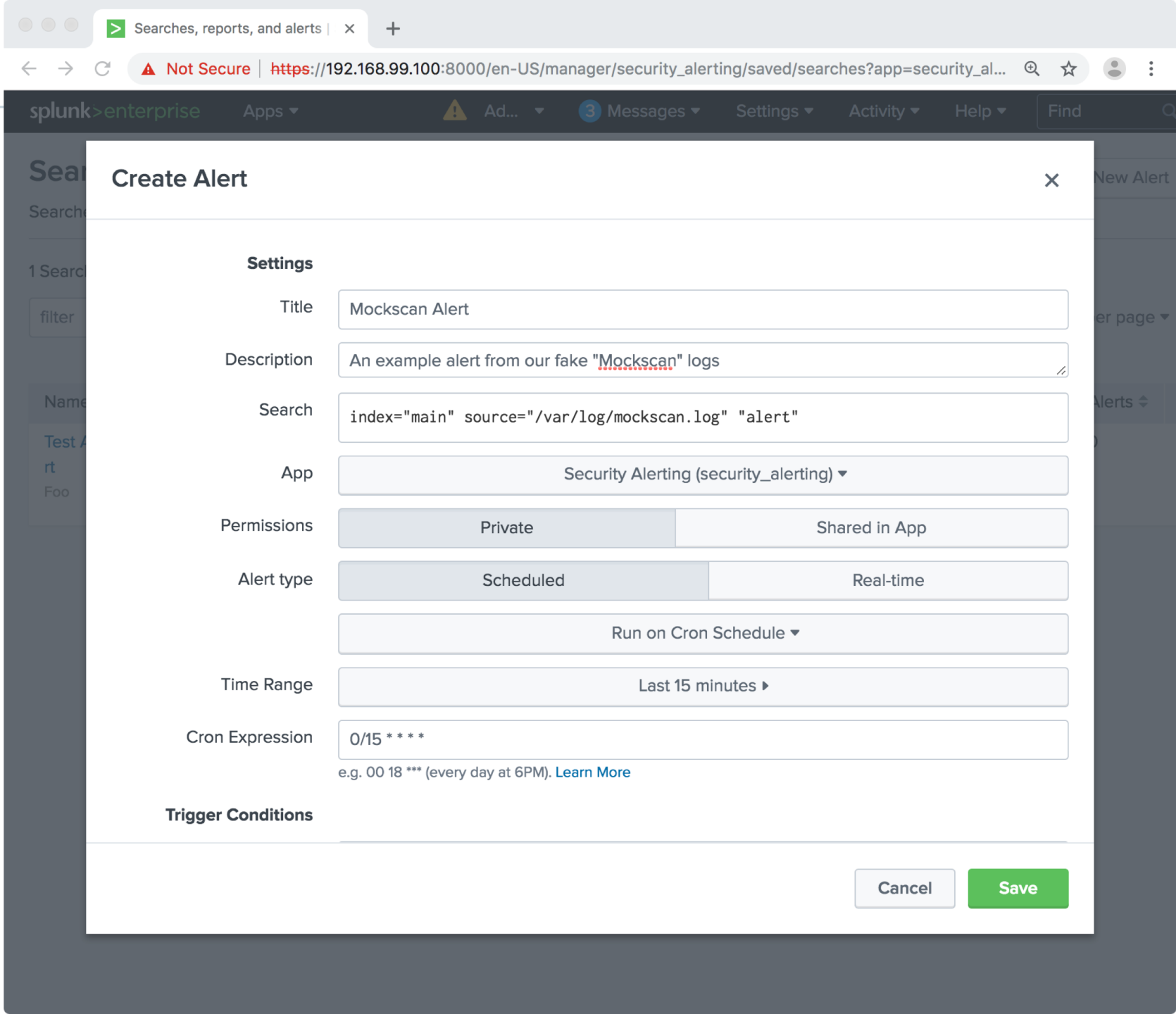
Alert development

Create an alert using your search query

Best practice is to create an App to contain all of your custom settings

I like to specify a cron schedule for maximum flexibility

Ensure your time range matches the schedule



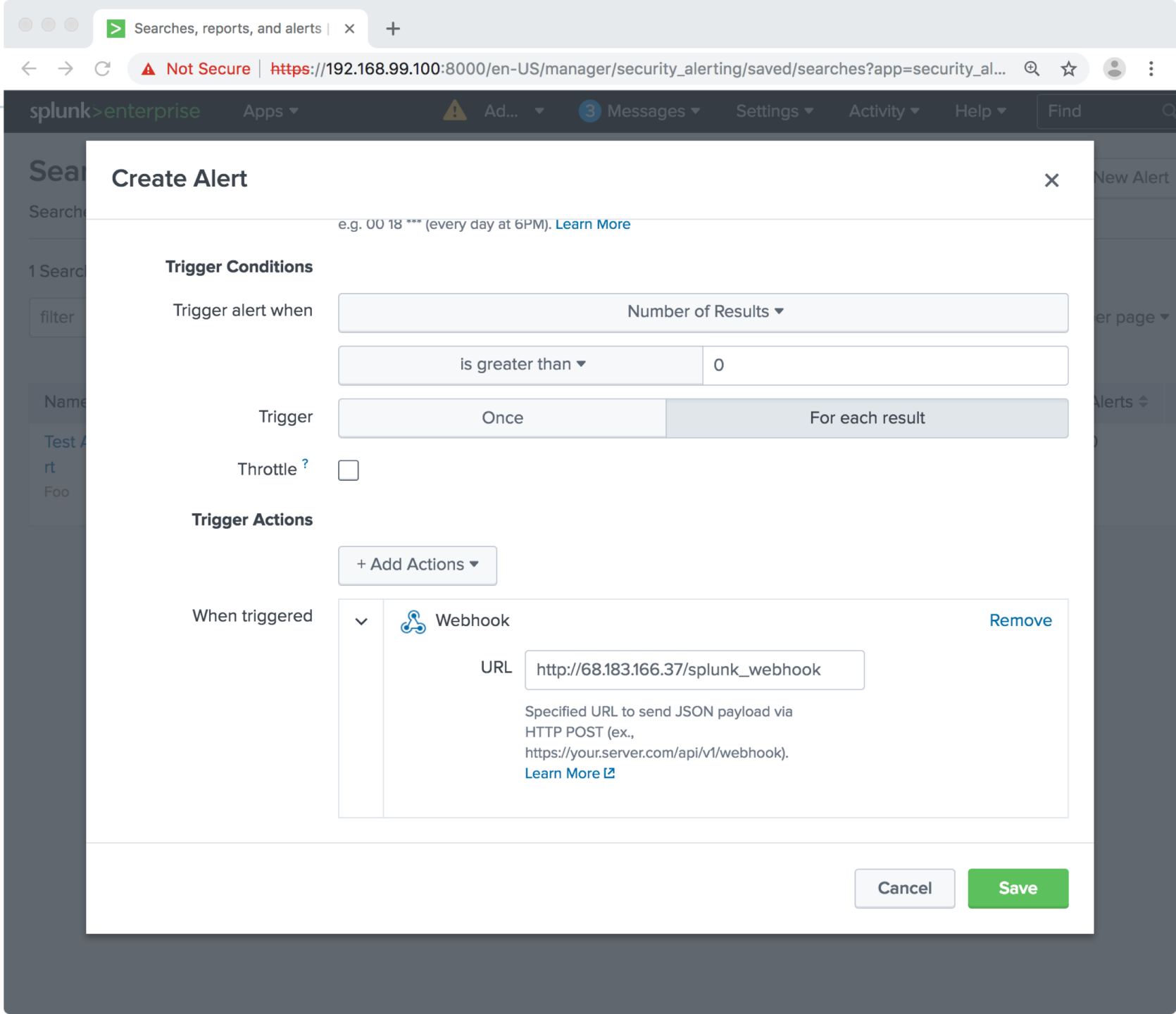
Splunk

Alert development

Trigger when **Number of Results is greater than 0**

Trigger **for each result** - Splunk webhooks only include the first row of data

Add a Webhook Trigger Action and aim it at your server (we'll build this in the next step)



automation_server.py

Receive Splunk webhook payload

We will use **Flask** for super simple API development

Always include a status/health endpoint

splunk_webhook will write the JSON payload with indentation to a file

`host="0.0.0.0"` exposes this to the world!

automation_server_1.py

```python
import json

from flask import Flask, request

app = Flask(__name__)


@app.route("/status")
def status():
    return "up"


@app.route("/splunk_webhook", methods=["POST"])
def splunk_webhook():
    webhook_payload = request.get_json()
    with open("splunk_webhook.json", "w") as f:
        json.dump(webhook_payload, f, indent=4)
    return "success"


app.run(host="0.0.0.0", port=80, debug=False)

```

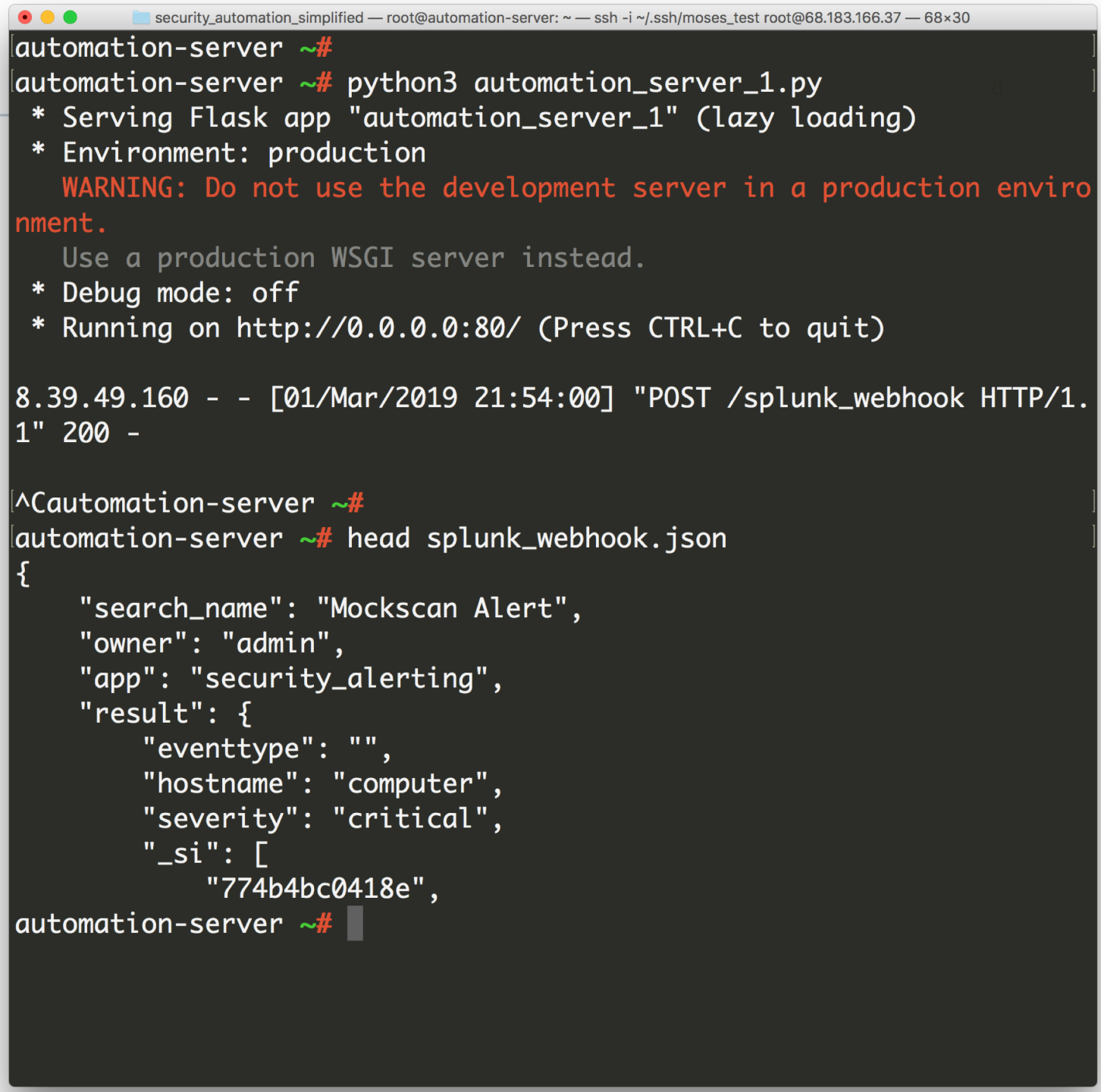
automation_server.py

Run the server and check the output

It's JSON from our alert!

Development tip: modify that alert to run every minute and extend the time range

Don't run it like this in production - there are many tutorials on deploying a Flask app with Nginx or Apache and a WSGI server



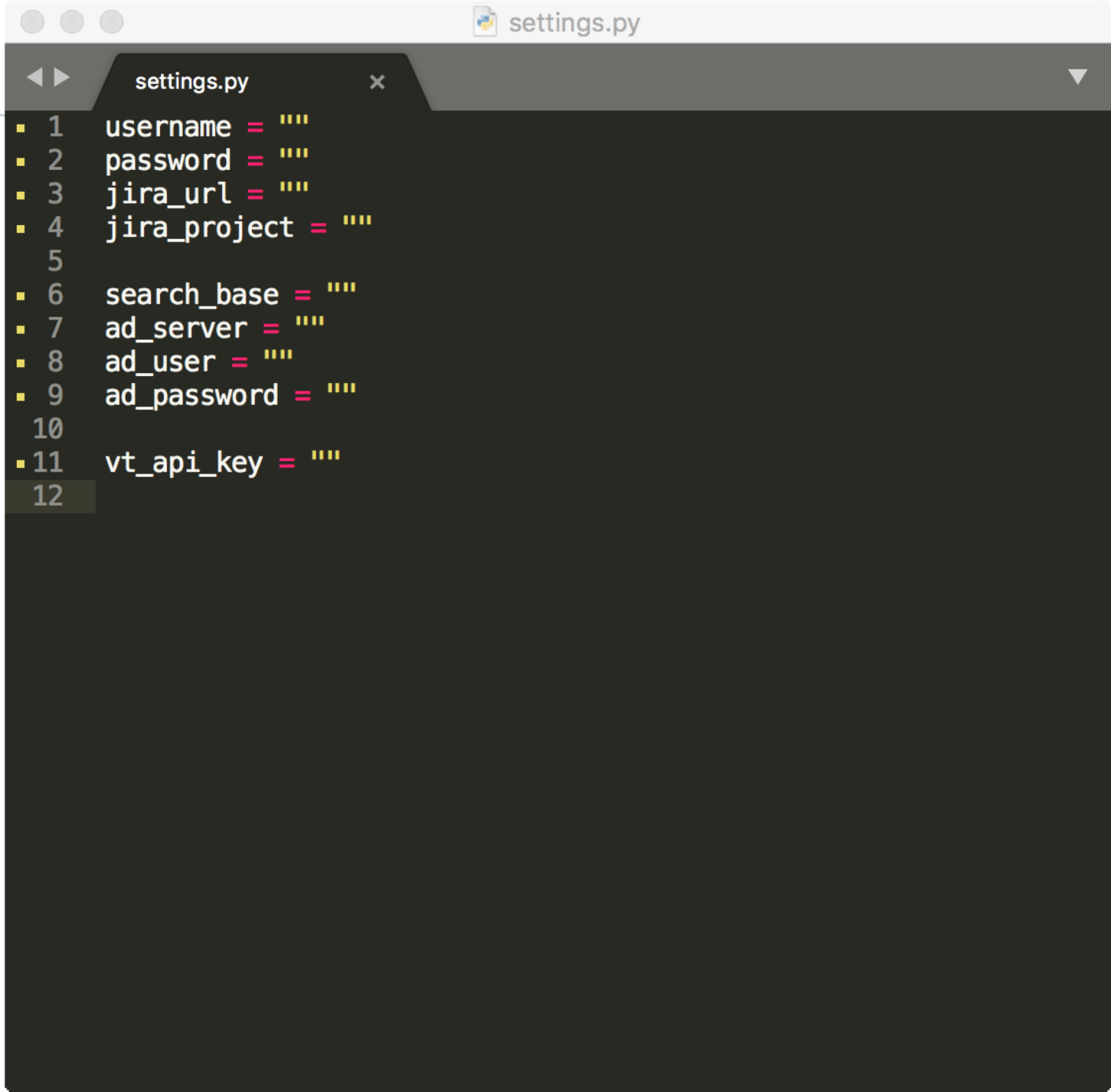
settings.py

Keep secrets out of git!

Your code should be under version control, but your passwords shouldn't!

A super lightweight approach is to keep your secrets and settings in a Python file that is NOT checked in with code (don't forget to add this file to your .gitignore)

This file can then be pushed as part of configuration management or manually



automation_server.py

Round two: ticket creation

Let's create a ticket in Jira

Create your authenticated `JIRA` object using the Python library

Use the `create_issue` method to create the ticket and set fields

automation_server_2.py

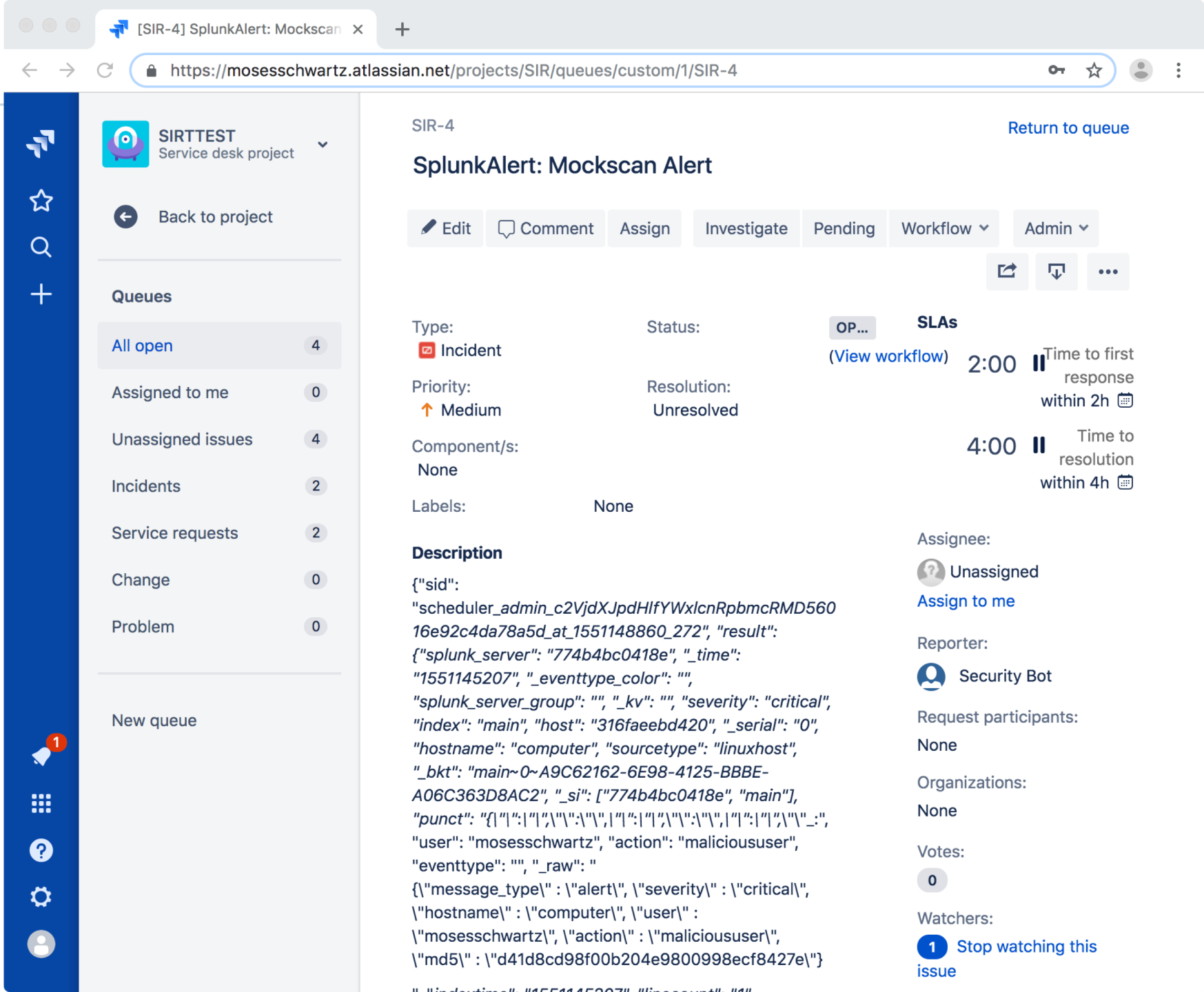
```python
@app.route("/status")
def status():
    return "up"


@app.route("/splunk_webhook", methods=["POST"])
def splunk_webhook():
    webhook_payload = request.get_json()
    ticket = create_jira_issue(
        webhook_payload["search_name"], webhook_payload
    )
    return "success"


def create_jira_issue(alert_name, alert_body):
    j = JIRA(
        settings.jira_url,
        basic_auth=(settings.username, settings.password),
    )
    j.create_issue(
        project=settings.jira_project,
        summary="SplunkAlert: {}".format(alert_name),
        description=json.dumps(alert_body),
        issuetype="Incident",
    )


app.run(host="0.0.0.0", port=80, debug=False)
```

Jira

Issue created

After the next Splunk webhook fires, we'll have a Jira ticket

Right now the description is just a JSON blob of the alert



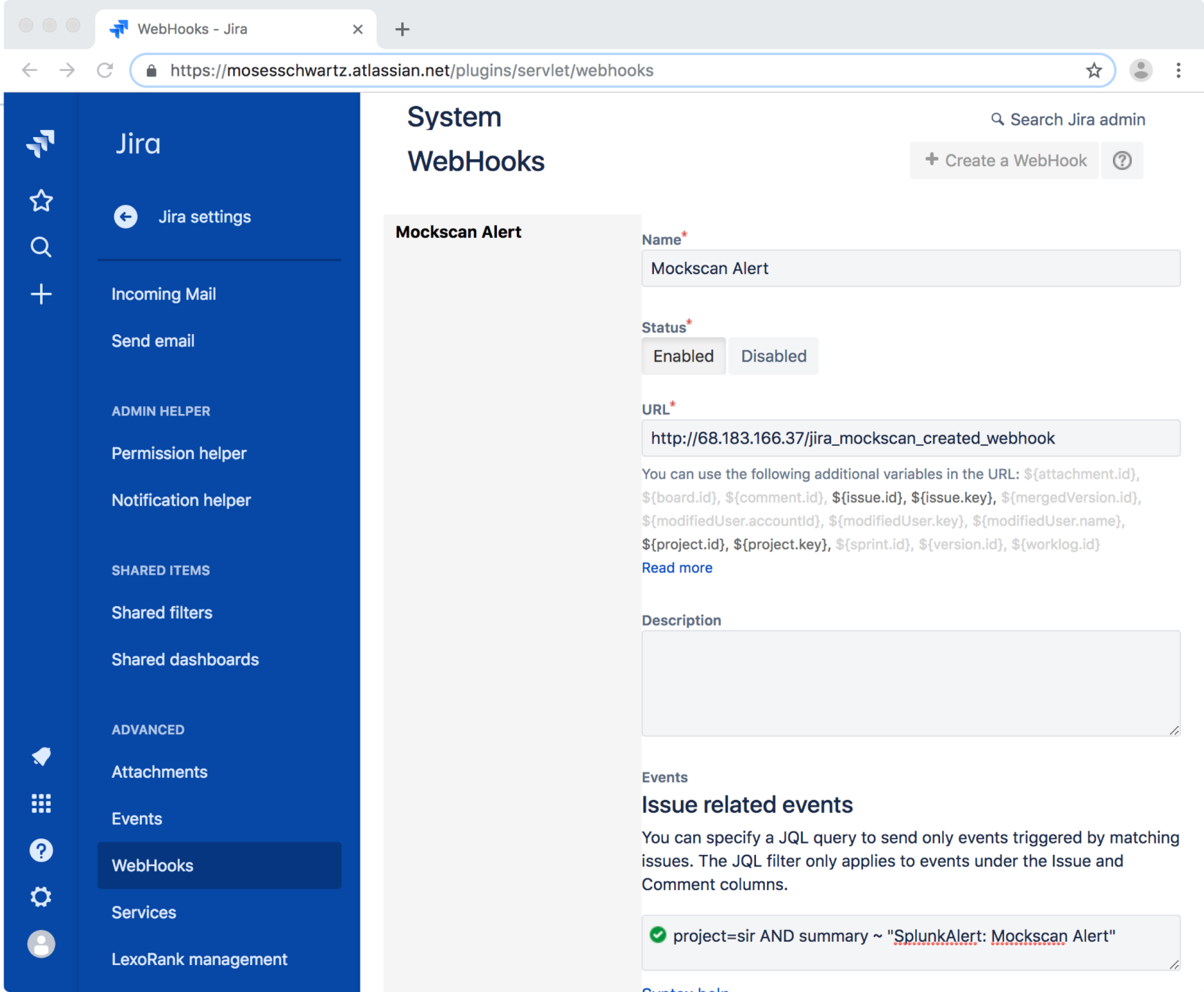
Jira

Webhook configuration

Create a webhook to do enrichments – start by just extracting `user` and `MD5` and commenting on the ticket

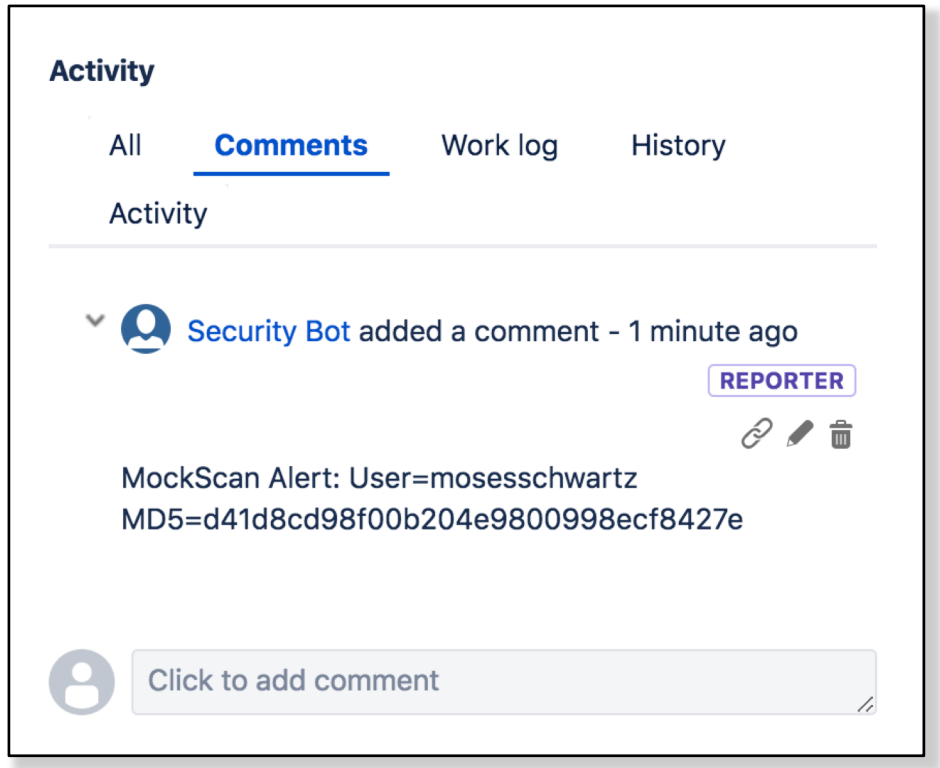
Point the URL to your automation server with a new endpoint

Filter for `Issue created` events that match our project and alert name



automation_server.py

Add a comment to Jira



automation_server_3.py

```python
def status():
    return "up"


@app.route("/jira_mockscan_created_webhook", methods=["POST"])
def mockscan_created():
    jira_webhook = request.get_json()
    jira_desc = jira_webhook["issue"]["fields"]["description"]
    issue_key = jira_webhook["issue"]["key"]
    splunk_alert = json.loads(jira_desc)
    user = splunk_alert["result"]["user"]
    md5 = splunk_alert["result"]["md5"]
    comment = "MockScan Alert: User={} MD5={}".format(user,md5)
    jira_comment(issue_key, comment)
    return "success"


def jira_comment(issue_key, comment):
    j = JIRA(
        settings.jira_url,
        basic_auth=(settings.username, settings.password),
    )
    issue = j.issue(issue_key)
    j.add_comment(issue, comment)


@app.route("/splunk_webhook", methods=["POST"])
def splunk_webhook():
    webhook_payload = request.get_json()
```

ad_lookup.py

Lookup a user in Active Directory

Returns a JSON object:

```
{'entries': [{'attributes': {
    'cn': 'Moses Schwartz',
    'title': 'Staff Security Engineer',
    'company': 'Box, Inc',
    'department': 'Security Automation',
    'employeeID': '1234',
    'l': 'Redwood City',
    'streetAddress': '900 Jefferson Avenue',
    # ... tons more fields omitted
}}]}
```

ad_lookup.py

```python
import json

from ldap3 import (
    Server, Connection, ALL, NTLM, SUBTREE, ALL_ATTRIBUTES
)
import settings


def search_ad(username):
    server = Server(settings.ad_server, get_info=ALL)
    search_filter = "(&(sAMAccountName={}))".format(username)
    conn = Connection(
        server,
        user=settings.ad_user,
        password=settings.ad_password,
        authentication=NTLM,
        auto_referrals=False,
    )
    conn.bind()
    conn.search(
        search_base=settings.search_base,
        search_filter=search_filter,
        search_scope=SUBTREE,
        attributes=ALL_ATTRIBUTES,
        get_operational_attributes=True,
    )
    return json.loads(conn.response_to_json())
```

automation_server.py

Active Directory lookup enrichment

Activity

All | **Comments** | Work log | History

Activity

Security Bot added a comment - Just now

REPORTER

Name: Moses Schwartz
Company: Box, Inc
Dept: Security Automation
Location: Redwood City

Click to add comment

automation_server_4.py

```python
@app.route("/jira_mockscan_created_webhook", methods=["POST"])
def mockscan_created():
    jira_webhook = request.get_json()
    jira_desc = jira_webhook["issue"]["fields"]["description"]
    issue_key = jira_webhook["issue"]["key"]
    splunk_alert = json.loads(jira_desc)
    user = splunk_alert["result"]["user"]
    md5 = splunk_alert["result"]["md5"]
    ad_lookup_enrichment(issue_key, user)
    return "success"


def ad_lookup_enrichment(issue_key, user):
    ad_info = ad_lookup.search_ad(user)
    user_attributes = ad_info["entries"][0]["attributes"]
    comment = "Name: {}\n Company: {}\nDept: {}\nLocation: {}"
    comment = comment.format(
        user_attributes["cn"],
        user_attributes["company"],
        user_attributes["department"],
        user_attributes["l"],
    )
    jira_comment(issue_key, comment)


def jira_comment(issue_key, comment):
    j = JIRA(
```

virustotal.py

Get a file scan report

```
{'scan_id': 'e3b0c44298fc1c149afbf48996f...',
 'sha1': 'da39a3ee5e6b4b0d3255bff9560189...',
 'resource': 'd41d8cd98f00b204e980098ecf...',
 'scan_date': '2019-03-01 23:35:34',
 'permalink': 'https://www.virustotal.com/...
 'total': 60,
 'positives': 0,
 'md5': 'd41d8cd98f00b204e9800998ecf8427e'
 {'scans': {'Bkav': {'detected': False,
   'version': '1.3.0.9899',
   'result': None,
   'update': '20190301'}
   ....
```

virustotal.py

```python
import requests

import settings


def get_file_scan_report(file_hash):
    vt_url = "https://www.virustotal.com/vtapi/v2/file/report"
    params = {
        "apikey": settings.vt_api_key,
        "resource": file_hash,
    }
    response = requests.get(vt_url, params=params)
    return response.json()

```

automation_server.py

Now with AD and VT enrichments

Security Bot added a comment - 1 minute ago

REPORTER

Name: Moses Schwartz
Company: Box, Inc
Dept: Security Automation
Location: Redwood City

Security Bot added a comment - 1 minute ago

REPORTER

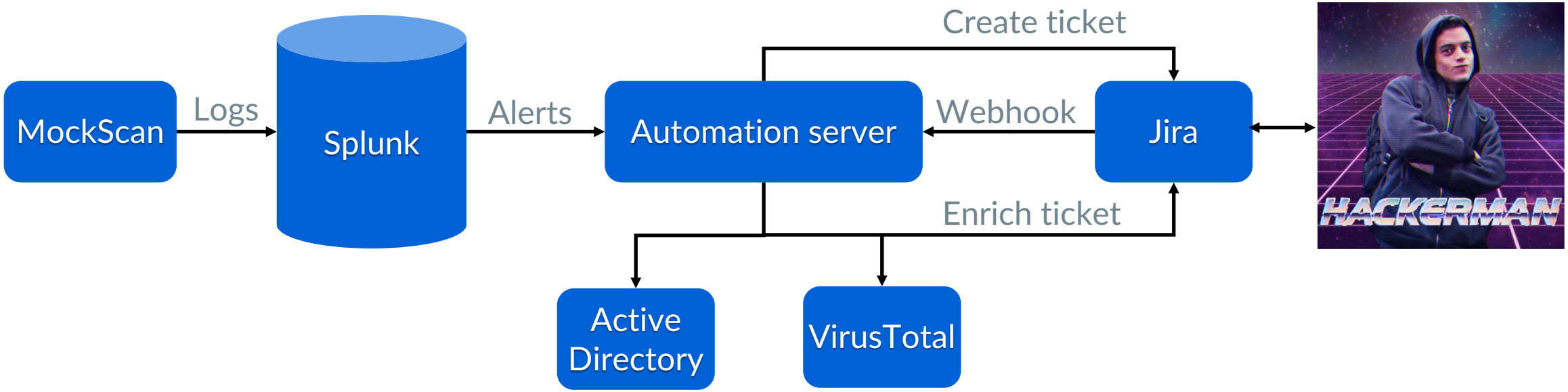
VirusTotal: d41d8cd98f00b204e9800998ecf8427e
Positives: 0/60
Link

automation_server_5.py

```python
@app.route("/jira_mockscan_created_webhook", methods=["POST"])
def mockscan_created():
    jira_webhook = request.get_json()
    jira_desc = jira_webhook["issue"]["fields"]["description"]
    issue_key = jira_webhook["issue"]["key"]
    splunk_alert = json.loads(jira_desc)
    user = splunk_alert["result"]["user"]
    md5 = splunk_alert["result"]["md5"]
    ad_lookup_enrichment(issue_key, user)
    vt_filescan_enrichment(issue_key, md5)
    return "success"


def vt_filescan_enrichment(issue_key, file_hash):
    vt_result = virustotal.get_file_scan_report(file_hash)
    comment = "VirusTotal: {}\nPositives: {}/{}\n[Link|{}]"
    comment = comment.format(
        vt_result["md5"],
        vt_result["positives"],
        vt_result["total"],
        vt_result["permalink"],
    )
    jira_comment(issue_key, comment)


def ad_lookup_enrichment(issue_key, user):
    ad_info = ad_lookup.search_ad(user)
```

Incident response / security monitoring infrastructure

With automation



More things we could automate

Anything you can write a script to do

- Search for and link to previous tickets, populate ticket fields, close duplicate tickets
- Run a Splunk search
- Lookup DNS and WHOIS records
- Run Ansible playbooks
- Send a sample to a sandbox
- Upload files to Box
- Isolate hosts and grab memory
- Pull PCAPs
- Flash a light or connect to other smart devices

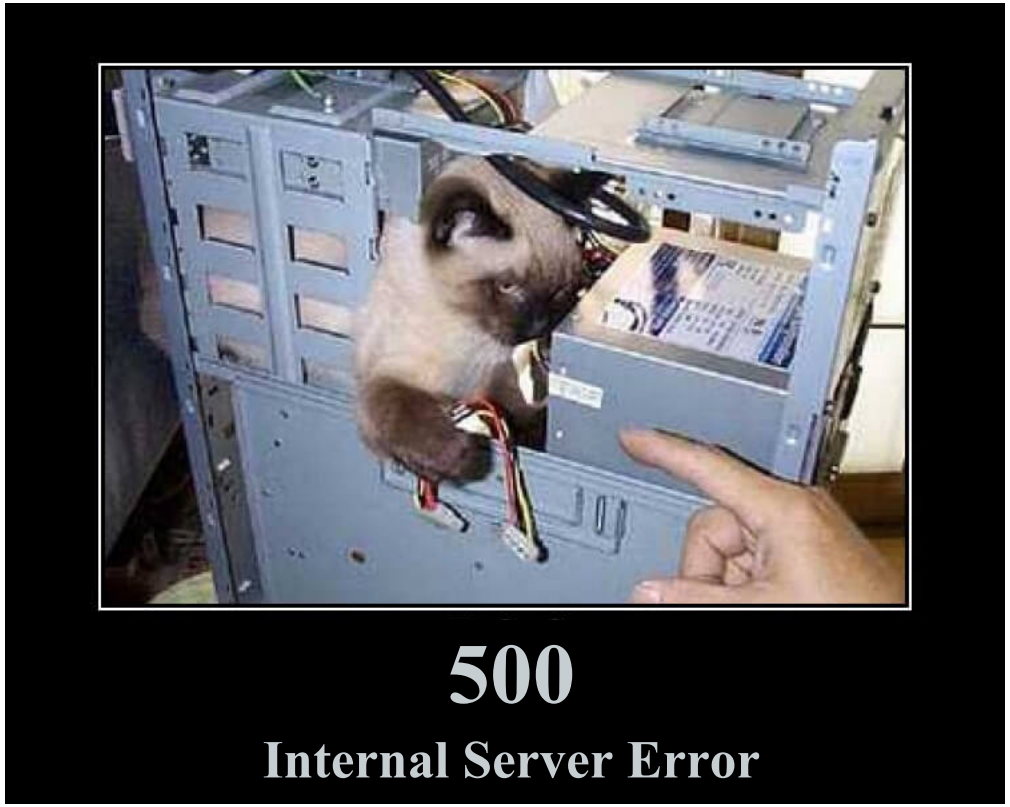
Some considerations

Enrichments and other tasks should be asynchronous to avoid the scenario where if one fails, they all fail

Our example was synchronous, some other options to run jobs asynchronously are:

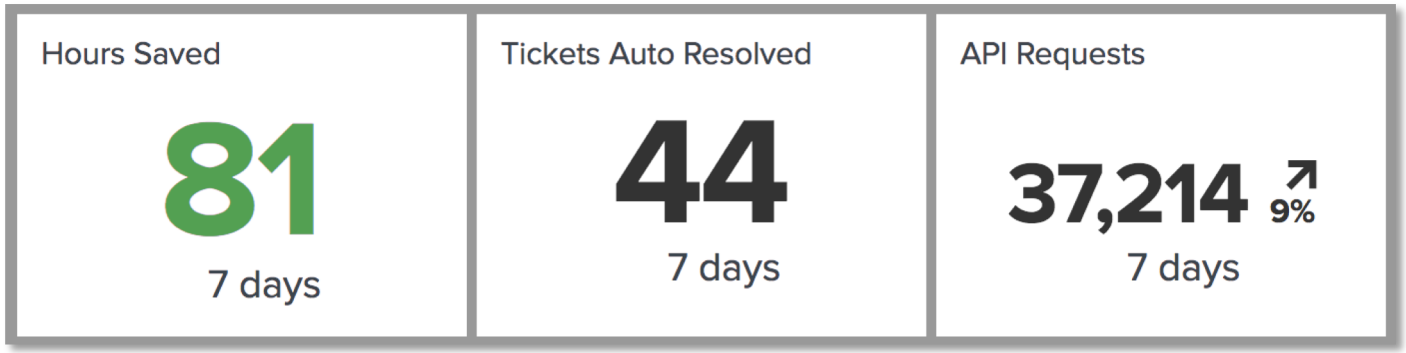
- Individual Jira webhooks for each enrichment
- Celery, asyncio (DIY Python approaches)
- Jenkins, StackStorm (DevOps tools that fit this use case)
- AWS Lambda jobs (hey, we could build this whole thing out of Lambdas)

Exception handling and logging are critical: even if our code is perfect, external API lookups will fail



Metrics

Quantify your impact



Management loves colorful numbers

Assign a number of minutes saved per enrichment or action and calculate the total from your logs

Security automation isn’t about replacing people, and it’s not a set-it-and-forget-it solution

- Security automation is different from factory automation – you can’t replace human incident responders
- Automation should become a core part of your process – continually improve
- Spend at least 25% of your time and effort automating

Takeaways

Security automation is not black magic

Existing tools that aren't marketed toward security can work great in this space

There is so much low hanging fruit

Our job is to make the rest of the team more effective (which is pretty awesome)

This niche is a great path into security from development or into development from security

Security automation simplified

An intro to DIY security automation

Moses Schwartz, Security Automation Engineer
moses@box.com | @mosesschwartz

Tristan Waldear, Security Automation Engineer
twaldear@box.com

BSidesSF 2019

Submit questions at **https://sli.do** event code **#BSidesSF2019**