



Security automation simplified

An intro to DIY security automation

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

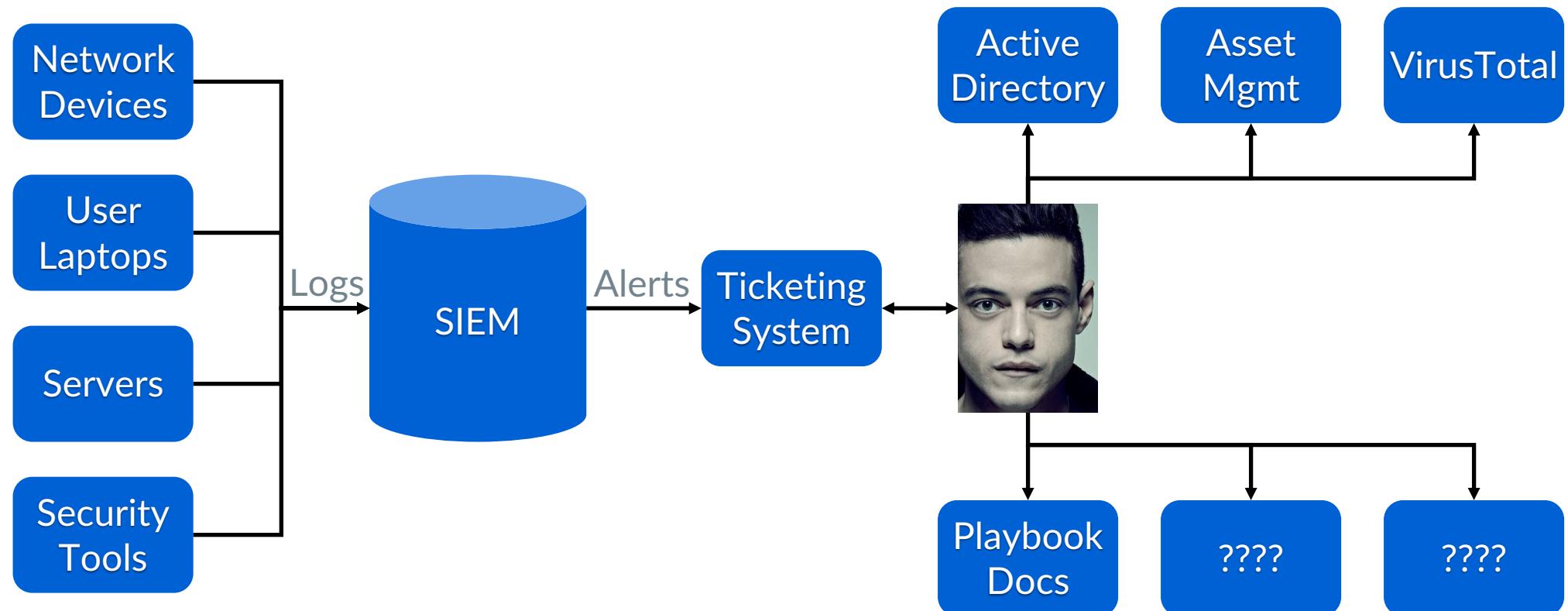
Tristan Waldear, Security Automation Engineer
twaldear@box.com

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

The screenshot shows the Splunk Enterprise interface for security alerting. The top navigation bar includes 'splunk>enterprise', 'App: Securi...', 'Administrator', 'Messages', 'Settings', 'Activity', 'Help', 'Find', and a search icon. The main title is 'Mockscan Alert'. The search bar contains the query 'index="main" source="/var/log/mockscan.log" "alert"'. The results show one event from 2/26/19 between 12:41:00.000 AM and 1:41:52.000 AM, with 'No Event Sampling' selected. The event details are listed below:

i	Time	Event
>	2/26/19 1:40:07.000 AM	{ [-] action: malicioususer hostname: computer md5: d41d8cd98f00b204e9800998ecf8427e message_type: alert severity: critical user: mosesschwartz } Show as raw text host = 316faeebd420 source = /var/log/mockscan.log sourcetype = linuxhost

On the left, the 'Selected Fields' section lists 'host 1', 'source 1', and 'sourcetype 1'. The 'Interesting Fields' section lists 'action 1', 'hostname 1', 'index 1', 'linecount 1', 'md5 1', 'message_type 1', 'punct 1', 'severity 1', 'splunk_server 1', and 'timestamp 1'. Navigation controls include 'Events (1)', 'Patterns', 'Statistics', 'Visualization', 'Format Timeline', 'Zoom Out', 'Zoom to Selection', 'Deselect', and a time range of '1 minute per column'.

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

The screenshot shows the Splunk Enterprise web interface. At the top, there's a navigation bar with tabs for 'Searches, reports, and alerts' and a warning icon for 'Not Secure'. Below the navigation is a header with 'splunk>enterprise' and various dropdown menus for 'Apps', 'Messages', 'Settings', 'Activity', and 'Help'. A search bar is on the right.

The main content area is titled 'Create Alert'. It has a sidebar on the left listing '1 Search' and 'filter' under 'Searches', and 'Name', 'Test Alert', and 'Foo' under 'Saved Searches'. The main form is titled 'Settings' and contains the following fields:

- Title:** Mockscan Alert
- Description:** An example alert from our fake "Mockscan" logs
- Search:** index="main" source="/var/log/mockscan.log" "alert"
- App:** Security Alerting (security_alerting)
- Permissions:** Private (selected)
- Alert type:** Scheduled (selected)
- Run on Cron Schedule:** Run on Cron Schedule
- Time Range:** Last 15 minutes
- Cron Expression:** 0/15 * * * *
- e.g. 00 18 *** (every day at 6PM). [Learn More](#)

At the bottom of the dialog are 'Trigger Conditions' and two buttons: 'Cancel' and 'Save'.

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)

The screenshot shows the Splunk Enterprise interface with a modal dialog titled "Create Alert". The dialog is used to configure an alert trigger. The configuration includes:

- Trigger Conditions:** Set to trigger when "Number of Results" is greater than 0.
- Trigger:** Set to "Once" (radio button selected).
- Throttle:** An unchecked checkbox.
- Trigger Actions:** A "Webhook" action is added, with the URL set to "http://68.183.166.37/splunk_webhook".

At the bottom right of the dialog are "Cancel" and "Save" buttons.

automation_server.py

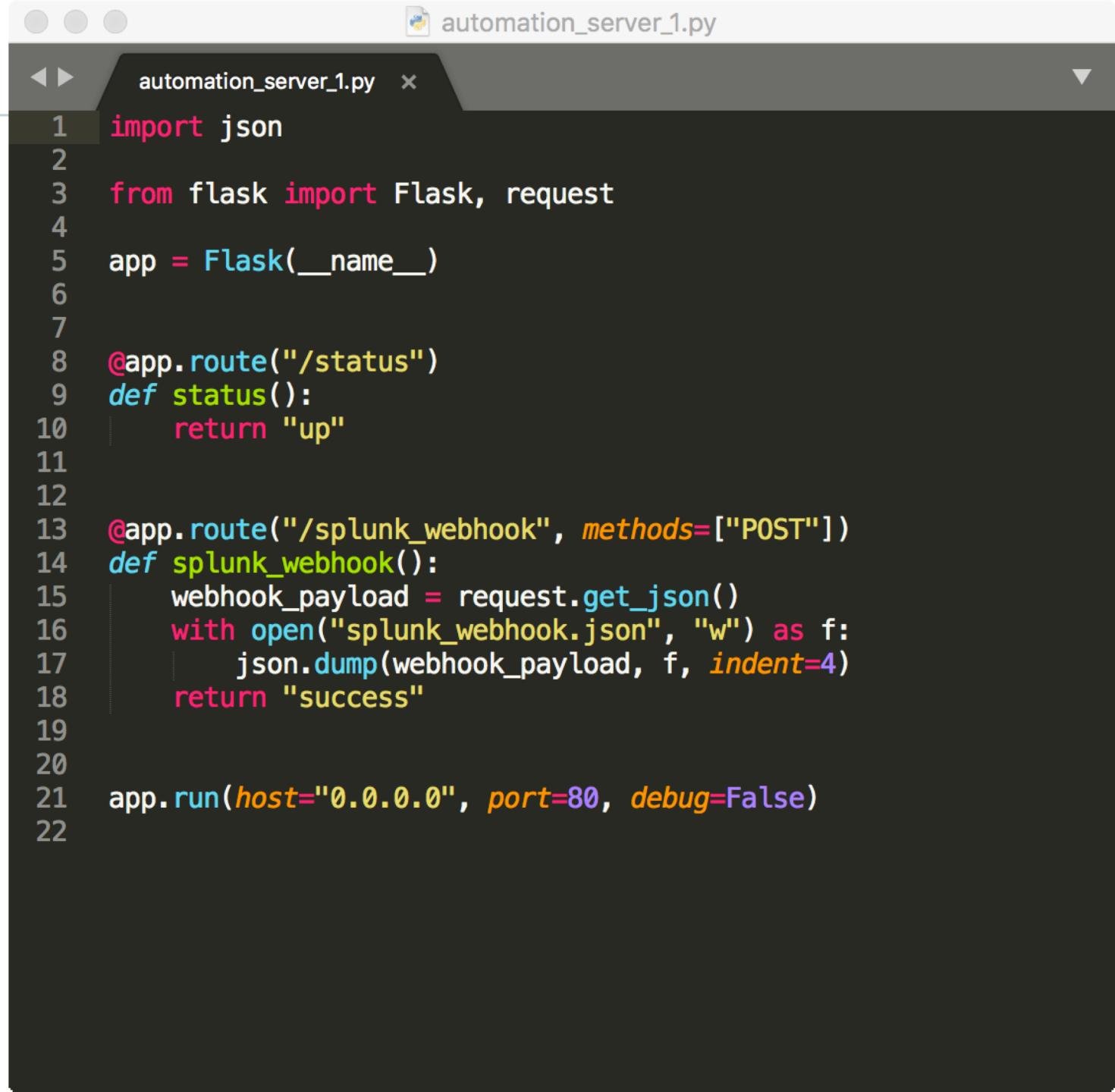
Receive Splunk webhook payload

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!



The image shows a screenshot of a code editor window titled "automation_server_1.py". The code is written in Python and uses the Flask framework to handle requests. It includes a status endpoint and a POST endpoint for receiving Splunk webhook payloads, which are then saved to a file.

```
automation_server_1.py
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

```
security_automation_simplified — root@automation-server: ~ — ssh -i ~/ssh/moses_test root@68.183.166.37 — 68x30
automation-server ~#
automation-server ~# python3 automation_server_1.py
* Serving Flask app "automation_server_1" (lazy loading)
* Environment: production
WARNING: Do not use the development server in a production environment.
Use a production WSGI server instead.
* Debug mode: off
* Running on http://0.0.0.0:80/ (Press CTRL+C to quit)

8.39.49.160 - - [01/Mar/2019 21:54:00] "POST /splunk_webhook HTTP/1.1" 200 -

[^Cautomation-server ~#
automation-server ~# head splunk_webhook.json
{
    "search_name": "Mockscan Alert",
    "owner": "admin",
    "app": "security_alerting",
    "result": {
        "eventtype": "",
        "hostname": "computer",
        "severity": "critical",
        "_si": [
            "774b4bc0418e",
automation-server ~# ]
```

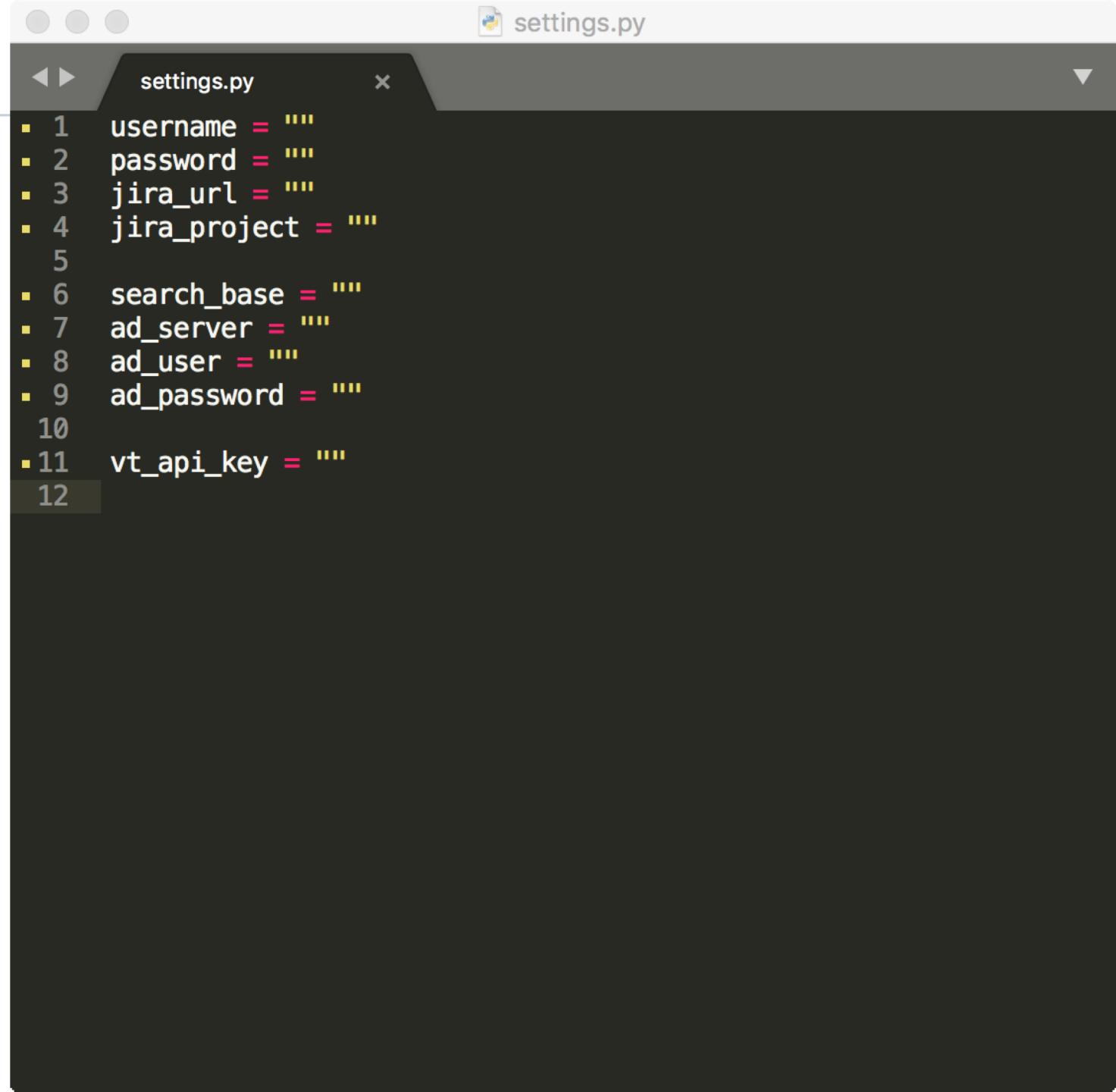
settings.py

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



The image shows a screenshot of a code editor window titled "settings.py". The file contains the following code:

```
 1 username = ""
 2 password = ""
 3 jira_url = ""
 4 jira_project = ""
 5
 6 search_base = ""
 7 ad_server = ""
 8 ad_user = ""
 9 ad_password = ""
10
11 vt_api_key = ""
12
```

The code editor has a dark theme with syntax highlighting. Line numbers are visible on the left. The file is currently open in the editor.

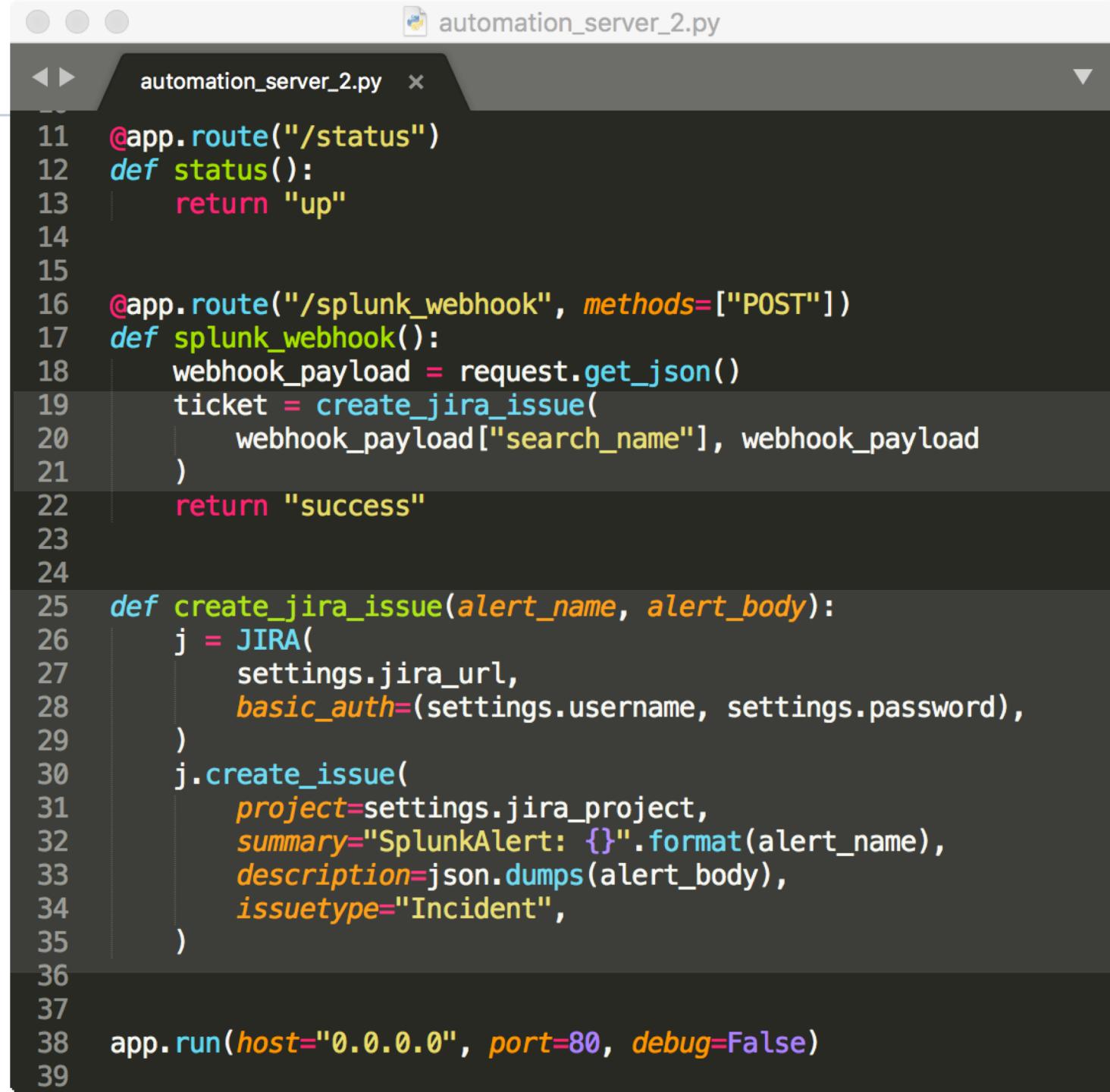
automation_server.py

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
11 @app.route("/status")
12 def status():
13     return "up"
14
15
16 @app.route("/splunk_webhook", methods=["POST"])
17 def splunk_webhook():
18     webhook_payload = request.get_json()
19     ticket = create_jira_issue(
20         webhook_payload["search_name"], webhook_payload
21     )
22     return "success"
23
24
25 def create_jira_issue(alert_name, alert_body):
26     j = JIRA(
27         settings.jira_url,
28         basic_auth=(settings.username, settings.password),
29     )
30     j.create_issue(
31         project=settings.jira_project,
32         summary="SplunkAlert: {}".format(alert_name),
33         description=json.dumps(alert_body),
34         issuetype="Incident",
35     )
36
37
38 app.run(host="0.0.0.0", port=80, debug=False)
39
```

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

The screenshot shows the Jira Webhooks configuration page. On the left, a sidebar menu is open under the 'Jira' heading, with 'WebHooks' selected. The main content area is titled 'System WebHooks' and shows a single webhook configuration named 'Mockscan Alert'. The configuration includes:

- Name***: Mockscan Alert
- Status***: Enabled
- URL***: http://68.183.166.37/jira_mockscan_created_webhook
- Description**: (empty)
- Events**: **Issue related events**
You can specify a JQL query to send only events triggered by matching issues. The JQL filter only applies to events under the Issue and Comment columns.
Example: `project=sir AND summary ~ "SplunkAlert: Mockscan Alert"`

automation_server.py

Add a comment to Jira

Activity

All **Comments** Work log History

Activity

▼  Security Bot added a comment - 1 minute ago REPORTER

MockScan Alert: User=mosesschwartz
MD5=d41d8cd98f00b204e9800998ecf8427e

 Click to add comment

automation_server_3.py

```
12 def status():
13     return "up"
14
15
16 @app.route("/jira_mockscan_created_webhook", methods=["POST"])
17 def mockscan_created():
18     jira_webhook = request.get_json()
19     jira_desc = jira_webhook["issue"]["fields"]["description"]
20     issue_key = jira_webhook["issue"]["key"]
21     splunk_alert = json.loads(jira_desc)
22     user = splunk_alert["result"]["user"]
23     md5 = splunk_alert["result"]["md5"]
24     comment = "MockScan Alert: User={} MD5={}".format(user,md5)
25     jira_comment(issue_key, comment)
26     return "success"
27
28
29 def jira_comment(issue_key, comment):
30     j = JIRA(
31         settings.jira_url,
32         basic_auth=(settings.username, settings.password),
33     )
34     issue = j.issue(issue_key)
35     j.add_comment(issue, comment)
36
37
38 @app.route("/splunk_webhook", methods=["POST"])
39 def splunk_webhook():
40     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  
}}]}
```

The screenshot shows a code editor window with the file 'ad_lookup.py' open. The code is written in Python and uses the ldap3 library to search Active Directory for a user by their username. It imports json, ldap3, and settings. A function search_ad takes a username as input and returns a JSON object. It creates a Server object with the ad_server and get_info=ALL parameters, and a Connection object with various authentication and search parameters. Finally, it binds to the server and performs a search with the specified filter, base, scope, and attributes, returning the response as a JSON object.

```
ad_lookup.py  
1 import json  
2  
3 from ldap3 import (  
4     Server, Connection, ALL, NTLM, SUBTREE, ALL_ATTRIBUTES  
5 )  
6 import settings  
7  
8  
9 def search_ad(username):  
10     server = Server(settings.ad_server, get_info=ALL)  
11     search_filter = "(&(sAMAccountName={}))".format(username)  
12     conn = Connection(  
13         server,  
14         user=settings.ad_user,  
15         password=settings.ad_password,  
16         authentication=NTLM,  
17         auto_referrals=False,  
18     )  
19     conn.bind()  
20     conn.search(  
21         search_base=settings.search_base,  
22         search_filter=search_filter,  
23         search_scope=SUBTREE,  
24         attributes=ALL_ATTRIBUTES,  
25         get_operational_attributes=True,  
26     )  
27     return json.loads(conn.response_to_json())  
28  
29  
30 
```

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

```

15
16
17 @app.route("/jira_mockscan_created_webhook", methods=["POST"])
18 def mockscan_created():
19     jira_webhook = request.get_json()
20     jira_desc = jira_webhook["issue"]["fields"]["description"]
21     issue_key = jira_webhook["issue"]["key"]
22     splunk_alert = json.loads(jira_desc)
23     user = splunk_alert["result"]["user"]
24     md5 = splunk_alert["result"]["md5"]
25     ad_lookup_enrichment(issue_key, user)
26     return "success"
27
28
29 def ad_lookup_enrichment(issue_key, user):
30     ad_info = ad_lookup.search_ad(user)
31     user_attributes = ad_info["entries"][0]["attributes"]
32     comment = "Name: {}\\n Company: {}\\nDept: {}\\nLocation: {}"
33     comment = comment.format(
34         user_attributes["cn"],
35         user_attributes["company"],
36         user_attributes["department"],
37         user_attributes["l"],
38     )
39     jira_comment(issue_key, comment)
40
41
42 def jira_comment(issue_key, comment):
43     j = JIRA(
44         options={"server": "https://jira.box.com"})

```

virustotal.py

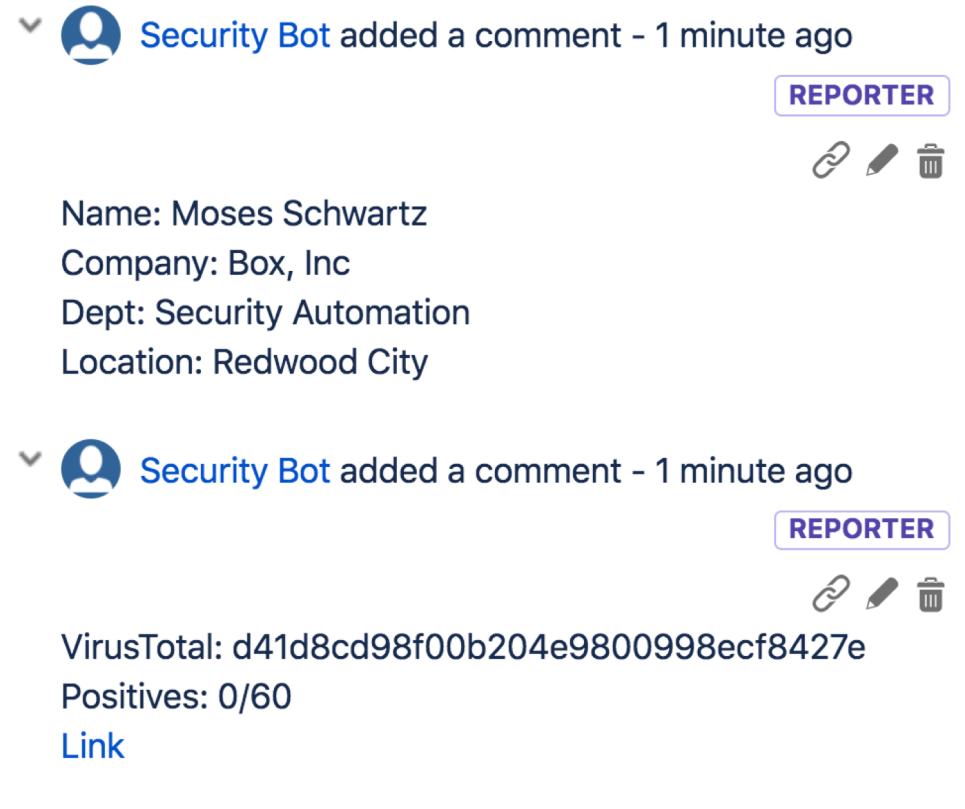
Get a file scan report

```
{'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' }  
....
```

```
1 import requests  
2  
3 import settings  
4  
5  
6 def get_file_scan_report(file_hash):  
7     vt_url = "https://www.virustotal.com/vtapi/v2/file/report"  
8     params = {  
9         "apikey": settings.vt_api_key,  
10        "resource": file_hash,  
11    }  
12    response = requests.get(vt_url, params=params)  
13    return response.json()  
14
```

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

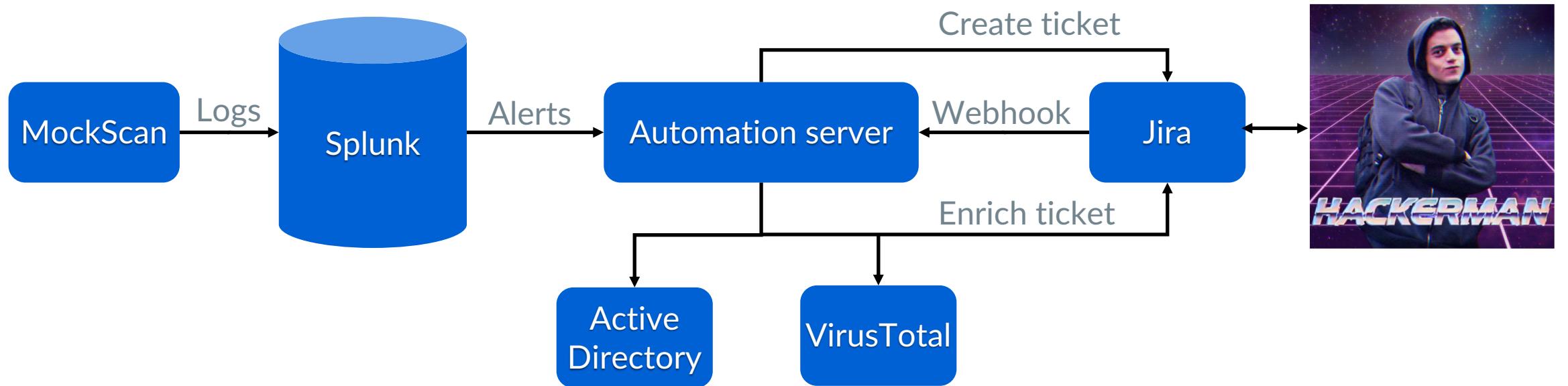
REPORTER

VirusTotal: d41d8cd98f00b204e9800998ecf8427e
Positives: 0/60
Link

```
automation_server_5.py
```

```
16
17
18 @app.route("/jira_mockscan_created_webhook", methods=["POST"])
19 def mockscan_created():
20     jira_webhook = request.get_json()
21     jira_desc = jira_webhook["issue"]["fields"]["description"]
22     issue_key = jira_webhook["issue"]["key"]
23     splunk_alert = json.loads(jira_desc)
24     user = splunk_alert["result"]["user"]
25     md5 = splunk_alert["result"]["md5"]
26     ad_lookup_enrichment(issue_key, user)
27     vt_filescan_enrichment(issue_key, md5)
28     return "success"
29
30
31 def vt_filescan_enrichment(issue_key, file_hash):
32     vt_result = virustotal.get_file_scan_report(file_hash)
33     comment = "VirusTotal: {}\nPositives: {} / {} \n[Link]({})".format(
34         vt_result["md5"],
35         vt_result["positives"],
36         vt_result["total"],
37         vt_result["permalink"],
38     )
39     jira_comment(issue_key, comment)
40
41
42
43 def ad_lookup_enrichment(issue_key, user):
44     ad_info = ad_lookup.search_ad(user)
45     user_attributes = ad_info["entries"][0]["attributes"]
```

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

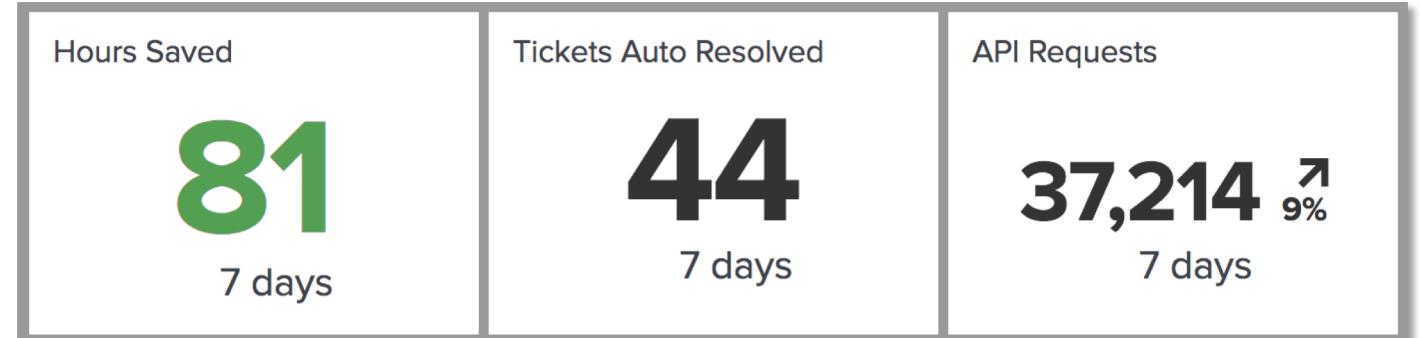
Quantify your impact

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