Programming Language Concepts

What is ReDoS, and what part does 'Evil RegEx' play?

ReDOS is a denial of service attack targeted due to algorithmically complex regular expressions that generally utilises backtracking engines to build non-deterministic finite automatons (NFA) and exhibits worst-case exponential matching behaviour. The complexity of an NFA is determined by the number of states it must match. If a mismatch occurs, it backtracks to a previous decision point to try another decision. Such backtracking leads the victim's CPU resources to be exhausted, which triggers a Denial of Service. Davis et al. (2021) state the up to ten percent of regular expressions exhibit ReDOS behaviour. Given that regular expressions are utilised in roughly 40% of all Python code (Davis, 2019), ReDOS significantly impacts the stability of applications that require regular expression engines.

"Evil Regex" consists of a prefix string, a pump string and a suffix string which work together to force the regex engine to evaluate redundant states during backtracking. Evil regex aims to exploit an engine's algorithm to trigger worst-case execution time and is usually achieved when the regular expression input is *ambiguous*.

What are common problems associated with regex?

Some common problems associated with regular expressions include:

- Complexity. Reading regular expression input is not straightforward and requires a high degree of understanding to construct and interpret the meaning of a regular expression. (Michael et al., 2019). Complexity can be exacerbated by lack of documentation of poor style, such as writing an expression across a single source line.
- Performance. Developers may incorrectly use syntax such as `(a-Z+)+` when they meant to use `([a-Z]+)`. The syntax of regular expressions may not be readily understandable to inexperienced developers, leading to performance issues or exponential worst-case time complexity.
- Portability. Numerous regular expression dialects exist, and executing one dialect in another engine may have unintended consequences (Michael et al., 2019). For example, not all regular expression engines use identical escape sequences across systems.

- Vulnerabilities. Several libraries exist in JavaScript and Python that contain vulnerabilities in their regular expression engines.
- Reuse. Many pattern matching solutions are domain-specific, so it is problematic for developers to locate existing, tried-and-tested patterns that meet their needs (Michael et al., 2019)

How can common problems with regex be mitigated?

Common problems can be addressed in regular expression engines by:

- Adding an execution timeout feature to regular expression engines, such as C#'s engine (Van Der Merwe et al., 2017).
- Leverage other regular expressions libraries that minimise ReDOS attacks, such as RE2.
- Leverage libraries such as safe-regex and rxxr2 (Bai et al., 2021) to check the safety of regular expressions.
- Convert NFA to a directed finite automaton (DFA) while reduces ambiguity in analysing states.

Common problems can be addressed in regular expression inputs by:

- Rewriting the input string to reduce ambiguity.
- Make use of atomic groups such that the regular expression engine need not match the identified subgroup again—this requires the engine to forget all previous backtracking data. However, Python's regular expression module does not support atomic groups.
- Utilise lookahead assertions that assert that at a given position in the input string, the remainder of the input contains a prefix that can be matched (Van Der Mewer et al., 2017).

Common problems can be addressed by external means such as:

- Automate the detection and blockage of long-running CPU requests from persistent IP addresses (Bai et al., 2021).
- Implement reactive payload recovery services such as RegExNet (Bai et al., 2021)
- Train developers to better understand and develop correct regular expressions (Staicu & Pradel, 2018).
- Limit the total size of a regular expression input.

How and why is regex useful for security?

Regular expressions are utilised in intrusion detection systems to analyse the content of data packets entering a network. A ReDOS at this level would leave the network vulnerable and open to other attacks (Van der Merwe et al., 2017). Similarly, Pacifico et al. (2021) reference regular expressions to provide application-aware network traffic classification (Pacifico et al., 2021) that classify the network traffic in real-time.

Capture groups can be leveraged for rewriting URLs on web servers so that insecure requests are directed to secure request locations or even clean up the URL and make it easy to remember.

Kothapalli and Mitchell (2018) consider the use of linkography to model behaviour cause and effect relationships between sequences of shell commands executed by attackers. Here sequences are grouped into abstractions which are defined by regular expressions.

References

Bai, Z., Wang, K., Zhu, H., Cao, Y. & Jin, X. (2021). Runtime Recovery of Web Applications under Zero-Day ReDoS Attacks. *IEEE Symposium on Security and Privacy (SP)*:1575-1588.

Davis, J.C. (2019). Rethinking Regex engines to address ReDoS. *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*:1256-1258.

Davis, J.C., Servant, F. & Lee, D. (2021) Using selective memoisation to defeat regular expression denial of service (ReDoS). *IEEE Symposium on Security and Privacy (SP): Los Alamitos, CA, USA:* 543-559.

Kothapalli, A. & Mitchell, R. (2018). Regex-Based Linkography Abstraction Refinement for Information Security. *Proceedings of the Fourth ACM International Workshop on Security and Privacy Analytics*: 1-7.

Michael, L.G., Donohue, J., Davis, J.C., Lee, D. & Servant, F. (2019). Regexes are hard: Decision-making, difficulties, and risks in programming regular expressions. *34th IEEE/ACM International Conference on Automated Software Engineering (ASE):* 415-426.

Pacífico, R.D., Castanho, M.S., Vieira, L.F., Vieira, M.A., Duarte, L.F. & Nacif, J.A. (2021). Application Layer Packet Classifier in Hardware. *IFIP/IEEE International Symposium on Integrated Network Management (IM):*515-522.

Staicu, C.A. & Pradel, M. (2018). Freezing the web: A study of redos vulnerabilities in javascript-based web servers. *27th {USENIX} Security Symposium ({USENIX} Security 18):*361-376.

Van Der Merwe, B., Weideman, N. & Berglund, M. (2017). Turning evil regexes harmless. *Proceedings of the South African Institute of Computer Scientists and Information Technologists*:1-10.