CS5331 Web Security – ReDOS

Group 18 Members – Xiao Yikai, Zhang Xinyue, Huang Kun, Miao Anbang
School of Computing, National University of Singapore

Introduction

- **Denial-of-Service Attack (DOS Attack)** seeks to make resource unavailable to its intended user
- **Regular Expression Denial of Service (ReDOS**) exploits on imperfect implementations using regular expression with special designed patterns, to cause the target to respond very slowly or even temporally not able to provide service to intended users.
- **Evil Regex** is the pattern that may cause this vulnerability. With malicious string, attacker can perform ReDOS to target website because it is using Evil Regex.

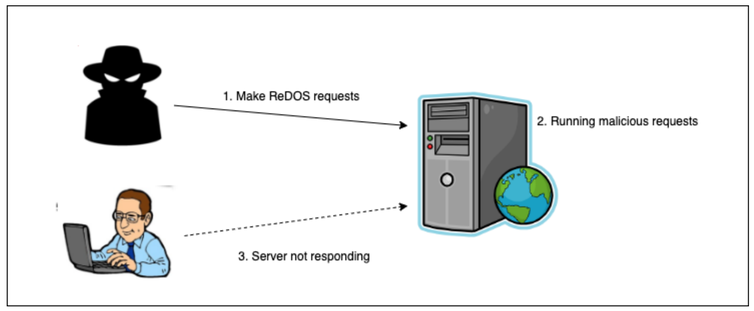
Objective

- Our project aims to **identify the logic** of Regex checking and its vulnerability of being attacked
- Demonstrate how attackers can **attack on vulnerable servers** and occupy the resources of target machines.
- Figure out and verify **solutions** to this ReDOS attack

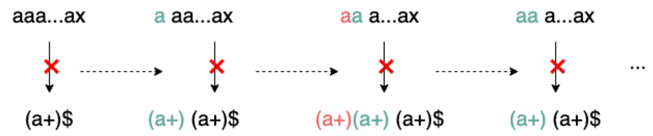
ReDOS Attack Experiment

- **Web Server:** PHP web server with simple function is deployed. This server simulates a login page and it verifies username input with Evil Regex. Both legal the illegal message will be returned to client who calls it.
- **Normal Client:** Web browser (specifically Firefox) is used as normal client to request.
- **Vulnerability Detection:** Web browser is used to detect the vulnerability by requesting using malicious string.
- **DOS Attack:** A bash script is used to attack server. Requests are sent in group of 1000. New group of requests are sent after the previous group finishes.
- **Observation:** Two observation methods are adopted. One is to observe the resources on the server machine. Another one is to observe response time from a separated request.
- **Solution Verify:** A new PHP web server with solution (avoid using Evil Regex) is deployed. Same ReDOS attack is conducted. And same observations are done.

Regex Logic & Attack Logic



String: aaa...ax
Pattern: /(a+)+$/



Failure Attempts

ax

ax	(a+)$

aax

aax	(a+)$
aax	(a+)(a+)$

aaax

aaax	(a+)$
aaax	(a+)(a+)$
aaax	(a+)(a+)(a+)$
aaax	(a+)(a+)$

Time spending:
- ax: T(1)
- aax: 2*T(1)
- aaax: 4*T(1)
...
- a...ax: 2^(n-1)T(1)

Implementation & Results:

- **Bash script for attack:**
  - Group the following task is sending

```
task(){
    curl -s "http://192.168.154.132/redos/index.php?email=aaaaaaaaaaaaaa!&password=test"
}
```

- **Evil Regex Preparation:**
  - Read “email” and “password”
  - Check “email” using Evil Regex
  - Return Checking result

```
<?php
    $p = '/(a+)+$/';
    if(isset($_GET['email']) && isset($_GET['password'])){
        $matched = false;
        $email = $_GET['email'];
        $password = $_GET['password'];
        if((!empty($email)) && (!empty($password))){
            if(preg_match($p, $email)){
                $matched=true;
            }
            if ($matched){
                echo "<p>Hello {$email} </p>";
            }else{
                echo "<p>Ops, invalid email </p>";
            }
        }else{
            echo "<p>Value cannot be empty!!!</p>";
        }
    }
```

- **Resource Monitoring**
  - Use Linux resource monitoring to observe the result

With Evil Regex

Without Evil Regex (Solution)

Solutions

- **Input Validation (check the length of string before checking Regex):** Since the time is increasing exponentially with the length of the string, a possible way of preventing the attack is to filter those long inputs.
- **Flaw Checking (avoid using vulnerable Regex pattern):** It is not possible to avoid using Regex, but it is possible to avoid using Evil Regex patterns. Make sure not using patterns like /(a+)+$/ or some hard to identify ones like /([a-z, 0-9]+)+$/

Conclusion and Future Work

- **ReDOS Attack** can be reproduced on **PHP server** with vulnerable pattern and string.
- We have also tested the Evil Regex pattern and string using other language, namely JavaScript and GoLang. It shows that **JavaScript has the same issue** but **GoLang does not**. Future explore can be conducted on different Regex implementation in different language.
- **Other Application DOS attacks** caused by server logic can also be further studied, eg. PHP hash collision.