CHECKMARX

SOURCE CODE ANALYSIS TECHNOLOGIES

VAC – ReDoS

Regular Expression Denial Of Service

Alex Roichman

Adar Weidman

December 2009

Overview

- *"Everybody knows…"*
  - Good(?) **old** Regular expressions problems
  - Good(?) **old** DoS

- Change of perspective:
  - **New** attitude – not a bug — **V**ulnerability
  - **New** examples & demonstration — **A**ttack
  - **New** ways to deal with it — **C**ountermeasures

Vulnerability

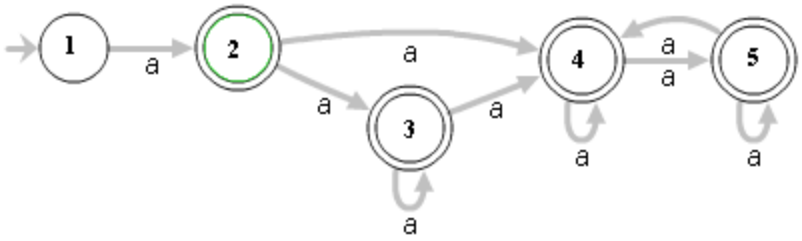
- **D**enial **o**f **S**ervice
  - Brute force **D**enial **o**f **S**ervice
    - **D**istributed **D**enial **o**f **S**ervice
  - Sophisticated **D**enial **o**f **S**ervice
    - **R**egular **e**xpression **D**enial **o**f **S**ervice

Regex (Regular Expressions)

- Provide a flexible means for identifying strings
- Written in a formal language interpreted by a Regex engine
- Regexes are widely used
  - Text editors
  - Parsers/Interpreters/Compilers
  - Search engines
  - Text validations
  - Pattern matchers…

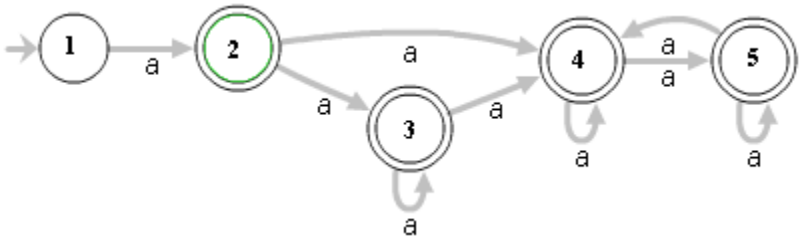
Regex naïve algorithm

- Build Nondeterministic Finite Automata (NFA)
- Transition until end of input
- Several “next” states
- Deterministic algorithms to get to all states



Regex naïve algorithm - complexity

- Might be exponential
- Example
  - Regex: ^(a+)+$
  - Payload: aaaaX



  - 2^4=16 different paths
- What about aaaaaaaaaaaaaaaaX?

Notice

- Not all algorithms are naïve
- Pure Regex algorithms are NOT exponential
  - Only Regexes with back-reference should be difficult to be “solved” efficiently:
    - Back-reference example: **([a-c])x\1x\1**
      - Will match axaxa, bxbxb, cxcxc
      - Will not match axbxa
    - http://www.regular-expressions.info/brackets.html
- Still, most existing implementations use exponential algorithms, **for all Regexes**

Regex can be evil…

- Regex is “evil” if it can stuck on crafted input
- Evil Regex pattern contains:
  - Grouping with repetition
  - Inside the repeated group:
    - Repetition
    - Alternation with overlapping

Evil patterns examples

- (a+)+
- ([a-zA-Z]+)*
- (a|aa)+
- (a|a?)+
- (.*a){x} | for x > 10

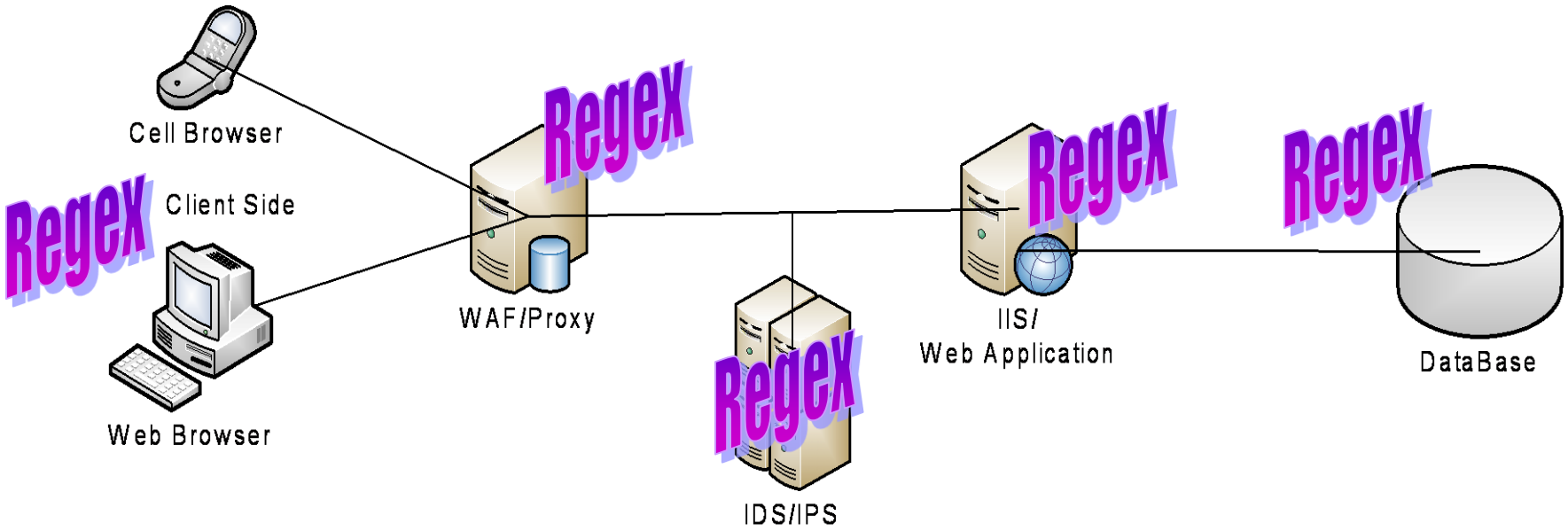
Payload* – “aaaaaaaaaaaaaaaaaaaaaaaaa!”

[Any more ideas for evil patterns?]

**Notice that the payload length depends on the pattern and the system used*

Why is it a threat?

- The Web is Regex-Based:



- In this presentation we will discuss ReDoS attacks on:
  - Web application
  - Client-side

ReDoS - Real examples 1

- Regex Library (http://regexlib.com/)
  - Multiple Email address validation (id 749)
    - Regex: ^[a-zA-Z]+(([\'\,\.\-][a-zA-Z])? [a-zA-Z]*)*\s+<(\w[-._\w]*\w@\w[-._\w]*\w\.\w{2,3})>$|^(\w[-._\w]*\w@\w[-._\w]*\w\.\w{2,3})$
    - Payload: aaaaaaaaaaaaaaaaaaaaaaa!
  - Email Validator (id 1755)
    - Regex: ^([a-zA-Z0-9]+)([\._-]?[a-zA-Z0-9]+)*@([a-zA-Z0-9]+)([\._-]?[a-zA-Z0-9]+)*([\.]{1}[a-zA-Z0-9]{2,})+$
    - Payload: a@aaaaaaaaaaaaaaaaaaaaaaaaaaaaa!

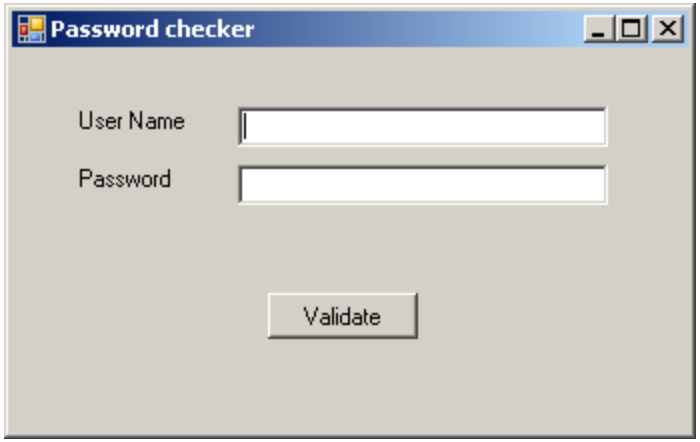
ReDoS - Real examples 2

- OWASP Validation Regex Repository
  - Person Name
    - Regex: ^[a-zA-Z]+(([\'\,\.\-][a-zA-Z])? [a-zA-Z]*)*$
    - Payload: aaaaaaaaaaaaaaaaaa!
  - Java Classname
    - Regex: ^(([a-z])+.)+[A-Z]([a-z])+$
    - Payload: aaaaaaaaaaaaaaaaaa!

Attack

- Two ways to ReDoS a system:
  - Crafting a special input for an existing Regex
    - Regex: (a+)+b
    - Payload: aaaaaaaaaaaaaaaaaaaaaaaaX
  - Regex Injection if the system builds the Regex dynamically, then uses it on some “problematic” input
    - Input : aaaaaaaaaaaaaaaaaaaaaaaaaaa
    - Payload : (a+)+X

Demonstration 1



Web Applications

Web application– Regex validations

- Regular expressions validation rules
- Two main strategies:
  - Accept known good
    - Begin with “^” and end with “$”
    - Not too tight (otherwise False Positives DoS for users)
  - Reject known bad
    - Identify an attack fingerprint
    - Too relaxed Regex => False Negatives

Web application – malicious inputs

- Crafting malicious input for a given Regex
- Blind attack
  - Try to understand which Regex can be used
  - Try to divide Regex into groups
  - For each group try to find an unmatched string
- Not blind attack
  - Open source
  - Client side Regex:
    - Understand a given Regex and build a malicious input

Demonstration 2

http://10.31.0.74/bookstore

Web application – Attack

- Application ReDoS attack vector
  - Open a JavaScript
  - Find evil Regex
  - Craft a malicious input for a found Regex
  - Submit a valid value via intercepting proxy
  - Change the request to contain a malicious input
  - You are done!

Need source code? – "Ask Google"

- All in Google: http://www.google.com/codesearch
- We can use operators and Meta-Regex
  - Regex.+\(\.*\)\+
  - Regex.+\(.\.*\)*
- **Google CodeSearch Hacking** – using meta-Regexes to find evil Regexes in open sources

Web application ReDoS Examples

- DataVault:
  - Regex: ^\[(,.*)*\]$
  - Payload: [,,,,,,,,,,,,,,,,,,,,,,,,,,,,
- WinFormsAdvansed:
  - Regex: \A([A-Z,a-z]*\s?[0-9]*[A-Z,a-z]*)*\Z
  - Payload: aaaaaaaaaaaaaaaaa!
- EntLib
  - Regex: ^([^\"]+)(?:\\([^\"]+))*$
  - Payload: \\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\“
- http://www.us-cert.gov/cas/bulletins/SB09-271.html

Client side

Client-side ReDoS – really?

- Internet browsers usually prevent DoS
- Between issues that browsers prevent:
  - Infinite loops
  - Long iterative statements
  - Endless recursions
- But what about Regex?

* In your free time you can have a look at http://github.com/EnDe/ReDoS/ to test your browser…

Client-side ReDoS – where?

- New multiple vendor Web Browsers
  - Java/JavaScript based browsers
- Cellular devices with a browsing ability
  - DoS on a cellular device is a serious attack
- Other devices – the future is so “promising”…

Client-side ReDoS – so easy!

- Browsers ReDoS attack vector:
  - Deploy a page with the following JavaScript code:

```
<html>
  <script language='jscript'>
    myregexp = new RegExp(/^(a+)+$/);
    mymatch = myregexp.exec("aaaaaaaaaaaaaaaaaaaaaaaab");
  </script>
</html>
```

  - Trick a victim to browse this page
  - You are done!

Demonstration 3

```
<html>
  <script language='jscript'>
    myregexp = new RegExp(/^(a+)+$/);
    mymatch = myregexp.exec("aaaaaaaaaaaaaaaaaaaaaaaab");
  </script>
</html>
```

Countermeasures

- No Regex-source is safe – always check for ReDoS prior to using a Regex
- Dynamic Regexes are dangerous – Regexes should generally not be user input-based
- Client validation can reveal your secrets – remember, the client side code is visible to all
- Beware WAF, IDS, Proxy – all can be easily ReDoS-ed if wrongly configured

ReDoS testing tools

- Proposed tools for Regex safety testing:
  - **Dynamic Regex testing, pen testing/fuzzing**
    - http://confoo.ca/en/2010/session/le-fuzzing-et-les-tests-d-intrusions
  - **Static Regex code analyzer**
    - Soon…

ReDoS and dynamic tools

- Prevention vector 1
  - Try to penetrate the system with different inputs
  - Check response time
  - If it increases- repeat characters
  - If a response time get slow – you are ReDoS-ed!
- Prevention vector 2
  - Try to inject an invalid escape sequence like “\m”
  - If a response is different from a response on a valid input – you are ReDoS-ed!

ReDoS and static code analysis

- Prevention vector 3
  - Analyze the source code and look for Regex
  - Check each Regex
    - Does it contain evil patterns?
    - Can it be data-influenced by a user?
  - If it does/can – you are ReDoS-ed!

Conclusion – Regexes might be evil…

- The web is Regex-based.
- The border between safe and unsafe Regex is very ambiguous.
- Regex worst (exponential) case may be easily leveraged to DoS attacks on the web.

Thank you!

Questions?

Adar Weidman <adarw@checkmarx.com>

CHECKMARX
SOURCE CODE ANALYSIS TECHNOLOGIES

References - Books

- **A. V. Aho**, 1991: Algorithms for finding patterns in strings, in Handbook of theoretical computer science (vol. A): algorithms and complexity, Pages: 255 – 300.
- **Jeffery E.F. Friedl**, 2006: Mastering Regular Expressions (Third Edition), O'Reilly Media, Inc.

References – Links (1)

- http://www.owasp.org/index.php/OWASP_Validation_Regex_Repository
- http://regexlib.com/
- http://www.cs.rice.edu/~scrosby/hash/slides/USENIX-RegexpWIP.2.ppt
- http://www.regular-expressions.info/brackets.html
- http://www.regular-expressions.info/catastrophic.html
- http://swtch.com/~rsc/regexp/regexp1.html
- http://www.usenix.org/event/woot08/tech/full_papers/drewry/drewry_html/
- http://hauser-wenz.de/playground/papers/RegExInjection.pdf
- http://www.google.com/codesearch
- http://github.com/EnDe/ReDoS/
- http://www.checkmarx.com/NewsDetails.aspx?id=23&cat=3

References – Links (2)

- Code examples:

  http://www.us-cert.gov/cas/bulletins/SB09-271.html

  - http://www.google.com/codesearch/p?hl=en&sa=N&cd=3&ct=rc#4QmZNJ8GGhI/trunk/DataVault.Tesla/Impl/TypeSystem/AssociationHelper.cs
  - http://www.google.com/codesearch/p?hl=en&sa=N&cd=1&ct=rc#nVoRdQ_MJpE/Zoran/WinFormsAdvansed/RegeularDataToXML/Form1.cs
  - http://www.google.com/codesearch/p?hl=en&sa=N&cd=4&ct=rc#Y_Z6zi1FBas/Blocks/Common/Src/Configuration/Manageability/Adm/AdmContentBuilder.cs
- Fuzzer:
  - http://www.mail-archive.com/w3af-develop@lists.sourceforge.net/msg00657.html
  - http://confoo.ca/en/2010/session/le-fuzzing-et-les-tests-d-intrusions