UNIT 4 ACTIVITY: PROGRAMMING LANGUAGE CONCEPTS (REDOS & EVIL REGEX)

QUESTIONS

- Read Larson (2018) and Weidman (n.d.) then answer the questions below, adding them as evidence to your **e-portfolio**. You may want to complete this activity in conjunction with or after completing Seminar 2 preparation.

1. What is ReDOS and what part do ‘Evil Regex’ play?
2. What are the common problems associated with the use of regex? How can these be mitigated?
3. How and why could regex be used as part of a security solution?

Regular expressions (Regex)

- Regular expressions are a popular mechanism for processing strings that are in a particular format such as dates, email addresses, and phone numbers.
- Regular expressions are used in searching, validating data on a web page form, and data processing.
- Regular expressions are specified using a concise language with punctuation marks representing different operations.

1. What is ReDOS and what part do ‘Evil Regex’ play?

- The **Regular expression Denial of Service (ReDoS)** "is a Denial of Service attack, that exploits the fact that most Regular Expression implementations may reach extreme situations that cause them to work very slowly (exponentially related to input size)" (Weidman, n.d).
- In other words, a **denial of service attack** is when an attacker makes an **online service slow down or become unavailable** to its users. During a ReDoS attack, a hacker produces a denial of service by **providing a regex engine with a string that takes a long time to evaluate**. Hackers do this by exploiting so-called "evil regex patterns.“
- **Evil Regex**: A Regex pattern which can get stuck on crafted input. **Evil Regex contains**: Grouping with repetition; Inside the repeated group: Repetition, Alternation with overlapping
- ReDOS: an algorithmic complexity attack that basically takes advantage of the fact that the regex engine will try out every possible permutation and combination of characters to find a match. **(Source:** https://sec.okta.com/articles/2020/04/attacking-evil-regex-understanding-regular-expression-denial-service)
- In simple words, If you have a certain string and you want the computer to tell you if it matches a certain pattern, the computer just tries every single combination until it finds a match or exhausts all the combinations to try from. This is called catastrophic backtracking. **(Source:** https://medium.com/@nitinpatel_20236/what-are-evil-regexes-7b21058c747e)
- In every layer of the WEB there are Regular Expressions, that might contain an **Evil Regex**. An attacker can hang a WEB-browser (on a computer or potentially also on a mobile device), hang a Web Application Firewall (WAF), attack a database, and even stack a vulnerable WEB server.
- **Source:** Weidman, A. (n.d.) Regular expression Denial of Service - ReDoS. https://owasp.org/www-community/attacks/Regular_expression_Denial_of_Service_-_ReDoS

2. What are the common problems associated with the use of regex? How can these be mitigated?

Common problems with Regex:

- the regular expression language has nuances, confusing and easy to forget rules that can lead to bugs : very few regular expressions fail to compile (e.g. while unbalanced parentheses will cause a failure, unbalanced braces [] will not; some symbols have different meanings in different situations (e.g the ^ symbol).
- They can lead to failed input validation, leaky firewalls, and even denial of service attacks

To mitigate this:

- checkers can be used to find regular expression bugs. (e.g ACRE , which is available for use, including the source code, at http://fac-staff.seattleu.edu/elarson/web/regex.htm, consists of a set of 11 checkers that focus on common mistakes when developing regular expressions Or use the EGRET (Evil Generation of Regular Expression Tests) tool)
- Use safe regex engines: Instead of using built-in, unsafe regex engines, use safealternatives instead
- Detect and sanitize evil regexes (use evil regex detection libraries like safe-regex).
- Sources:
- Larson, E. (2016) Generating Evil Test Strings for Regular Expressions. IEEE International Conference on Software Testing, Verification and Validation (ICST).
- Larson, E. (2018) Automatic Checking of Regular Expressions. 18th IEEE International Working Conference on Source Code Analysis and Manipulation (SCAM)
- Y. Liu, M. Zhang and W. Meng, "Revealer: Detecting and Exploiting Regular Expression Denial-of-Service Vulnerabilities," *2021 IEEE Symposium on Security and Privacy (SP)*, 2021, pp. 1468-1484, doi: 10.1109/SP40001.2021.00062.
- https://www.dropbox.com/scl/fi/pnyjn01ay546ny98n0pj8/1-Activity-Programming-language-concepts.docx?dl=0&rlkey=jfb0v4t0h36s61cl7rkrmkgc6

3. How and why could regex be used as part of a security solution?

- How can we make regexes safe without breaking existing software?
- The authors propose incorporating a state cache into existing Spencer-based regex engines. This approach has two desirable properties. First, it will offer the same worst-case linear-time complexity as Thompson's algorithm, albeit with larger space complexity. Second, it will not change the match/mismatch behavior of an engine, merely the time and space required to reach the conclusion.

- Source: Davis, J.C. (2019) **Rethinking Regex engines to address ReDoS,** ESEC/FSE 2019: Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, p. 1256–1258. Available from: https://doi.org/10.1145/3338906.3342509