

CREATING DICTIONARY ATTACK SOFTWARE USING A POWERFUL SERVER AND JAVAFX.

Author: Isaiah Dicristoforo

Project Advisor: Bill Nicholson, Assistant Professor, IT Program, UC Clermont

April 13, 2020

UC UNDERGRADUATE SCHOLARLY SHOWCASE 2020

Research Question

How can a high-powered server, multi-threading, and various dictionary attack word lists be used to develop a fast and informative password cracking tool?

Abstract

The goal of this project was to create a password cracking tool and measure its effectiveness when deployed on a high-powered server. The password cracking tool was built using the Java FX framework and implements various multi-threading techniques to maximize the speed at which it cracks passwords. The password cracking tool was also created to deliver informative results to its end-user and contains a comprehensive interface that provides statistics about the passwords that the tool cracks. By performing identical tests on a laptop and the high-powered server, the performance increase of running the software on the server was able to be measured. Attempts were made to further increase the performance of the password cracker by executing code on the server's graphics processing units (GPUs). Experimentation with several libraries built for Java GPU programming resulted in the successful implementation of basic Java code that could run on a GPU; however, several limitations prevented the use of GPU programming for the password cracking software. Ultimately, however, the software developed for this project turned out to be both a fast password cracker and a beneficial tool for providing analysis about the characteristics found in cracked passwords.

Methodology

Overall Goal

When developing the dictionary attack software, there were two objectives in mind.

First, there was a need to create a fast application that could spawn hundreds of threads to utilize the full computing power of a machine. The second objective was to give the user of the password cracker helpful feedback and analysis about the passwords that they cracked, to help them pinpoint specific characteristics found in weak passwords. The results generated by the password tool could help a user strengthen their passwords or aid a cybersecurity professional in strengthening their password policy. This second objective was, in part, the impetus for the decision to design a graphical user interface.

User Interface Vs. Command Line

In short, the decision to design a graphical user interface for this project was made to provide the user with a more accessible and informative experience than they would have received using a command-line application. Many common password cracking tools used today, like John the Ripper, are command-line utilities. The graphical user interface in this project was designed to see how command-line tools like John the Ripper would be realized as a user interface. The user interface was designed as a

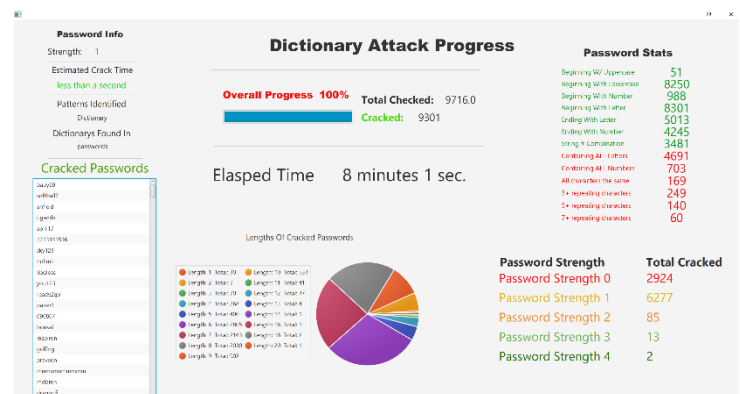
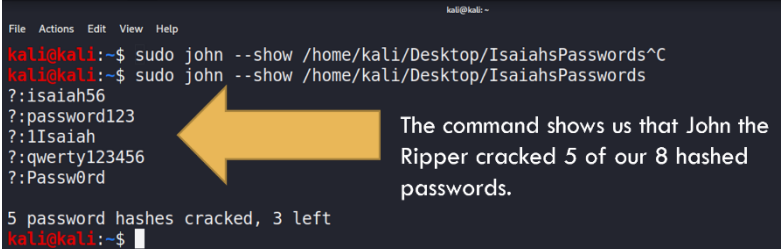


Figure 1: The password cracking user interface created for this project.

more accessible alternative to command-line utilities, which often require the user to memorize advanced commands in order to use them. However, while a user interface can provide someone with more informative and visually appealing data than a command-line utility can, there is a speed-trade off that must be taken into consideration. Is it wise to waste resources updating a chart, and a timer on the screen? That question exemplifies the challenge faced when building the user interface for this project: how does a developer walk the line between developing a fast application, while spending time doing arguably unnecessary updates to the UI that provide the user interesting visual feedback?

A password cracking tool like John the Ripper dedicates its resources to cracking passwords as fast as possible but provides very little analysis. To demonstrate this, I created a small eight-word password list, hashed it, and passed the file into a version of John the Ripper running on a Kali Linux virtual machine. John the Ripper recognized the hashing algorithm used and ran the hashes



```
kali@kali: ~  
File Actions Edit View Help  
kali@kali:~$ sudo john --show /home/kali/Desktop/IsaiahsPasswords^C  
kali@kali:~$ sudo john --show /home/kali/Desktop/IsaiahsPasswords  
?:isaiah56  
?:password123  
?:lisaiah  
?:qwerty123456  
?:Passw0rd  
  
5 password hashes cracked, 3 left  
kali@kali:~$
```

The command shows us that John the Ripper cracked 5 of our 8 hashed passwords.

Figure 2: John the Ripper Test

against its default word list. I used the *show* command to display a list of passwords the tool had cracked, and they appeared on the terminal. There was a notable absence of analysis and statistics about the dictionary attack I had just performed. A black-hat hacker might only care about breaking into a system, but a security analyst might want to know some information about the passwords they crack, such as their entropy, and common patterns found in the cracked passwords. That premise was the reason that the password cracking tool built for this project includes so much information about the passwords that the tool cracks. For example,

the password cracking tool integrates some existing open-source software that measures the strength of a password.

Designing A Dictionary Attack

Two common methods of cracking passwords are dictionary and brute force attacks. This project's scope was limited to dictionary attacks. A 2018 study published by the Institute of Electrical and Electronics Engineers (IEEE), titled *Brute-force and dictionary attacks on hashed real-world passwords*, revealed that "dictionary attacks with the exception of larger hybrid approaches are must faster than the brute-force method" (Bosnjak et al. 1161). Dictionary attacks are not only faster but are an intriguing field of study because of the social engineering component. Dictionary attacks are designed to target common patterns users create when constructing a password. For this project, an effort was made to only utilize dictionary lists that had been tested and analyzed in other studies by cyber security professionals.

Two dictionary attack word lists were used with the password cracking tool. First, the rockyou.txt word list was used when performing quicker password attacks. This word list is included on Kali Linux distributions, and widely available online. In a study titled *Cracking More Password Hashes with Patterns*, Engineering professor, Emin Tatli, from Istanbul Medipol University, Turkey, writes that this word list surfaced because a data breach of a website, rockyou.com (1656). Tatli writes that "In the past, security researchers did not have such a large real-life resource for password analysis. Therefore, the published 32.6 million real-life passwords have become very valuable data for security experts and researchers" (1656).

The rockyou.txt word list contains 14,344,188 unique passwords. The password cracking tool developed for this project contains a dictionary analysis module that calculates the total occurrences of common special characters, and determines word length distribution of the words in the dictionary. Figure 2 is an image from the dictionary analysis tool. All of the charts were generated programatically using Java.

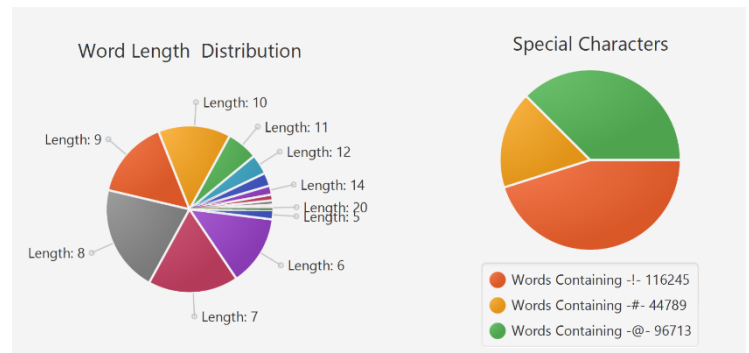


Figure 3 Rockyou.txt dictionary analysis tool.

The second word list used to perform dictionary attacks with is called Rocktastic.txt. The word list is curated by the cybersecurity firm Nettitude. While as the RockYou word list contains 14 million passwords, the Rocktastic word list contains over 1.1 billion. According to the dictionary analysis tool, the most common word length in the Rocktastic word list was 10, followed by 9, then 11. In contrast, the most common word lengths in the shorter RockYou word list, were 8, 7, and 9. These findings suggest that the Rocktastic word list may be more effective at cracking longer passwords.

Multi-threading

Multi-threading is an integral piece of the password cracking tool. The use of multi-threading allows the password cracking tool to crack more than one password at a time. It was imperative when developing the graphical user interface for this tool that all password cracking be performed on a separate thread than the JavaFX application thread (the GUI thread). If this is not the case, the user interface would completely freeze up.

An additional challenge was that the user interface needed to update in real time as passwords were cracked. According to the JavaFx Documentation, "The JavaFx scene graph, which represents the graphical user interface of a JavaFX application, is not thread-safe and can only be accessed and modified from the UI thread." This problem can be solved by using the JavaFX concurrent package. The JavaFX documentation says that "The JavaFX concurrent package leverages the existing API by considering the JavaFX Application thread and other constraints faced by GUI developers". The JavaFX concurrent package is the backbone for most of the multi-threading techniques implemented in the password cracking tool. It contained useful properties for updating progress bars, and other GUI components; however, it had its limitations, and provided many challenges when designing the user interface that updated in real time.

Two multi-threading techniques were implemented in the password cracking tool to maximize the number of passwords the tool could crack at a time. The tool has the capability to check an n number of passwords at a time where n is a user specified number of passwords. The program will spawn one thread to check each password. That thread can in turn, spawn more threads to search from a specific location in a word list. For example, one thread could iterate over the first fourth of the dictionary, another could start searching from three fourths into the list, and so on. This method proved costly in some instances because it takes time to navigate to a certain position in a word list (in this case, the dictionary lists used to crack passwords were stored in a file). Consideration was given to loading the word lists into an in-memory data structure like a hash table instead of reading from a file, but this would have been unrealistic for the Rocktastic word list which contained over one-billion words. Furthermore,

development of this project was largely constrained to a laptop and its limited memory, as frequent interaction with the server over a VPN proved slow and difficult to work with.

The multi-threading algorithms used in this project dramatically increase the speed at which the password cracker can attempt to crack passwords. The current algorithm used is strong but has some areas for improvement that could further increase the speed of the software. Ultimately, the current iteration of the password cracking tool is both fast and provides data in real-time to the user that is both accurate and informative. A good foundation has been built for future additions to the project.

Hash Compatibility

The current iteration of the password cracking tool only works with SHA-1 password hashes and does not expect that the hashed passwords will be salted. This is a shortcoming compared to John the Ripper. John the Ripper can recognize the type of a password hash and hash its dictionary attack list accordingly. This should be an improvement made in later versions of the password cracking tool. The strategy design pattern is an object-oriented design pattern that would be helpful in implementing this functionality.

Adding Project Dependencies

Much of the password analysis displayed on the user interface of the password cracking tool was created from scratch. There is functionality to track the lengths of cracked passwords, the total passwords containing a number, and so on. In addition, the project integrates some existing software that measures password strength. Zxcvbn4j is the name of a popular password strength measuring tool on Git Hub. According to the project's ReadMe file, a typical

use case for the Zxcvbn4j tool is for a password strength meter typically found on a login page. The first iteration of this project was built with JavaScript, then reproduced in languages such as Java and Python. The project measures a password's strength by estimating the total number of guesses it would take to crack the password and assigns a password a number from zero to five depending, with zero being the weakest password. How does the software calculate the total number of estimated guesses to crack a password? According to their ReadMe file, "Through pattern matching and conservative estimation, it recognizes and weighs 30k common passwords, common names and surnames according to US census data, popular English words from Wikipedia and US television and movies, and other common patterns like dates, repeats(aaa), sequences (abcd), keyboard patterns (qwertyuiop), and l33t speak." This software is already implemented by various organizations including Jet Brains hub.

The Zxcvbn4j library integrated smoothly into the password cracking software as a Maven Dependency and provides valuable information to the user. Utilizing this library, the password cracker can aggregate the strength of each cracked password, display the estimated crack time, identify patterns such as repetition, give the user feedback as to how to increase their password strength, and even let the user know what other cracking dictionaries their password may be found in. This information is displayed on the user interface and can be written to a file along with the cracked password. In summary, integrating this software into the project added valuable feedback about the passwords that were cracked, while not dramatically compromising the speed of the password cracking program.

GPU Programming in Java

One of the goals for this project was to utilize the high-powered server I was given access to for this project to its full capacity. Max CPU utilization on the server was reached by spawning thousands of threads to attempt to crack multiple passwords at the same time. While the password cracker was able to utilize the server's full CPU capacity, the attempt to access the processing power of the GPUs proved more challenging. When first run on the server, the password cracking tool did not utilize the GPUs on the machine. JCuda, TornadoVM, and Aprapi were some of the libraries used when trying to execute Java code on the GPU.

The Aprarpi framework was the closest I got to executing code on the GPU for this project. The Aparapi website describes Aparapi as an "Open-source framework for executing native Java code on the GPU." This promising statement was undercut when reading the Aparapi documentation and discovering that, "Only the Java primitive data types, boolean, byte, short, int, long, and float and one-dimensional arrays of these primitive data types are supported by Aprarpi." After writing some code using the Aparapi framework, it became clear that any attempt to use a Java object would cause your program execution to fall back to the CPU. Regardless, using the Aprarpi framework, I was able to implement a prime number checker that utilized a GPU device. It became clear that the use of Aprapi framework for the password cracker was not feasible or intended. It was also evident by reading the documentation and looking at example code that frameworks like Aparapi, and Tornado VM were designed for repetitive calculations with primitive data types and did not include the functionality to work with objects. Similarly, a framework like Jcuda proved too difficult to

implement, and involved many low-level programming concepts that I did not have the knowledge to understand. Further discussion with my project advisor suggested that the inability to use these frameworks for the password cracker, was not a limitation of these frameworks, but rather a limitation of the capabilities of GPU programming.

Finding Passwords to Crack

To perform tests on the password cracking software, I needed some passwords to crack. I built a password generation tool to aid in this purpose. It is integrated into the password cracking software. The password generation tool allows you to generate passwords from a variety of sources. Users can generate passwords from a list of dictionary words, names, or a random combination of characters, *and* have the option to append or prepend a random number of characters to the strings generated from the word list. The user of the password cracker may generate a password list with their tool or select a hashed or un-hashed version of their own. Another password source I used was haveibeenpwned.com where the site says their word list contains "555,278,657 real world hashed passwords previously exposed in data breaches."

Testing and Results

Laptop Vs. Server Performance Test

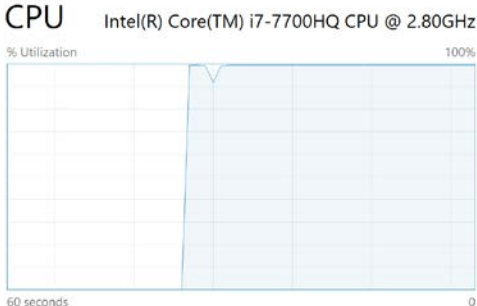
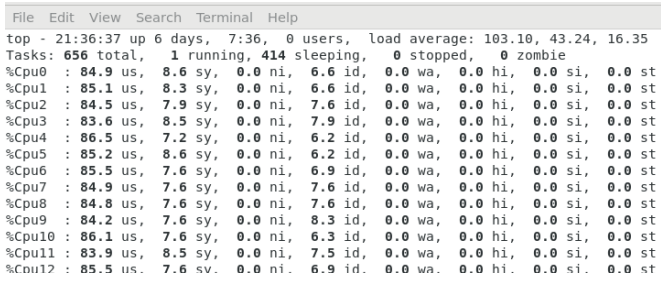
To compare the speed of the server with the speed of a standard computer, identical tests were performed on a laptop with an I7 processor and 8 logical processors, and the server, which contained 28 CPU cores.

Dictionary attack List: Rockyou (14,344,188 words)

Password List: *Generated with my password generation tool*

- 500 human names with no characters appended or prepended
- 500 human names with 2 random numbers appended
- 500 random dictionary words with no characters prepended or appended
- 1000 dictionary words with one random number appended
- 1000 dictionary words with one random letter, number, or special character prepended

Threads: Both the laptop and server were configured to crack 100 passwords at a time.

Laptop CPU Utilization	Linux Server CPU Utilization
	
Laptop: <i>Time to Complete</i> 44 minutes, 2 seconds	Server: <i>Time to Complete</i> 7 minutes, 17 seconds

These results illustrate the significant performance increase in the server when cracking passwords. In this case, the server was 83.46% faster than the laptop when identical tests were performed. While the laptop easily reached 100 percent CPU utilization, the server did not expend all its resources. 543 or 15.51 percent out of the 3,500 passwords were cracked.

Password Strength	Total Cracked	Beginning W/ Uppercase	231	Containing ALL Letters	416
Password Strength 0	94	Beginning With Lowercase	310	Containing ALL Numbers	0
Password Strength 1	404	Beginning With Number	2	All characters the same	1
Password Strength 2	36	Beginning With Letter	541	3+ repeating characters	0
Password Strength 3	6	Ending With Letter	420	5+ repeating characters	0
Password Strength 4	3	Ending With Number	123	7+ repeating characters	0
		String # Combination	123		

Figure 4: Password analysis from the 543 cracked passwords. Screenshots are taken from the dictionary attack GUI.

Rockyou vs Rocktastic word list.

Using the same generated password list used in the laptop and server performance test above, the test was repeated, only this time using the Rocktastic dictionary attack list containing over one billion hashed passwords. Similarly, only 100 threads were spawned on the server to replicate the previous test with the smaller Rockyou word list.

The results, shown below, indicate that the larger Rocktastic word list cracked 955, or 75.87% more passwords than the rockyou word list. This is expected, considering the size difference between the two-word lists. While the Rocktastic word list was able to crack more passwords, the dictionary attack took 500.8 minutes, or 6764.8% longer to complete than when the Rockyou word list was used. This is in part due to the fact that only 100 threads were spawned to replicate the test above, and also a result of the massive size of the Rocktastic word list. In summary, these results demonstrate that the Rocktastic word list comes with a large time trade-off but is capable of cracking more passwords.

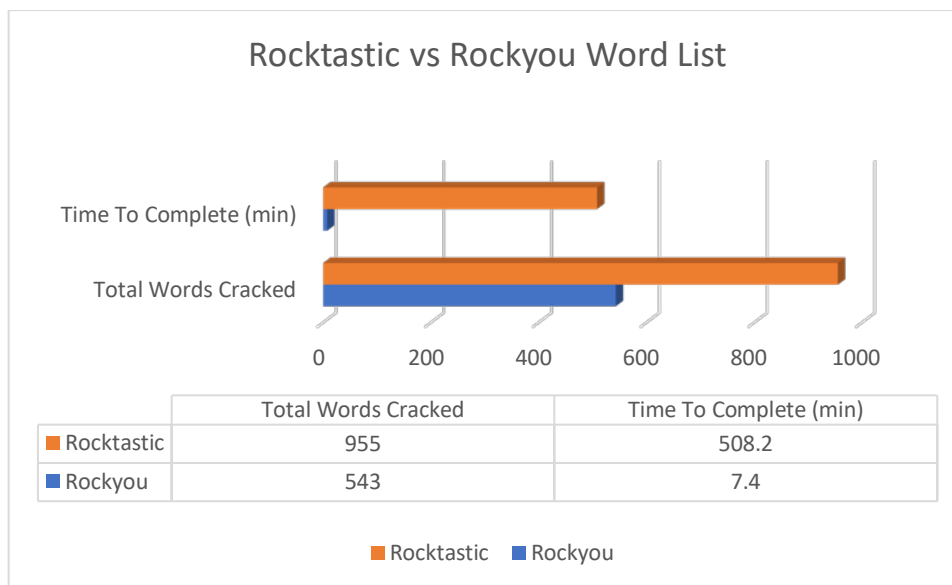


Figure 5: Side by side comparison of the same test performed with different word lists.

100,000 Passwords from the PwnedPasswordList

For this test I extracted one-hundred-thousand passwords from the *pwnedPasswordList* and ran a dictionary attack on the Linux server. The rockyou dictionary was used for this test, and the program was configured to spawn two-thousand threads at once.

Time to Complete Attack: 21 Minutes, 45 seconds.

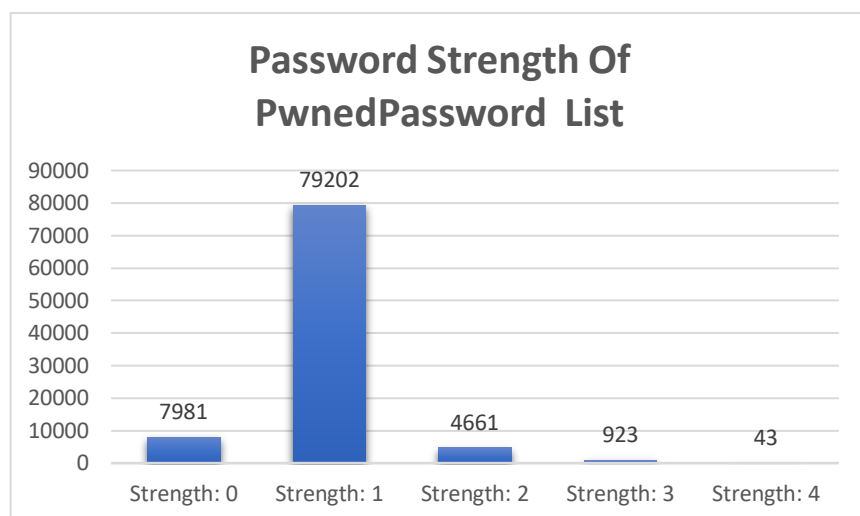
Total Cracked Passwords: 92,810

Total Not Found: 7,190

The results below utilize some of the feedback generated by the user interface of the password cracking tool. The results of this dictionary attack confirm that the hashed passwords in the pwnedpassword list are weak and would *not* meet the password requirements generated by a website or organization. For instance, 33% of the cracked passwords from the list contained letters only, and no occurrence of numbers or special characters. A significant 21% of the cracked passwords were comprised entirely of numbers. When creating these passwords users likely may have based these numerical passwords off phone numbers or dates of birth. The string-number password combination was also identified by the password cracking tool as a common password pattern. As indicated in the results set below, the string number combination is where a password is comprised of a string of any length, followed by any number of a random length. Looking at the individual passwords that were cracked revealed that this pattern was commonly used to append characters to someone's name (i.e. Isaiah123).

Passwords Including Letters Only	30,863
Passwords Comprised Entirely of Numbers	20,560
3 or more repeating characters ex: Isaiah111	2490
String-Number Combination (a string of any length followed by a number of any length)	36741

By integrating the password strength measurer into the password cracking code, the password cracking



tool was able to calculate the password strength of each cracked password, where password strength is the estimated number of guesses to crack a password. The results of this dictionary attack illustrate that most of the cracked passwords has a password strength of 1. A security analyst could use the password strength tool for testing out the efficiency of

their organization's password policy.

Conclusion

The password cracking tool developed for this project meets the original objectives of being both a fast password cracker that also provides informative feedback to a user about their cracked passwords by integrating new and existing software. Some areas of expansion for this project include extending the types of hashes and encryption types that are compatible with the password cracking tool. In addition, new ways of writing Java code that can be executed on a GPU should be explored to maximize the speed and efficiency of the password cracking tool.

Demonstration

A demonstration of the dictionary attack software is available in a YouTube video (link below).

<https://www.youtube.com/watch?v=GsvxYYNS3g0>

Works Cited

- Bošnjak, L., et al. "Brute-Force and Dictionary Attack on Hashed Real-World Passwords." *Brute-Force and Dictionary Attack on Hashed Real-World Passwords - IEEE Conference Publication*, 2018, ieeexplore.ieee.org/document/8400211.
- E. İ. Tatlı, "Cracking More Password Hashes With Patterns," in *IEEE Transactions on Information Forensics and Security*, vol. 10, no. 8, pp. 1656-1665, Aug. 2015.
- Fedortsova, Irina. "Release: JavaFX 2.1." *Concurrency in JavaFX / JavaFX 2 Tutorials and Documentation*, 2 June 2012, docs.oracle.com/javafx/2/threads/jfxpub-threads.htm.
- "Getting Started." *Aparapi*, aparapi.com/introduction/getting-started.html.
- "Jcuda.org." *Java Bindings for CUDA*, www.jcuda.org/.
- Nulab. "Nulab/zxcvbn4j." *GitHub*, Nulab Inc, 19 Dec. 2019, github.com/nulab/zxcvbn4j.
- Nettitude Labs. "Rocktastic: a Word List on Steroids." *Nettitude Labs*, Nettitude Labs https://labs.nettitude.com/Wp-Content/Uploads/2019/10/NETT_LABS_LOGO-New.png, 11 Oct. 2018, labs.nettitude.com/blog/rocktastic/.
- P.W.D. Charles, Aparapi, (2013), GitHub repository, <https://github.com/charlespwd/project-title>