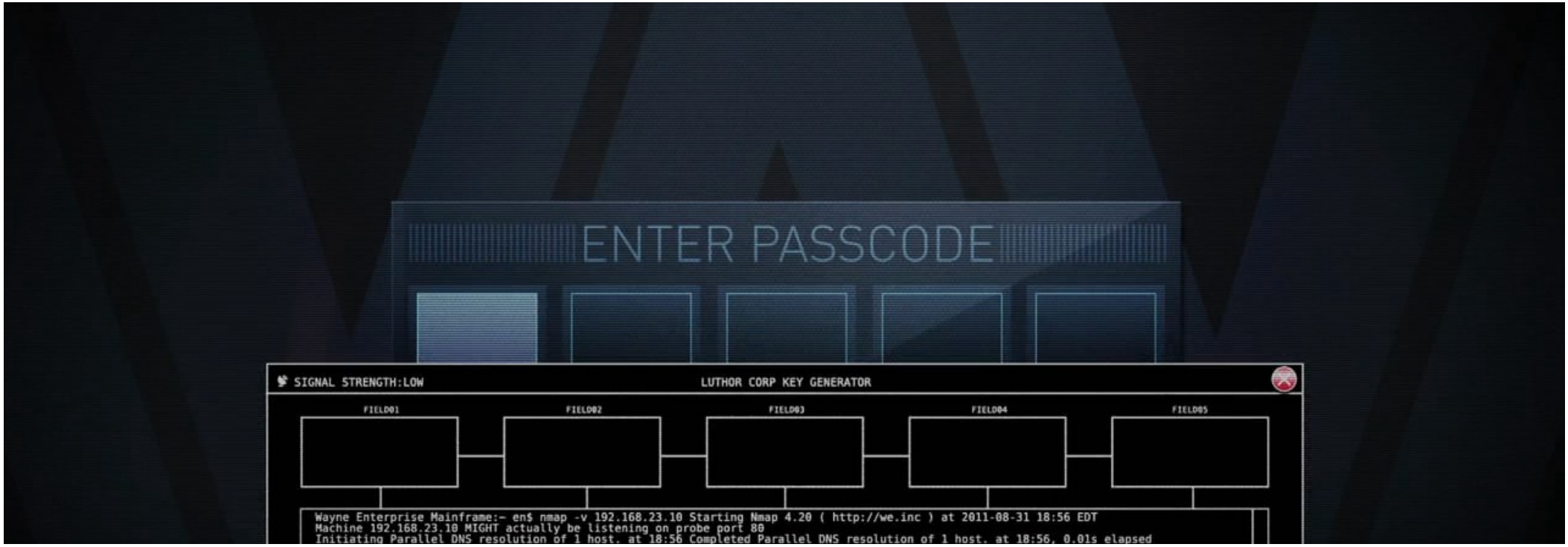
# Writing NMAP Scripts Like A Super-Hero

Peter Benjamin Follow

May 21, 2016 · 6 min read




Justice League: Doom (2012)

What do *Matrix Reloaded (2003)*, *The Bourne Ultimatum (2007)*, *Die Hard 4 (2007)*, *The Girl with the Dragon Tattoo (2009)*, *Justice League: Doom (2012)*, *G.I. Joe: Retaliation (2013)*, and *Fantastic Four (2015)* have in common?

## NMAP

### What is NMAP?

NMAP ("Network Mapper") is arguably the most widely used, free, open-source utility for network discovery and security auditing. In laymen's

terms, NMAP is used to scan a network and identify exposed ports (e.g. 80, 443 …etc).

Any script-kiddie can point a port-scanner against a network or a C.I.D.R. block (e.g. 192.168.0.0/24) and spray, but an advanced user knows how to extend this tool to be more than just a port scanner.

## NMAP Primer

The most basic scan looks like this:

```
$ nmap 192.168.0.0/24

Starting Nmap 7.12 ( https://nmap.org ) at 2016-04-27 18:56 PDT

Nmap scan report for 192.168.1.79
Host is up (0.00044s latency).
Not shown: 748 closed ports, 251 filtered ports
PORT     STATE SERVICE
3000/tcp open  ppp


Nmap scan report for 192.168.1.80
Host is up (0.011s latency).
Not shown: 997 closed ports
PORT     STATE SERVICE
```

```
80/tcp   open  http
443/tcp  open  https

Nmap done: 256 IP addresses (2 hosts up) scanned in 8.20 seconds
```

This will just return all open ports in the C.I.D.R. block. Not that interesting.

But, wait a second, what is port 3000 running “ppp” service on 192.168.1.79?

Well, I started a Rails development server on port 3000 on my local machine to demonstrate the power of NMAP scripts.

## Setting up our scenario...

So, we’ve identified a web server running on port 3000. So what?

It’s not unusual for System Administrators (SysAdmins) to run a service on a different port (e.g. HTTP on 8080, HTTPS on 8443, SSH on 2222).

If we try to curl that host on port 3000:

```
$ curl localhost:3000
<!DOCTYPE html>
<html>
 <head>
 <title>Ruby on Rails: Welcome aboard</title>
 <style media="screen">
 body {
 margin: 0;
 margin-bottom: 25px;
 padding: 0;
 background-color: #f0f0f0;
 font-family: "Lucida Grande", "Bitstream Vera Sans", "Verdana";
 font-size: 13px;
 color: #333;
 }
...
</html>
```

As you can see, we get an HTML response back.

## Why is this important?

Well, this particular scenario is benign. However, web servers can possibly leak sensitive information.

Let's poke a little port at our web application running on port 3000:

```
$ curl localhost:3000/users
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="utf-8" />
  <title>Action Controller: Exception caught</title>
  <style>
    body {
      background-color: #FAFAFA;
      color: #333;
      margin: 0px;
    }
...
</head>
```

I tried to see if there was a "/users" path available, but the application complained. Maybe it doesn't exist or maybe authentication is required to access that resource. Oh, well, the party doesn't end here:

```
$ curl localhost:3000/admin
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="utf-8" />
  <title>Action Controller: Exception caught</title>
  <style>
    body {
      background-color: #FAFAFA;
      color: #333;
```

```
      margin: 0px;
    }
...
</head>
```

Still, no luck! How about:

```
$ curl localhost:3000/admins
<!DOCTYPE html>
<html>
<head>
  <title>App</title>
...


<h1>Listing Admins</h1>


<table>
 <thead>
 <tr>
 <th>Username</th>
 <th>Password</th>
 <th colspan="3"></th>
 </tr>
 </thead>


<tbody>
 <tr>
 <td>PeterBenjamin</td>
 <td>MySuperSecr3t</td>
 <td><a href="/admins/1">Show</a></td>
```

```
 <td><a href="/admins/1/edit">Edit</a></td>
 <td><a data-confirm="Are you sure?" rel="nofollow" data-
method="delete" href="/admins/1">Destroy</a></td>
 </tr>
 </tbody>
</table>


<br>


<a href="/admins/new">New Admin</a>


</body>
</head>
```

Bingo!

Can you imagine having to do this manually across tens, or hundreds, or even thousands of web-servers looking for this type of information leak across your organization? If only NMAP can help us here!

Of course, NMAP can help us automate this process. This is what differentiates a script-kiddie from someone who knows what they're doing.

Let's examine how we can write this NMAP script…

# NMAP Scripts

We are barely scratching the surface of what NMAP Scripting Engine (NSE) can do, so I want to point out that NMAP has an online book dedicated to NSE. Want to make your scripts run in parallel? What about collaborative multithreading? Check out their docs.

. . .

First off, NMAP scripts are written in a language called *Lua*, a small, lightweight, and embeddable language that is surprisingly performant and can talk directly to C code. If you have any programming experience, Lua's syntax is simple and intuitive.

With Lua's introduction out of the way, let's dive right in.

. . .

We will start by breaking down our script into 3 components: the *head*, the *rule*, and the *action*.

## The Head

The *Head* essentially contains meta-informaiton, such as *description*, *author*, *categories*, *dependencies*, *license*, and any other comment pertinent to the script (e.g. importing libraries, example usage, sample output …etc).

## DESCRIPTION:

```
-- This is a comment. We'll use this to denote the section of the
script.
-- HEAD


description = [[


This is a multi-line literal string. This is where we offer a simple
explanation of what our script aims to do. For instance:


Attempts to enumerate "/admins" resource on web apps running on port
3000 and retrieves Admin usernames and passwords.


]]
```

## USAGE:

```
---
-- @usage
-- nmap -p 3000 --script rails-admins <host>
--
-- @output
-- PORT     STATE SERVICE
-- 3000/tcp open  ppp
```

```
-- | rails-admins:
-- | <td>PeterBenjamin</td>
-- | <td>MySuperSecr3t</td>
--
```

## AUTHOR:

```
author = "Peter Benjamin"
```

## IMPORTS:

```
-- we will be using these imported libraries in the Rule section.
local nmap = require "nmap"

-- we will use these in the action
local http = require "http"
local stdnse = require "stdnse"
```

. . .

## The Rule

The *Rule* section decides whether to execute this script or not based on some rule(s).

In our scenario, the rule is that port 3000 must be explicitly passed to the *nmap* port argument (-p):

```
-- RULE
portrule = function(host, port)
  local auth_port = { number=3000, protocol="tcp" }
  local identified = nmap.get_port_state(host, auth_port)
  -- "nmap" imported library gives us access to "get_port_state()"
function


  return identified ~= nil -- The operator "~=" is "not equal"
    and identified.state == "open"
    and port.protocol == "tcp"
    and port.state == "open"
end
```

. . .

## The Action

The *Action* is where we implement the actual functionality.

```
--ACTION SECTION
local DEFAULT_URI = "/admins"


-- helper function to check if response contains "password"
local function check_rails_admin(host, port, path)
 local resp = http.get(host, port, path)
 if not http.response_contains(resp, "password") then
   return false
 end
 return resp
end


-- main logic
action = function(host, port)
 local vuln_rails = check_rails_admin(host, port, DEFAULT_URI)
 local output = {}
 if not vuln_rails then
   stdnse.print_debug(1,"%s: This does not look like a vulnerable
Rails app", SCRIPT_NAME)
   return
 else
   output = string.match(vuln_rails["body"], "%<td%>.*%<%/td%>")
 end


 return output
end
```

Essentially, we check the target by sending a GET request to /admins. We're interested in targets that return a response containing the word "password". If such response is returned, we pattern-match using regular expressions (Regex).

The output:

```
Nmap scan report for localhost (127.0.0.1)
Host is up (0.00028s latency).
PORT STATE SERVICE
3000/tcp open ppp
| rails-admins:
| <td>PeterBenjamin</td>
| <td>MySuperSecr3t</td>

Nmap done: 1 IP address (1 host up) scanned in 0.19 seconds
```

Tada! We've successfully written a custom NMap script to look for sensitive information being disclosed/leaked by web applications.

You can find this script on my GitHub for reference.

. . .

## Next Steps...

Well, this was just a very crude NMap script and it barely scratched the surface of what we can do with NMap and Lua scripts.

As with anything in life, practice makes perfect. Lua is a quirky little language that took me a week to become basically familiar with Lua and its libraries/API.

Need some practice with NMap? Join InfoSecLabs.
Need some practice with Lua? Try out repl.it.

Happy Hacking!

Devsecops SEC Dev

12 claps

WRITTEN BY

## Peter Benjamin

Now migrated to https://pbnj.dev & https://dev.to/petermbenjamin

Follow

Write the first response

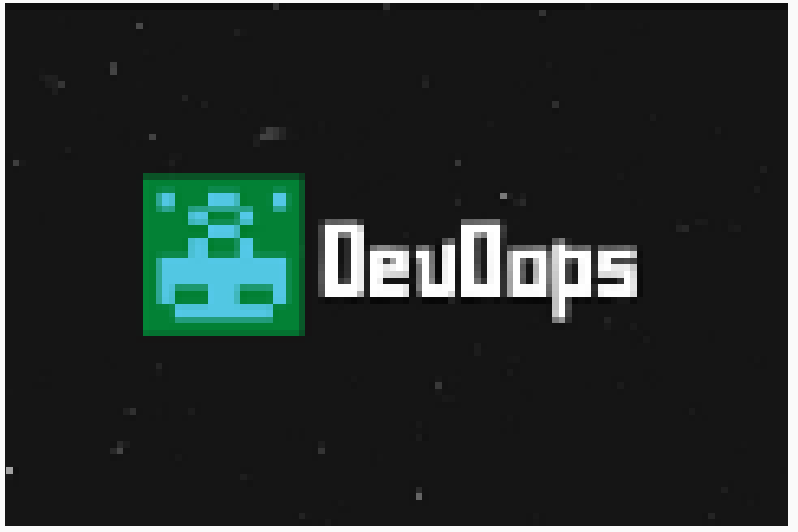
## More From Medium

Related reads

### DevOops Write-up (HTB)

George O in CTF Writeups
Oct 13, 2018 · 4 min read

207

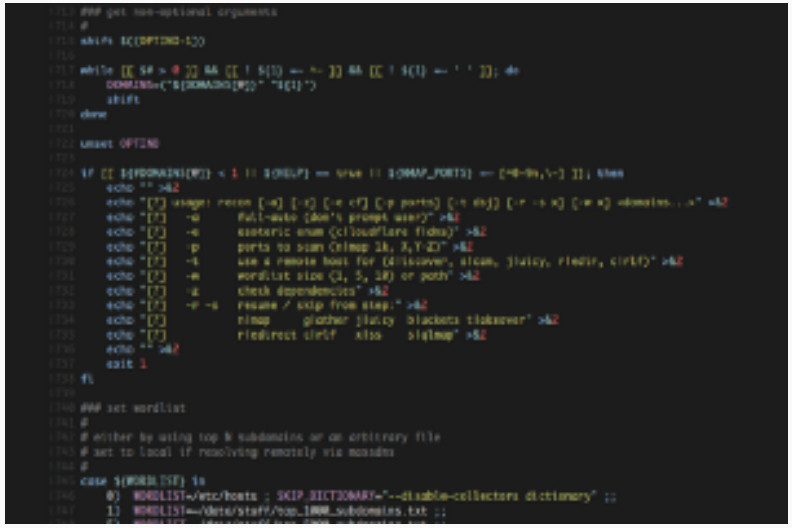


Related reads

## Reconnaissance: a eulogy in three acts

europa
Feb 11, 2018 · 8 min read

1.5K



Related reads

## Secnotes Write-up (HTB)

George O in CTF Writeups
Jan 20 · 6 min read

261