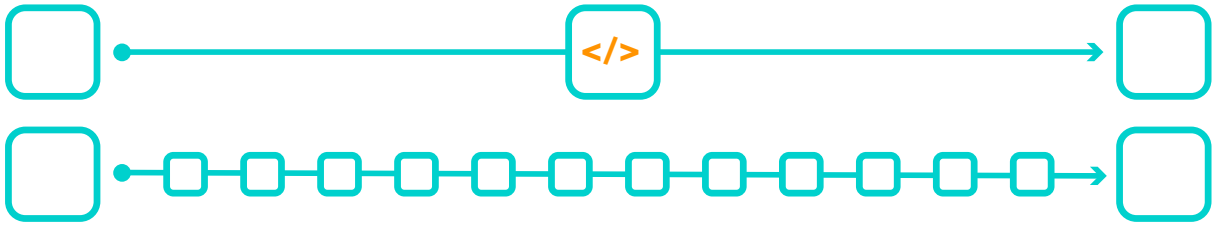
Whitepaper ■ June 2019

VULCAN

Automating Your Vulnerability Remediation Process

examples and usecases



Contents

Why Automate?

Managing your vulnerabilities - from identifying and prioritizing all the way through to remediation and resolution is an extremely time-consuming task. Vulnerability managers need to find ways to take the most mundane aspects of vulnerability management such as scanning, opening tickets, and integrating them into an efficient process.
That's where automation comes in.

> Automating vulnerability response processes isn't just a luxury in your cyber hygiene arsenal. It has become an absolute necessity. There are simply too many vulnerabilities (over 16 thousand disclosures last year) to keep up with.

Imagine if you were to take just five minutes to open a ticket for every single vulnerability you encounter, your team could end up wasting tens of thousands of staff-hours on this task alone! Not to mention the considerable amount of time the actual remediation process for thousands of vulnerabilities would take. Let's face it - unless you are willing to let your team drown in vulnerability alerts, there is simply no other choice than to simplify, streamline and considerably speed up vulnerability management by automating manual processes.

So how do we go about automating these vulnerabilities?

See below the code and scripts we have developed for precisely this purpose (also freely available on GitHub). Adapting this code for your organization's needs will get you focused on remediating the vulnerabilities- and not just opening tickets!

Important caveats before we dive into the automation scripts below:

- It's a very bad idea to put authentication credentials in code. Although the examples below show them for ease of use – it's definitely not the correct way to do it!
- The code is simplified for the bare minimum without any error handling, logging, edge case handling – please bear in mind when using.
- You'll probably need some imports...

```
import json
import urllib.request
import zipile
import xml.etree.ElementTree as ET
from jira import JIRA
```

Automate Prioritization: Fuse TI with Vulnerability Data

Before we can even think about remediating a security issue, we need to identify the most urgent issues. Prioritizing the never-ending list of vulnerabilities is a challenge - and just relying on a CVE score is not enough. Something that might tip the scale towards remediating is whether there is an active exploit in the wild for the specific vulnerability. A published exploit makes hacking the vulnerable asset child's play and should affect prioritization.

What we need is a way to get free exploit information from public databases. By comparing that information with the internal scanner result, we can create a list of the specific vulnerabilities that are in both lists – meaning both vulnerable, and easily exploitable.

Let's look at a short Python script to help with the task.

- First, we'll download the free NIST CVE database, to use as our exploit database.
- After downloading the JSON file, we can unzip it, parse it, and create a simple list of all the CVES that have the "Exploit" tag.

```
# Download NIST JSON
nist_json = urllib.request.urlretrieve("https://nvd.nist.gov/feeds/
json/cve/1.0/nvdcve-1.0-modiied.json.zip", "nvdcve-1.0-modiied.json.
zip")

# Unzip NIST JSON
with zipile.ZipFile("nvdcve-1.0-modiied.json.zip", 'r') as zip_ref:
 zip_ref.extractall( ".")

# Parse NIST JSON
with open("nvdcve-1.0-modiied.json", encoding="utf8") as data:
  json_data = json.load(data):
  cve_list = data['CVE_Items']


# Create a list of exploitable CVES:
exploitable_cves = []
for cve in cve_list:
  for reference_data in cve['cve']['references']['reference_data']:
  for tag in reference_data['tags']:
        if tag == "Exploit":
          exploitable_cves.append(cve['cve']['CVE_data_meta']['ID'])
```

After we have our list of exploitable CVEs, let's create a similar list from our vulnerability scanner. For this example we'll use a Nessus XML report. The XML is built in the following structure:
Asset -> Report Items -> CVEs
So we simply go over the XML, look for all of the CVEs, and create a list:

```
# Parse the Nessus report for the list of vulnerabilities
vulnerable_cves = []
tree = ET.parse('nessus_report.nessus')
root = tree.getroot()
for report in root:
  for reportHost in report:
    for detail in reportHost:
        if detail.tag == "ReportItem":
            for inner_detail in detail:
                if (inner_detail.tag == "cve"):
                      vulnerable_cves.append(inner_detail.text)
```

Finally, all we have to do is fuse both lists to see which of our vulnerable CVEs are exploitable:

```
# Fusing both lists Together
fused_list = [vulnerable_cve for vulnerable_cve in vulnerable_cves if
vulnerable_cve in exploitable_cves]
```

Automate Response: Streamline Ticket Creation

So now we have a list of CVEs that we know are extra risky. What should we do next? Usually, we'd open a ticket for the relevant person to handle them. But how much time could we save if we automated this process? Python has a wonderful library for communicating with JIRA, our ticketing platform. For example, if we want to open a ticket for each of these CVEs we can attach a list, have the ticket can include additional information (such as the affected hosts), or even assign the ticket automatically to the person in charge. The sky's the limit.

We'll first connect to the Jira server, then go over our new list, and open the tickets accordingly:

```
jira = JIRA("https://domain.atlassian.net", auth=('user@domain.com',
'secretpassword'))

for cve in fused_list:
  issue_dict = {
     'project': {'key': 'TEST Project'},
     'summary': 'Exploitable CVE found!',
     'description': f'Exploitable CVE: {cve}',
     'issuetype': {'name': 'Task'},
 }
 jira.create_issue( fields=issue_dict)
```

Pretty easy right?

Well – JIRA can be replaced with e-mail, slack, ServiceNow, whatever you usually use. As long as the notification can be automated, you can save a lot of time.

Coverage check

As another security measure, we want to make sure that no asset is left unscanned. In our case, we have an AWS environment which is covered by a tenable.io scanner. But there might be a problem when EC2 instances are created - the scanner may not be configured correctly to scan them, or no agent is installed. Finding a way to identify these pesky instances and receive an alert about them would help put our minds at ease, knowing that our system is covered. A simple way to achieve this is to compare a list of our EC2 instances against a list of our tenable configured assets, and then find mismatches. Having a scheduled task for this process could help.

To do this we first connect to AWS and create a list of all IP addresses we find there.

```
AWS_ACCESS_KEY_ID = 'XXX'
AWS_SECRET_ACCESS_KEY = 'XXX'

ec2conn = ec2.connection.EC2Connection(AWS_ACCESS_KEY_ID, AWS_SECRET_
ACCESS_KEY)
reservations = ec2conn.get_all_instances()
aws_ips = []
instances = [i for r in reservations for i in r.instances]
for i in instances:
 aws_ips.append(i.ip_address)
```

Second step would be to do the same for the tenable account:

```
tc = TenableIOClient(access_key='XXX', secret_key='XXX')
asset_list = tc.assets_api.list().assets
tenable_ips = []
for asset in asset_list:
  for ip in asset.ipv4:
    tenable_ips.append(ip)
```

And the last step would be to see which one of our AWS IP addresses is not present in our tenable list and create a new list.

```
unscanned_ips = []
for ip in aws_ips:
  if ip not in tenable_ips:
    unscanned_ips.append(ip)
```

The list can be later sent to a JIRA ticket or to any other form of communication.

What's Next?

Vulnerability management is complicated.
It is an ongoing process that will probably not be solved - but needs to be managed properly. For that to happen, you have to make sure that you have eyes on the entire process - from the moment the vulnerability was discovered, until it was verified as solved. There are many tricks you can use to help you achieve the process management, and will save you the trouble of running manual, mundane tasks.

Each one of the examples above can evolve and grow to further help with the load. A quick example - after opening a Jira ticket, you might want to use a webhook to be informed when that ticket is updated or solved. Or how about automatically setting the vulnerabilities as solved when the ticket is closed? Wouldn't that be nice?

How about remediation?

What can save you the trouble of doing the research for solutions for the vulnerabilities in your environment? There are many public resources that can assist with the CVE <> Patch correlation. Can they be automated as well?

There really are too many vulnerabilities to be dealt with manually. If you're interested in learning more about automating your vulnerability management process, why not book a consultation with one of our vulnerability experts?