CS4400 Module Final Report

Name: Stefano Lupo - Student No: 14334933 - Date: 13/12/18

Inferno Ball Team Self Evaluation

Team number 32

- Stefano Lupo (me)
- Aron Hoffmann
- Yash Pandey
- Nicholas Blott

	Stefano Lupo	**Aron Hoffmann**	**Yash Pandey**	**Nicholas Blott**
Effort	High	High	Low	Low
Effectiveness	High	High	Low	Low

What I Learned in the Module

Effective Use of Bash and Scripting

As I run Linux on all of my machines, I use Bash pretty much every day and as such I had some basic experience working with the terminal and was pretty comfortable with it. However, I was by no means competent. As such, I opted to stay for the Linux basics lecture. I knew some of what was discussed but picked up the basics of using things like `awk`, `grep` and `find` which I knew existed but hadn't yet bothered to figure out!

I forced myself to stay away from manually writing Python scripts to do everything that was needed for the module. As such, I learnt a lot about making effective use of the functionality that ships with Linux including a bunch of the basic commands, I/O redirection, piping and combining all of these into simple scripts. `awk` and `grep` proved to be particularly useful for looking through and partitioning large wordlists to be used for cracking. I also made extensive use of `scp` for moving files around across machines. I ended up writing a couple of reusable bash scripts for the module which can be found here. They were used for formatting the potfile the way submitty liked it, partitioning the given list of hashes by hash type using regex, printing out the progress I made for each hash type (e.g 299/354 DES, 305/350 MD5 etc) and printing out the passwords sorted by length and hash type to gain some insights into previously cracked passwords.

Password Cracking & Security

I knew about some of the basics of password storage from building (very questionable) login systems using PHP a few years ago. However, I had always wondered how password cracking worked. Starting off with John The Ripper (JTR) was pretty trivial as you could just throw a list of passwords at it and it knew what to do. Learning to use Hashcat was a little more tricky as you had to specify the `-m` flag, indicating which hash algorithm to use (MD5 / SHA256 etc). One of the things that took me far longer to realise than it should have was that the salts of the passwords were stored with the password hashes themselves. However once I understood that, everything started to click. I spent a lot of time reading the Hashcat forums and documentation to try and understand what each of the modes did. For the most part, I just threw monumental wordlists along with as much compute power I could get my hands on at the problem. However after only getting a couple of hundred hashes for practical 4 (prior to the hints being given out), I started looking at other more complex cracking modes such as masks, combinators and wordlist/mask hybrids. I also spent some time researching the new prince mode. My uses of these modes are described in

the practical 4 section. I didn't make much progress using these tools as the hints were given out pretty soon after I started using them which made wordlist attacks much faster again. However, it was interesting playing around with the prince mode in particular.

Using GPUs and GPU Programming

I had used GPUs in the past for training machine learning models and neural nets, but most of the GPU complexity was entirely abstracted away. The module gave me the opportunity to actually get my hands dirty with running compute tasks on GPUs. The lecture series on OpenCL was really interesting and some further reading gave me a good overview of the OpenCL ecosystem. I haven't taken any graphics programming modules so this module was my first real exposure to GPU programming. Setting up John to use GPUs was a little tricky at the start but once I got it running on the GPU in my home machine, I could repeat the process on Rosetta Hub instances pretty easily. I also learnt a small amount about the important metrics for GPUs (e.g. number of CUDA cores etc).

Using Google Cloud

Although not an intended learning outcome of the module, I ended up learning a lot about the Google Cloud infrastructure. The GPU compute power on offer through Google Cloud was monumentally higher than what was obtainable through Rosetta Hub as p3.8 instances were disabled on Rosetta Hub. I got to know the Google Cloud SDK and was able to spin up instances which had 8 NVIDIA Tesla V100s in minutes. I learnt how to set up instance groups and use preemptable (spot) instances to keep the costs down.

What I Did

Practical 2

For the initial practical, I mainly got to know Rosetta Hub and JTR. I have several machines which I frequently use over SSH so this was pretty simple. I spun up the cheapest spot instance I could find and SSH'd in. The install of JTR was pretty trivial also. I ran the benchmark and compared it to my laptop and old desktop and the lack of compute power on the spot instance became very evident.

Practical 3

I started out this practical by doing some research into how people usually go about cracking passwords. I found a pretty useful article which went through the basics of using Hashcat to crack passwords. I continued using JTR instead of switching to Hashcat and just used the article to gain some understanding. The article suggested starting with the Battlefield 2015 wordlist, so that's what I initially tried with my hashes. This was a pretty small wordlist which didn't take long to run through at all and got a couple of passwords. I then spent some time looking into other wordlists and discovered that Rockyou.txt was a hugely popular wordlist. I then set off my local machine running a wordlist attack using JTR and Rockyou.

At this point, JTR was just using my CPU for cracking. After about 30 minutes I had cracked ~150 passwords. I decided to stop and look into setting it up with my GPU to see how much of a performance increase I would get. I found another useful article on how to use JTR with an NVIDIA GPU (which was what I had). This required installing some CUDA toolkits and updating my GPU drivers (which were pretty ancient, even for an 8 year old GPU!). This was surprisingly simple to set up and I could run JTR with my GPU by setting the format to `--format <hash_type>-opencl`. The second I did this, my screen filled up with hashes ridiculously quickly and I cracked all 1000 hashes in no time. The final thing I did for this practical was to get everything up and running on AWS. I spun up a spot of the cheapest GPU instance I could find and ran through the steps outlined in the assignment README. Everything went according to plan and I could crack the passwords even faster than on my home GPU.

Practical 4

This was a very challenging practical. Hashcat's refusal to run without specifying the hash algorithm forced me to spend some time researching how hash algorithms could be identified by their format. Fortunately, Hashcat's example hashes page made this pretty easy.

Up until this point, all of the hashes we had been given were in a single format. So the first step in the practical was partitioning the hashes by hash algorithm type. As specified in the assignment README, I knew there was 6 hash types. As I wanted to improve my bash knowledge and capabilities, I spent some time writing a simple script to partition a given list of hashes using REGEX. This function proved to be really useful for the remaining practicals. I ran this function and split the list of hashes into six separate files (one for each of the hash types). Comparing the partitioned hashes to the example hashes given on Hashcat's website, I noted the `-m` parameters for each of hash types.

For each of the practicals, I used a specific potfile. I version controlled these potfiles using Git/GitHub as these were essentially the deliverable for the entire assignment and if something happened to them, I would lose all of my progress. This worked quite well as I didn't come across a need to have multiple machines attempting to crack the same hash types. That is, each machine I was running was only cracking passwords for one hash type. Otherwise, I would have required a more real-time potfile syncing mechanism.

The next step was to determine which set of hashes to attempt first. Although my knowledge of hash algorithms was rather limited, I knew DES was simpler than MD5 and MD5 was simpler than SHA. I was unsure of where PBKDF2 / Argon would fit into things, so I ran a Hashcat benchmark for each of the hash types and determined the optimum cracking order to be: DES, MD5, SHA256, SHA512, PBKDF2, Argon.

I then began the long process of trying to crack passwords. The first thing I tried was just running Rockyou against DES on my GPU at home. This worked quite well and didn't take long to find 80/349 of the DES hashes. I had been using a simple one liner:

```
awk -F ":" '{print $1, $2}' hashcat.potfile > hashes.broken
```

for reformating the potfiles. However when I went to submit the 80 hashes, Submitty didn't seem to accept them. My first guess was that there may have been hash collisions, so I removed the troublesome hashes from Rockyou and the potfile and re-ran it against DES. This didn't get any more hashes and we later learnt there was a bug in the submitty checker. To get around this I wrote a simple reformat function which would tidy up my potfile in a submitty-friendly way.

Up until this point, I had only used Rockyou for each of the previous assignments. I decided to try and use some rules along with Rockyou to see if that would obtain more hashes. I used the Hob064 which hugely increased the time required to run through Rockyou to 4 hours for DES. I ran this for a few minutes and got a few hashes but ultimately abandoned the idea. In hindsight, this was a terrible idea since Rockyou contains modified versions of base passwords anyway.

I noticed Hashcat was complaining about my (ancient) GPU's CUDA compatibility version of 2.0, so I decided to switch over to Rosetta Hub. I spent some time looking into the GPU instances offered by AWS and determined that p3.x instances were the obvious choice as they contained Tesla V100s. Unfortunately, Rosetta Hub doesn't give us access to these instances, so I was stuck with g2/g3 instances which are designed for graphic-intensive applications. I began using a g2.8 which contains four NVIDIA GRID GPUs. I saw a speedup of ~3.5-4 times over running on my desktop at home which only really accounted for the extra GPUs the machine had.

After running most of Rockyou against each of the hash types, I decided to examine the ~300 passwords I had cracked so far to see if there was any obvious patterns. I noticed that 5 letter lowercase and compound words (e.g. two concatenated english words) seemed to appear quite frequently and decided to go after those next.

As the compound words seemed to be english-like, I got an english dictionary wordlist and filtered it down to words of length 4. I then ran a combinator attack with both source lists pointing to this file, generating 8 character compound words. This was still quite a substantial wordlist to check but it performed quite well on the faster hash algorithms (DES, MD5).

For the five character lowercase passwords, I initially tried a brute force attack. Again this worked well enough for DES and MD5 but was not feasible for the slower hash algorithms. At this point I had several hundred hashes based on the above process. We were then given clues for the sources of the three password types. We determined that the three sources were Rockyou in its entirety, the UNIX dictionary file for the 4 character words which yielded the 8 character compound words and the `pwgen` command.

In order to attempt to solve the remaining hashes, Rockyou needed to be run to completion on each of the hash algorithms. To achieve this in a somewhat reasonable time, I investigated all of the available compute instances and summarized the results in the following table (kH/s is 1000 hashes per second):

Instance Type	GPU(s)	Rosetta Hub	kH/s	Spot €/hour	kH/s / €
p2.x	1x K80	Y	259	0.30	863
p3.2	1x Tesla V100	N	2777	0.75	2805
p3.8	4x Tesla V100	N	11100	2.61	4253
g2.8	4x Grid	Y	282	0.85	332
g3.8	2x Tesla M60	Y	1147	0.72	1593

As seen in the table, the p3.8 instance with 4 Tesla V100s was the optimum choice. Unfortunately, these were not available through Rosetta Hub. The best choice for password cracking on Rosetta Hub was the g3.8 which is ~10x slower and is only ~3.6 times cheaper than the p3.8. As such, Google Cloud was used to gain access to 4x Tesla V100s.

Rockyou was run to completion on the 5 fastest hash algorithms - DES, MD5, SHA256, SHA512 and PBKDF2. However as Argon was designed to be GPU resistant, I attempted to crack these using several CPU optimized instances.

In order to crack the hashes generated using `pwgen`, I wrote a function which generates lowercase, pronounceable passwords of length 5. This wordlist of approximately 300000 passwords was then run through each list of hashes.

Finally in order to crack the final passwords, all of the 4 character words in the UNIX dictionary were extracted and used as both source lists for a combinator attack, giving the 8 character compound words.

Inferno Ball

The inferno ball practical turned out to be rather simple for all layers apart from the crackstation layer. Initially it looked like a lot of orchestration would be required as even the "fast" algorithms were very slow due to the tweaked parameters (e.g. number of rounds). Some basic benchmarking was done on each of the tweaked hash algorithms in order to find the optimum cracking order of pbkdf2, sha1, sha512.

Aron Hoffmann wrote some python scripts to handle checking the secrets and descending into inner layers. We also began writing some orchestration code. However, by the time the code was taking shape, the easter eggs had been discovered and there was simply no point in spending any time trying to write orchestration code. Instead, Aron could simply run the relatively small wordlists on his GTX 1080 and we progressed through the first six layers very quickly.

Layer 7 proved to be a challenge and required some research into keyboard walks. I began generating wordlists using Hashcat's kwprocessor. This was quite challenging as the algorithms were very slow and the keyboard walks could have been anything. Ultimately however, I found a premade keyboard-walk wordlist online which got us through the layer.

The final challenging layer was the Crackstation layer. Crackstation is a massive wordlist and required a colossal amount of compute power. However as I was the only member of my team with a Google Cloud account, I solved the layer alone simply by running 4 Tesla V100s for several days and spending over €120 of my €260 free Google Cloud budget. I initially considered switching back to Rosetta Hub to burn through

the teams remaining budgets, but after re-examining the performance table above, it became clear that this wasn't worth the time. As we had plenty of time remaining, I just opted to leave Google Cloud running for a few days.

Module Evaluation

What I Liked

I enjoyed most of the guest lecture content, especially the lectures surrounding GPUs and OpenCL. I also enjoyed Stephen's security lecture toward the end of the semester and it peaked my interest for the Security module next semester. The practicals were a great idea and have the potential to be great assignments! Understanding and building distributed, scalable systems is obviously extremely important and having access to €100 worth of compute power has the potential for some really interesting projects.

What I Disliked

I understand that this is the first year of the module and that there was far more students taking this module than anticipated. However, in my opinion the module was very unsatisfactory.

I felt that the lecture content was extremely lacking. I appreciate that different lectures have different styles of teaching, but in my opinion the lectures were very disorganized and unprepared. There was essentially no source material and the lectures seemed to just be ramblings from one topic to another.

Unfortunately the assignments ended up having absolutely nothing to do with scalability and absolutely everything to do with figuring out which password lists were used to generate the hashes. There was no benefit whatsoever in trying to build a system at all, nevermind a scalable one.