Preventing ReDoS Attacks

Analyzing eval times of Regex engines

Jake Roggenbuck

Introduction & Motivation

- Over **one-third** of all software projects use Regular Expressions (Regex)[1]
- A Regex Denial of Service (ReDoS) attack occurs when a malicious input is evaluated causing the execution time to take arbitrarily long
- Most programming languages implement Regex evaluators that are **vulnerable** to ReDoS attacks
- We explore the performance of Regex evaluation to confirm that these evaluators are vulnerable to ReDoS attacks

Methods & Results

Surveying documentation for programming languages show that **only a few languages include linear time Regex compilers** that prevent many of the catastrophic backtracking attacks. Languages like Rust specifically exclude features like "look-around" and "backreferences"[2]. This allows them to use Thompson's construction algorithm to evaluate Regex in linear time[3]. This is in contrast with other evaluators that run in exponential time in the worst case.

The Regex `^(a+)+$` checks for many groups of many "a"s and in doing so, causes catastrophic backtracking. This Regex was tested with an input consisting of 30 "a" characters followed by a capital "B".

This Regex was evaluated in several popular languages and produced the following table.

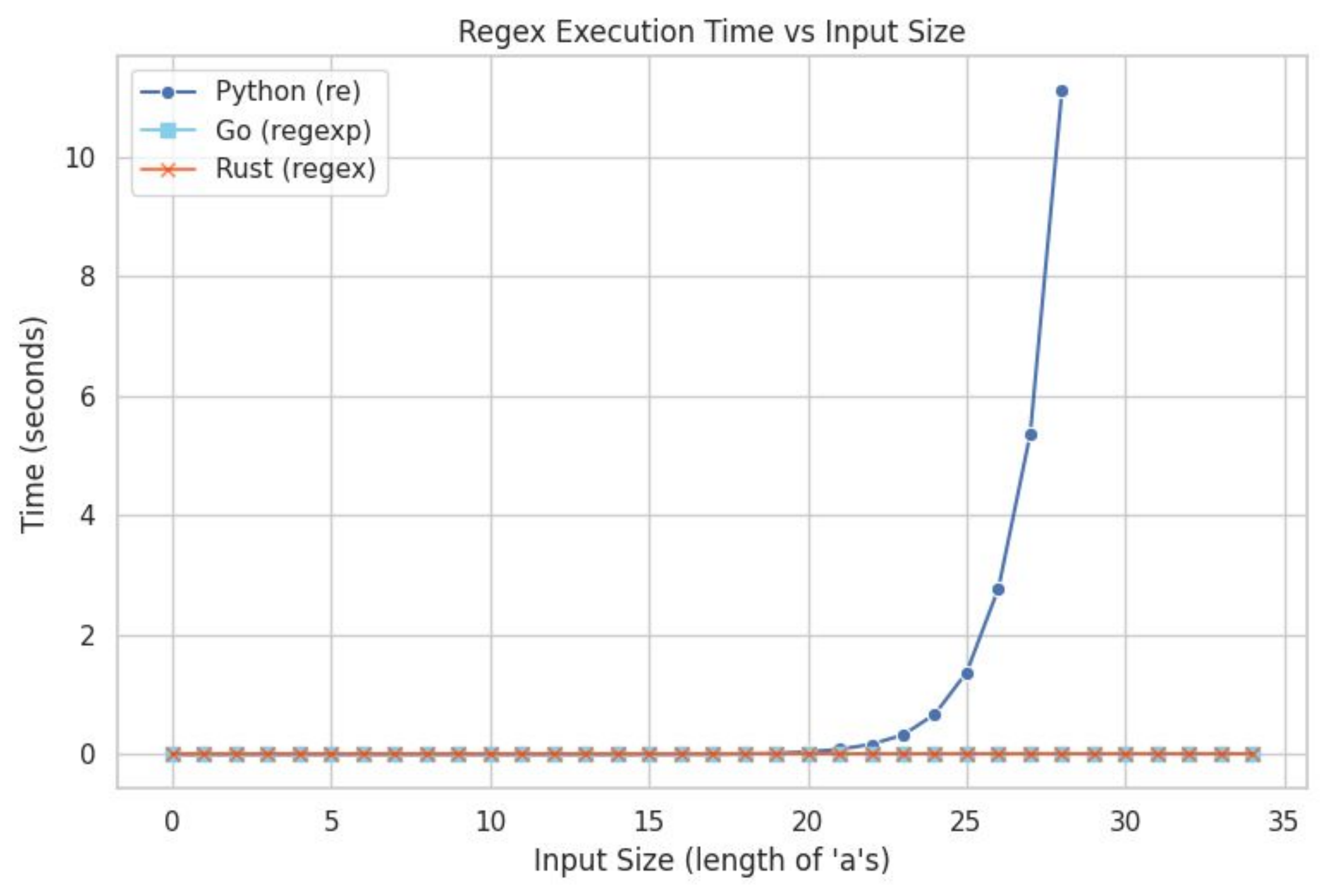
Language	Linear?	Time
Go (Regexp)	Yes	11.822μs
JavaScript (Node)	No	42.400s
Python (Re)	No	48.016s
Rust (Regex)	Yes	24.626μs
TypeScript (Deno)	No	56.534s

For the languages with execution times in the dozens of seconds for input size 30, ReDoS attacks are possible. Since the evaluators run in exponential time, increasing the input size slightly can bring services written with these types of Regex to a halt.

In addition to testing previously found Regex patterns, I created a program to search through the permutations with repeats the characters `^()ab+*$|?`.

I checked Regex patterns found in the source code of software to look for patterns that might be vulnerable.

This graph shows how the evaluation time of the Regex increases as the size of the input increases for an implementation in Python using the built-in Regex library called "re" for the pattern `^(a+)+$`



The equivalent implementation written in Rust and Go have a linear relationship with input size.

Searching through permutations of possible Regex expressions resulted in finding the pattern `^(a*)+$` that caused catastrophic backtracking and had an execution time 3x of the previous pattern.

When looking for vulnerable patterns in source code, I was able to find a pattern that could have text injected into the Regex, causing a ReDoS attack. I then wrote a pull request to fix the issue that got approved and it is now live in production.

Conclusion

Linear time Regex evaluators have been have implemented into a few languages however, they are not standard for many popular languages like Python or JavaScript. This leaves applications written in there languages vulnerable to ReDoS attacks. The tradeoff of not including functionality like "look-around" and "backreferences" makes sense from a security perspective.

In the future, other languages should adopt a safe mode for Regex that implements a linear time Regex engine to prevent these ReDoS attacks in production code.

Resources

1. Davis, J.C., Moyer, D., Kazerouni, A.M., & Lee, D. (2019). Testing Regex Generalizability And Its Implications: A Large-Scale Many-Language Measurement Study. 2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE), 427-439.
2. Gallant A. An implementation of regular expressions for Rust. https://docs.rs/regex/1.11.1/regex/index.html
3. Aho AV, eds. Compilers: Principles, Techniques, & Tools. 2nd ed. Pearson/Addison Wesley; 2007.
4. Davis J.C, Coghlan CA, Servant F, Lee D. The impact of regular expression denial of service (ReDoS) in practice: an empirical study at the ecosystem scale. In: Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering. ACM; 2018:246-256. doi:10.1145/3236024.3236027