RegexScalpel: Regular Expression Denial of Service (ReDoS) Defense by Localize-and-Fix

Yeting Li, **Yecheng Sun**, Zhiwu Xu, Jialun Cao, Yuekang Li, Rongchen Li,
Haiming Chen, Shing-Chi Cheung, Yang Liu, Yang Xiao

Institute of Information Engineering, Chinese Academy of Sciences
Institute of Software, Chinese Academy of Sciences
University of Chinese Academy of Sciences
Shenzhen University
The Hong Kong University of Science and Technology
Nanyang Technological University

Regular Expression Denial-of-Service

The Server Side

Validate the input email using a vulnerable regex:

`^[a-zA-Z0-9._]+@([a-zA-Z0-9]+.)+com$`

Enter your email:

The Client Side

Normal input :
abc@def.com

Respond immediately

Malicious input :
c@cccccccc...

Denial-of-Service

Regular Expression Denial-of-Service Defense

- Existing Solutions
  - **Regex engine substitution**

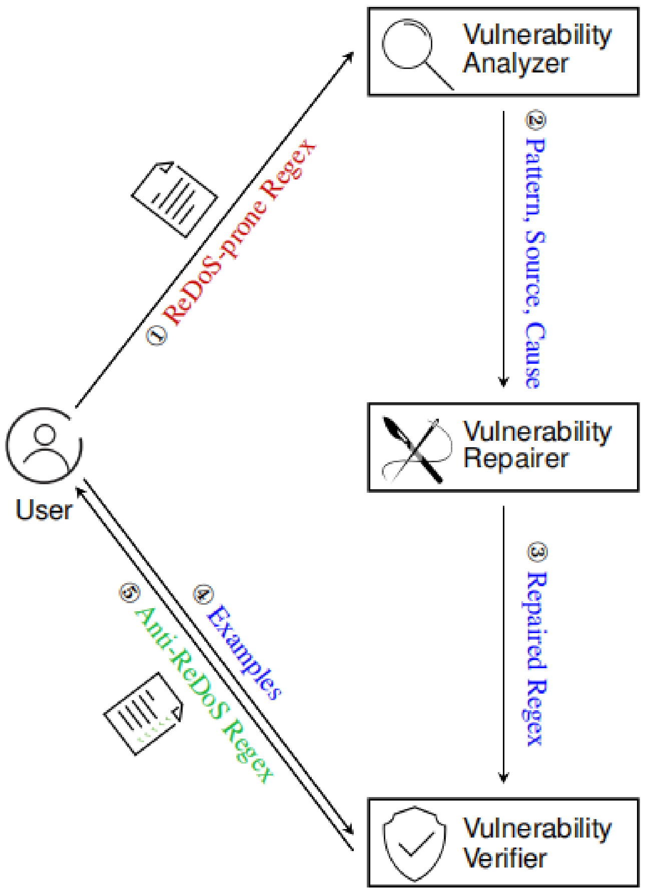
    Omitting extended features, consuming space, bringing semantic differences or incompatibilities.
  - **Input length restriction**

    Facing a dilemma known as "Goldilocks problem".
  - **Regex repair**
    - Repairing vulnerable regexes can greatly mitigate their vulnerabilities.
    - But it's challenging.

Table 13: The proportion of Maintainers' Defense Actions.

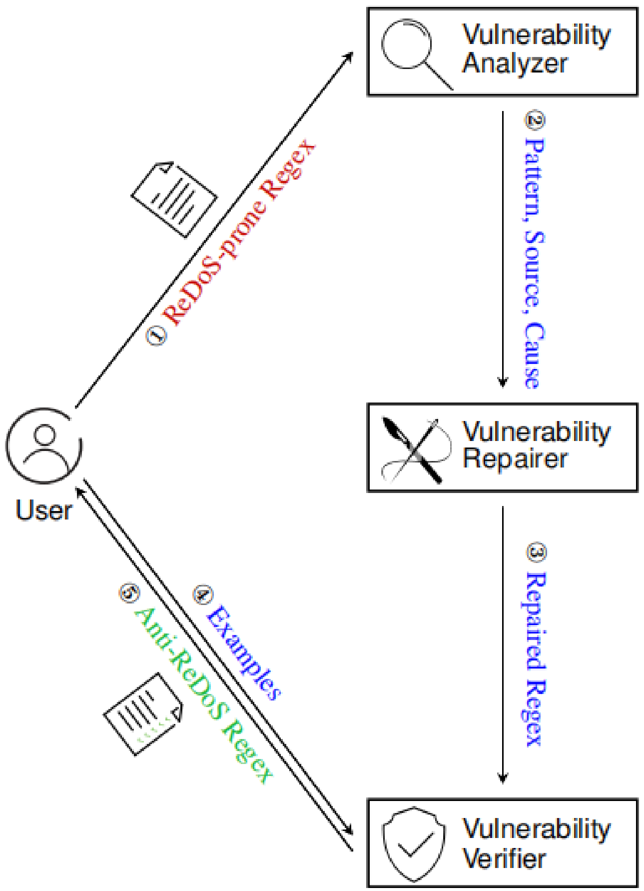
Defense Strategy	SOLA-DA	CVE	Total
Regex Engine Substitution	0 (0%)	1 (0.24%)	1 (0.22%)
Input Length Limit	1 (2.94%)	2 (0.48%)	3 (0.67%)
Code Logic Modification	6 (17.65%)	17 (4.11%)	23 (5.13%)
Regex Repair	**21 (61.76%)**	**392 (94.69%)**	**413 (92.19%)**
No Fix	6 (17.65%)	2 (0.48%)	8 (1.79%)
#Regex	34	414	448

Regular Expression Denial-of-Service Defense



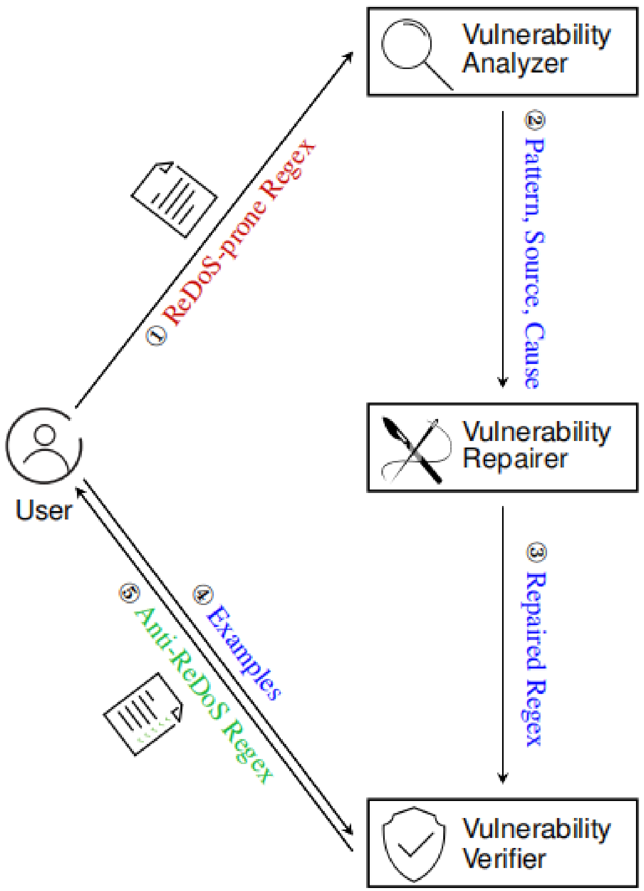
- Our Solution
  - We proposed RegexScalpel, a regex ReDoS vulnerability analysis and repair framework based on localize-and-fix.
  - RegexScalpel can preserve the semantics of the original regex, and the iterative repair method also keeps out vulnerabilities of the repaired regexes.

Regular Expression Denial-of-Service Defense



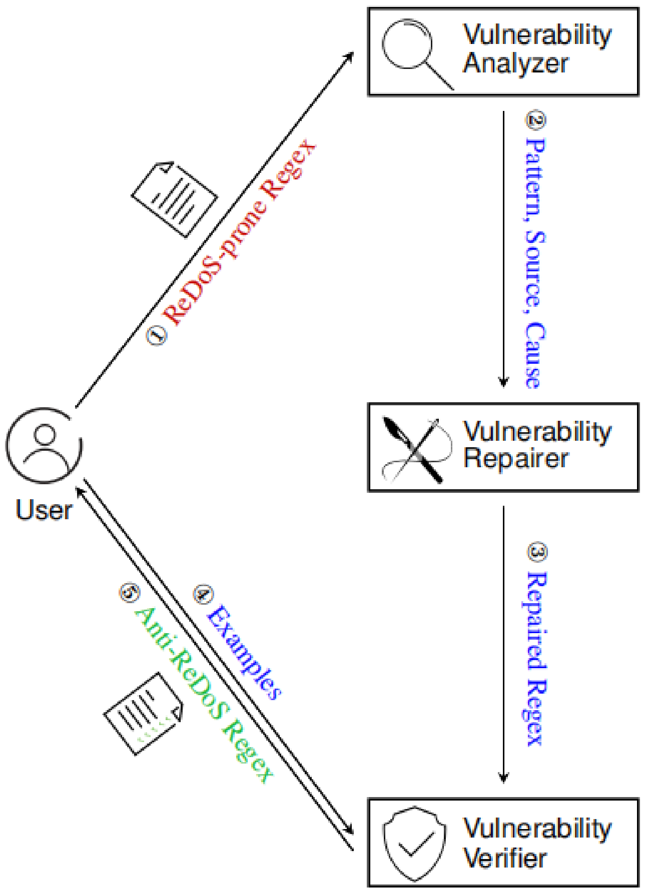
- Our Solution
  - RegexScalpel first leverages the **fine-grained vulnerability patterns** to localize the vulnerabilities and obtain the information for the repair.
  - The information includes their **vulnerable patterns**, **the source** (i.e., the pathological sub-regexes), and **the root causes** (e.g., the overlapping sub-regexes).

Regular Expression Denial-of-Service Defense



- Our Solution
  - RegexScalpel then fixes the pathological sub-regexes according to the **repair patterns** and the information.
  - The repair patterns use **micro-manipulations** (e.g., adding a lookaround, deleting a quantifier or sub-regex, modifying a quantifier or sub-regex) to remove the overlapping paths or reducing the maximum times of backtracking.

Regular Expression Denial-of-Service Defense



- Our Solution
  - RegexScalpel next determines whether the repaired regexes are ReDoS-invulnerable and whether it can pass all the given test cases.
  - If so, the repaired regex is called a successful one. If not, RegexScalpel continues the vulnerability analysis and repairs it.
  - RegexScalpel finally returns a repaired regex randomly chosen from the successful ones.

Vulnerable Patterns

Table 1: The Sub-pattern, Vulnerability Description, Example Regex, and Results from AnaNQ of the Pattern $\mathcal{NQ}$.

<table>
<tr><th>No.</th><th>Sub-pattern</th><th>Vulnerability Description</th><th>Example Regex</th><th>Returned Triple</th></tr>
<tr><td>#1</td><td>$\mathcal{N}Q_1$</td><td>$r = r_1\{m,n\}$, where $r_1 = r_p\{m_p,n_p\}$, $n_p > 1$, and $n > 1$</td><td>$\delta_1 =$ \[(\d+;)?(\d+)*m (CVE-2015-9239)</td><td>($\mathcal{N}Q_1$, (\d+)*, [+, *])</td></tr>
<tr><td>#2</td><td>$\mathcal{N}Q_2$</td><td>$r = (r_0r_1r_2)\{m,n\}$, where r_0 and r_2 are nullable, $r_1 = r_p\{m_p,n_p\}$, $n_p > 1$, and $n > 1$</td><td>$\delta_2 =$ ^(https?|ftp):\/\/(-\.)?([^\s\/?\.#]+\.?)+(\/[^\s]*)?[^\s\.,]$ (CVE-2021-26272)</td><td>($\mathcal{N}Q_2$, ([^\s\/?\.#]+\.?)+, [+, +])</td></tr>
<tr><td>#3</td><td>$\mathcal{N}Q_3$</td><td>$r = (r_0r_1r_2)\{m,n\}$, where r_0 and r_2 are nullable, $r_1 = (\ldots \mid r_p\{m_p,n_p\} \mid \ldots)$, $n_p > 1$, and $n > 1$</td><td>$\delta_3 =$ D[oD]?(\[[^\[\]]*\]|\s+)+MMMM? (moment)</td><td>($\mathcal{N}Q_3$, (\[[^\[\]]*\]|\s+)+, [+, +])</td></tr>
</table>

- Nested Quantifiers (NQ)
  - ➢ The NQ pattern is a regex with nested quantifiers.
  - ➢ In order to facilitate fixing the pathological regex, we subdivided NQ pattern into three sub-patterns (i.e., NQ1, NQ2 and NQ3).

Vulnerable Patterns

Table 2: The Sub-pattern, Vulnerability Description, Example Regex, and Results from AnaQOD of the Pattern $\mathcal{QOD}$.

<table>
<tr><th>No.</th><th>Sub-pattern</th><th>Vulnerability Description</th><th>Example Regex</th><th>Returned Triple</th></tr>
<tr><td>#1</td><td>$\mathcal{QOD}_1$</td><td>$r = \alpha\{m,n\}$, $\alpha = (r_1 \mid \ldots \mid r_k)$, and $\exists\ \mathcal{L}(\alpha_1) \cap \mathcal{L}(\alpha_2) \neq \emptyset$, where $\alpha_1 = r_{p_1} r_{p_2} \ldots r_{p_t}$, $\alpha_2 = r_{q_1} r_{q_2} \ldots r_{q_s}$, $1 \leq t, s \leq k$, $1 \leq i \leq t$, $1 \leq j \leq s$, $1 \leq p_i, q_j \leq k$, and $p_1 \neq q_1$</td><td>δ4 = <code>^(\d\.|\.\d|\d)+$</code> ([14])</td><td>($\mathcal{QOD}_1$, <code>(\d\.|\.\d|\d)+</code>, <code>[\d\.\d, \d\.\d]</code>)</td></tr>
<tr><td>#2</td><td>$\mathcal{QOD}_2$</td><td>$r = \alpha\{m,n\}$, $\alpha = (r_1 \mid \ldots \mid r_k)$, and $\exists\ \mathcal{L}(r_p) \cap \mathcal{L}(\alpha_1) \neq \emptyset$, $\mathcal{L}(r_l) \cap \mathcal{L}(r_p) = \emptyset$, where $\alpha_1 = r_{q_1} r_{q_2} \ldots r_{q_s}$, $1 \leq i \leq k$, $r_i = r_l r_p$, r_l is not nullable, $1 \leq s \leq k$, $1 \leq j \leq s$, $1 \leq q_j \leq k$</td><td>δ5=<code>(?:[\w-]|\$[-\w]+|#\{\$[-\w]+\})+(?=\s*:)</code> (CVE-2021-23341)</td><td>($\mathcal{QOD}_2$, <code>(?:[\w-]|\$[-\w]+|#\{\$[-\w]+\})+</code>, <code>[[\w-], \$[-\w]+]</code>)</td></tr>
</table>

- Quantified Overlapping Disjunction (QOD)
  - The QOD pattern is a quantified disjunction whose multiple inner sub-regexes overlap.
  - In order to facilitate fixing the pathological regex, we subdivided QOD pattern into two sub-patterns (i.e., QOD1 and QOD2).

Vulnerable Patterns

Table 3: The Sub-pattern, Vulnerability Description, Example Regex, and Results from AnaQOA of the Pattern $\mathcal{QOA}$.

<table>
<tr><th>No.</th><th>Sub-pattern</th><th>Vulnerability Description</th><th>Example Regex</th><th>Returned Triple</th></tr>
<tr><td>#1</td><td>$\mathcal{QOA}_1$</td><td>$r = (\ldots r_1 r_2 \ldots)\{m,n\}$, where $r_1 = r_p\{m_p,n_p\}$, $r_2 = r_q\{m_q,n_q\}$, $\mathcal{L}(r_1) \cap \mathcal{L}(r_2) \neq \emptyset$</td><td>$\delta_6$ = ^__[^\W_]+\w+__$ (pylint)</td><td>($\mathcal{QOA}_1$, [^\W_]+\w+, [[^\W_]+, \w+])</td></tr>
<tr><td>#2</td><td>$\mathcal{QOA}_2$</td><td>$r = (\ldots r_1 r_2 r_3 \ldots)\{m,n\}$, where $r_2 = r_t\{m_t,n_t\}$ is nullable, $r_1 = r_p\{m_p,n_p\}$, $r_3 = r_q\{m_q,n_q\}$, $\mathcal{L}(r_1) \cap \mathcal{L}(r_3) \neq \emptyset$</td><td>$\delta_7$ = ^(>=?|<=?)\s*(\d*\.?\d+)%$ (CVE-2021-23364)</td><td>($\mathcal{QOA}_2$, \d*\.?\d+, [\d*, \d+])</td></tr>
<tr><td>#3</td><td>$\mathcal{QOA}_3$</td><td>$r = (r_1 \ldots r_2)\{m,n\}$, where $r_1 = r_p\{m_p,n_p\}$, $r_2 = r_q\{m_q,n_q\}$, $n > 1$, $\mathcal{L}(r_1) \cap \mathcal{L}(r_2) \neq \emptyset$</td><td>$\delta_8$ = @([\w\-]+\.[\w\-:]+)+[:/]</td><td>($\mathcal{QOA}_3$, ([\w\-]+\.[\w\-:]+)+, [[\w\-]+, [\w\-:]+])</td></tr>
<tr><td>#4</td><td>$\mathcal{QOA}_4$</td><td>$r = (r_1 \ldots r_2 r_3)\{m,n\}$, where $r_3 = r_t\{m_t,n_t\}$ is nullable, $r_1 = r_p\{m_p,n_p\}$, $r_2 = r_q\{m_q,n_q\}$, $n > 1$, $\mathcal{L}(r_1) \cap \mathcal{L}(r_2) \neq \emptyset$</td><td>$\delta_9$ = ^([ab]+d[ac]+e?)+$</td><td>($\mathcal{QOA}_4$, ([ab]+d[ac]+e?)+, [[ab]+, [ac]+])</td></tr>
<tr><td>#5</td><td>$\mathcal{QOA}_5$</td><td>$r = (r_1 r_2 \ldots r_3)\{m,n\}$, where $r_1 = r_t\{m_t,n_t\}$ is nullable, $r_2 = r_p\{m_p,n_p\}$, $r_3 = r_q\{m_q,n_q\}$, $n > 1$, $\mathcal{L}(r_2) \cap \mathcal{L}(r_3) \neq \emptyset$</td><td>$\delta_{10}$ = ^(;(([\t]*[0-9a-zA-Z]+=[\x21-\x7E]*)*)[\t]*)?$</td><td>($\mathcal{QOA}_5$, ([\t]*[0-9a-zA-Z]+=[\x21-\x7E]*)*, [[0-9a-zA-Z]+, [\x21-\x7E]*])</td></tr>
</table>

- Quantified Overlapping Adjacent (QOA)
  - The QOA pattern is a quantified regex containing two adjacent overlapping sub-regexes.
  - In order to facilitate fixing the pathological regex, we subdivided QOA pattern into five sub-patterns (i.e., QOA1, QOA2, QOA3, QOA4, and QOA5).

Vulnerable Patterns

Table 4: The Sub-pattern, Vulnerability Description, Example Regex, and Results from AnaSLQ of the Pattern $\mathcal{SLQ}$.

No.	Sub-pattern	Vulnerability Description	Example Regex	Returned Triple
#1	$\mathcal{SLQ}_1$	Starting with $r = r_1$, where $r_1 = r_q\{m_q, n_q\}$, and $n_q > 1$	$\delta_{11} =$ `([A-Z]+)([A-Z][a-z])` (CVE-2021-3820)	($\mathcal{SLQ}_1$, `[A-Z]+`, `[A-Z]+`)
#2	$\mathcal{SLQ}_2$	Starting with $r = r_1 r_2$, where $r_1 = r_p\{m_p, n_p\}$ is nullable, $r_2 = r_q\{m_q, n_q\}$, and $n_q > 1$	$\delta_{12} =$ `\$?[A-Z]+` (PrismJS)	($\mathcal{SLQ}_2$, `\$?[A-Z]+`, `[A-Z]+`)
#3	$\mathcal{SLQ}_3$	Starting with $r = r_1 r_2$, where $r_1 = r_p\{m_p, n_p\}$ is not nullable, $r_2 = r_q\{m_q, n_q\}$, and $\mathcal{L}(r_1) \cap \mathcal{L}(r_2) \neq \emptyset$	$\delta_{13} =$ `{([\s\S]+?)}` (CVE-2021-3777)	($\mathcal{SLQ}_3$, `{([\s\S]+?)`, [{, `[\s\S]+`])
#4	$\mathcal{SLQ}_4$	Starting with $r = r_1 r_2$, where $r_1 = r_p\{m_p, n_p\}$ is not nullable, $r_2 = r_q\{m_q, n_q\}$, $r_q = r_{q_1} r_{q_2}$, r_{q_1} is not nullable, $r_{q_2} = r_t\{m_t, n_t\}$, and $\mathcal{L}(r_{q_2}) \cap \mathcal{L}(r_1) \neq \emptyset$	$\delta_{14} =$ `[ab](ca+)+d`	($\mathcal{SLQ}_4$, `[ab](ca+)+`, [`[ab]`, `a+`])
#5	$\mathcal{SLQ}_5$	Starting with $r = r_1 r_2$, where $r_1 = r_p\{m_p, n_p\}$ is not nullable, $r_2 = r_q\{m_q, n_q\}$, $r_q = r_{q_1} r_{q_2} r_{q_3}$, r_{q_1} and r_{q_3} are not nullable, $r_{q_2} = r_t\{m_t, n_t\}$, and $\mathcal{L}(r_{q_2}) \cap \mathcal{L}(r_1) \neq \emptyset$	$\delta_{15} =$ `[ab](ca{1,2}da)+e`	($\mathcal{SLQ}_5$, `[ab](ca{1,2}da)+`, [`[ab]`, `a{1,2}`])

- Starting with Large Quantifier (SLQ)
  - ➢ The SLQ pattern is a regex starting with a sub-regex with a large quantifier.
  - ➢ In order to facilitate fixing the pathological regex, we subdivided SLQ pattern into three sub-patterns (i.e., SLQ1, SLQ2, SLQ3, SLQ4, and SLQ5).

Repair Patterns

Table 5: The Repair Patterns for Nested Quantifiers ($\mathcal{NQ}$).

No.	Repair Pattern	
τ_1	$\dfrac{\mathcal{L}(r) = \mathcal{L}(r_p\{m_p \times m, n_p \times n\})}{r \Longrightarrow (r_p\{m_p \times m, n_p \times n\})\cancel{\{m,n\}}}$	$(\mathcal{N}Q_1)$
τ_2	$\dfrac{\mathcal{L}(r) = \mathcal{L}((r_0 r_p \cancel{\{m_p,n_p\}} r_2)\{m,n\})}{r \Longrightarrow (r_0 r_p \cancel{\{m_p,n_p\}} r_2)\{m,n\}}$	$(\mathcal{N}Q_2)$
τ_3	$\dfrac{\mathcal{L}(r) = \mathcal{L}((r_0(\ldots \mid r_p\cancel{\{m_p,n_p\}} \mid \ldots)r_2)\{m,n\})}{r_1 \Longrightarrow (\ldots \mid r_p\cancel{\{m_p,n_p\}} \mid \ldots)}$	$(\mathcal{N}Q_3)$

- Nested Quantifiers (NQ)
  - The NQ pattern has a redundant quantifier by nature. So to fix NQ pattern, we can remove the redundant quantifier.

Repair Patterns

Table 6: The Repair Patterns for Quantified Overlapping Disjunction ($\mathcal{QOD}$).

No.	Repair Pattern	No.	Repair Pattern	No.	Repair Pattern
τ_4	$\dfrac{\mathcal{L}(r)=\mathcal{L}((r_1\|\ldots\|r_q\|\ldots\|r_k)\{m,n\})}{\alpha \implies (r_1\|\ldots\|r_q\|\ldots\|r_k)}\ (\mathcal{QOD}_1)$	τ_5	$\dfrac{t>1}{r_{p_1} \implies r_{p_1}(?!\Phi(r_{p_2}))}\ (\mathcal{QOD}_1)$	τ_6	$\dfrac{s>1}{r_{q_1} \implies r_{q_1}(?!\Phi(r_{q_2}))}\ (\mathcal{QOD}_{\{1,2\}})$
τ_7	$\dfrac{}{r_{p_1} \implies (?!\Phi(\alpha_2))r_{p_1}}\ (\mathcal{QOD}_1)$	τ_8	$\dfrac{}{r_{q_1} \implies (?!\Phi(\alpha_1))r_{q_1}}\ (\mathcal{QOD}_1)$	τ_9	$\dfrac{\mathrm{scs}(\alpha_1)=true,\quad \Sigma_{\alpha_1}\setminus \mathrm{first}(r_{q_1})\neq\emptyset}{\alpha_1 \implies \Theta(\Sigma_{\alpha_1}\setminus \mathrm{first}(r_{q_1}))}\ (\mathcal{QOD}_1)$
τ_{10}	$\dfrac{\mathrm{scs}(\alpha_2)=true,\quad \Sigma_{\alpha_2}\setminus \mathrm{first}(r_{p_1})\neq\emptyset}{\alpha_2 \implies \Theta(\Sigma_{\alpha_2}\setminus \mathrm{first}(r_{p_1}))}\ (\mathcal{QOD}_1)$	τ_{11}	$\dfrac{}{r_p \implies r_p(?<!\Phi(\alpha_1))}\ (\mathcal{QOD}_2)$	τ_{12}	$\dfrac{}{r_{q_1} \implies (?!\Phi(r_p))r_{q_1}}\ (\mathcal{QOD}_2)$
τ_{13}	$\dfrac{\mathrm{scs}(\alpha_1)=true,\quad \Sigma_{\alpha_1}\setminus \mathrm{first}(r_p)\neq\emptyset}{\alpha_1 \implies \Theta(\Sigma_{r_{q_1}}\setminus \mathrm{first}(r_p))}\ (\mathcal{QOD}_2)$	τ_{14}	$\dfrac{r_p=r_u\{m_u,n_u\},\quad \mathrm{scs}(r_u)=true,\quad \Sigma_{r_u}\setminus \mathrm{first}(r_{q_1})\neq\emptyset}{r_p \implies \Theta(\Sigma_{r_u}\setminus \mathrm{first}(r_{q_1}))}\ (\mathcal{QOD}_2)$		

- Quantified Overlapping Disjunction (QOD)
  - ➢ The QOD pattern has multiple matching paths across the overlapping disjunctions for a string.
  - ➢ We proposed three strategies, namely, deleting one overlapping disjunction, adding a lookaround constraint to one overlapping disjunction, and modifying one overlapping disjunction by subtracting the first set of the other one.

Repair Patterns

Table 7: The Repair Patterns for Quantified Overlapping Adjacency ($\mathcal{QOA}$).

No.	Repair Pattern	No.	Repair Pattern	No.	Repair Pattern
τ_{15}	$\frac{\mathcal{L}(r_p)=\mathcal{L}(r_q)}{r_1 r_2 \implies r_p\{m_p+m_q, n_p+n_q\}}\ (\mathcal{QOA}_1)$	τ_{16}	$\frac{}{r_1 \implies r_1(?<!\Phi(r_q))}\ (\mathcal{QOA}_1)$	τ_{17}	$\frac{}{r_2 \implies (?!\Phi(r_p))r_2}\ (\mathcal{QOA}_1)$
τ_{18}	$\frac{n \leq 1}{n_p \implies n_\mu, n_q \implies n_\mu}\ (\mathcal{QOA}_{\{1,2\}})$	τ_{19}	$\frac{\mathrm{scs}(r_p)=true,\quad \Sigma_{r_p} \setminus \mathrm{first}(r_q) \neq \emptyset}{r_p \implies \Theta(\Sigma_{r_p} \setminus \mathrm{first}(r_q))}\ (\mathcal{QOA}_{\{1,3\}})$	τ_{20}	$\frac{\mathrm{scs}(r_q)=true,\quad \Sigma_{r_q} \setminus \mathrm{first}(r_p) \neq \emptyset}{r_q \implies \Theta(\Sigma_{r_q} \setminus \mathrm{first}(r_p))}\ (\mathcal{QOA}_{\{1,3\}})$
τ_{21}	$\frac{\mathrm{scs}(r_p)=true,\quad \Sigma_{r_p} \setminus \mathrm{first}(r_q) \neq \emptyset,\quad m_p \geq 1}{r_1 \implies r_p\{m_p-1, n_p-1\}\Theta(\Sigma_{r_p} \setminus \mathrm{first}(r_q))}\ (\mathcal{QOA}_1)$	τ_{22}	$\frac{\mathrm{scs}(r_q)=true,\quad \Sigma_{r_q} \setminus \mathrm{first}(r_p) \neq \emptyset,\quad m_q \geq 1}{r_2 \implies \Theta(\Sigma_{r_q} \setminus \mathrm{first}(r_p))r_q\{m_q-1, n_q-1\}}\ (\mathcal{QOA}_1)$	τ_{23}	$\frac{}{r_1 \implies (?!\Phi(r_q))r_1}\ (\mathcal{QOA}_3)$
τ_{24}	$\frac{}{r_2 \implies r_2(?<!\Phi(r_p))}\ (\mathcal{QOA}_3)$	τ_{25}	$\frac{\mathrm{scs}(r_p)=true,\quad \Sigma_{r_p} \setminus \mathrm{first}(r_q) \neq \emptyset,\quad m_p \geq 1}{r_1 \implies \Theta(\Sigma_{r_p} \setminus \mathrm{first}(r_q))r_p\{m_p-1, n_p-1\}}\ (\mathcal{QOA}_3)$	τ_{26}	$\frac{\mathrm{scs}(r_q)=true,\quad \Sigma_{r_q} \setminus \mathrm{first}(r_p) \neq \emptyset,\quad m_q \geq 1}{r_2 \implies r_q\{m_q-1, n_q-1\}\Theta(\Sigma_{r_q} \setminus \mathrm{first}(r_p))}\ (\mathcal{QOA}_3)$
τ_{27}	$\frac{m_t=0}{r \implies r_1 r_3 \mid r_1 r_t\{1,n_t\} r_3}\ (\mathcal{QOA}_2)$	τ_{28}	$\frac{m_t=0}{r_2 r_3 \implies r_2 \mid r_2 r_t\{1,n_t\}}\ (\mathcal{QOA}_4)$	τ_{29}	$\frac{m_t=0}{r_1 r_2 \implies r_2 \mid r_t\{1,n_t\} r_2}\ (\mathcal{QOA}_5)$

- Quantified Overlapping Adjacent (QOA)
  - The QOA pattern contains the corresponding two overlapping adjacencies.
  - We proposed three repair strategies, that is, merging the overlapping adjacencies, adding a lookaround constraint to one overlapping adjacency, and modifying one overlapping adjacency.

Repair Patterns

Table 8: The Repair Patterns for Starting with Large Quantifier ($\mathcal{SLQ}$).

No.	Repair Pattern	No.	Repair Pattern	No.	Repair Pattern
τ_{30}	$\dfrac{}{r \Longrightarrow \hat{}r}\ (\mathcal{SLQ}_{\{1,2,3,4,5\}})$	τ_{31}	$\dfrac{}{n_q \Longrightarrow n_\mu}\ (\mathcal{SLQ}_{\{1,2,3,4,5\}})$	τ_{32}	$\dfrac{m_p = 0}{r \Longrightarrow r_2 \mid r_p\{1,n_p\}r_2}\ (\mathcal{SLQ}_2)$
τ_{33}	$\dfrac{}{r_p \Longrightarrow (?!\Phi(r_q))r_p}\ (\mathcal{SLQ}_3)$	τ_{34}	$\dfrac{}{r_q \Longrightarrow (?!\Phi(r_p))r_q}\ (\mathcal{SLQ}_3)$	τ_{35}	$\dfrac{\mathrm{scs}(r_p) = true,\quad \Sigma_{r_p} \setminus \mathrm{first}(r_q) \neq \emptyset}{r_p \Longrightarrow \Theta(\Sigma_{r_p} \setminus \mathrm{first}(r_q))}\ (\mathcal{SLQ}_3)$
τ_{36}	$\dfrac{\mathrm{scs}(r_q) = true,\quad \Sigma_{r_q} \setminus \mathrm{first}(r_p) \neq \emptyset}{r_q \Longrightarrow \Theta(\Sigma_{r_q} \setminus \mathrm{first}(r_p))}\ (\mathcal{SLQ}_3)$	τ_{37}	$\dfrac{}{r_p \Longrightarrow r_p(?<!\Phi(r_t))}\ (\mathcal{SLQ}_{\{4,5\}})$	τ_{48}	$\dfrac{}{r_t \Longrightarrow (?!\Phi(r_p))r_t}\ (\mathcal{SLQ}_{\{4,5\}})$
τ_{39}	$\dfrac{\mathrm{scs}(r_p) = true,\quad \Sigma_{r_p} \setminus \mathrm{first}(r_t) \neq \emptyset}{r_p \Longrightarrow \Theta(\Sigma_{r_p} \setminus \mathrm{first}(r_t))}\ (\mathcal{SLQ}_{\{4,5\}})$	τ_{40}	$\dfrac{\mathrm{scs}(r_t) = true,\quad \Sigma_{r_t} \setminus \mathrm{first}(r_p) \neq \emptyset}{r_t \Longrightarrow \Theta(\Sigma_{r_t} \setminus \mathrm{first}(r_p))}\ (\mathcal{SLQ}_{\{4,5\}})$		

- Starting with Large Quantifier (SLQ)
  - ➢ The SLQ pattern contains the sub-regex starting with a large quantifier (for SLQ1 and SLQ2) or the overlapping sub-regexes (for SLQ3, SLQ4 and SLQ5).
  - ➢ We proposed four strategies, namely, adding a start-of-line anchor, replacing the large quantifier with a small one, adding a lookaround to one sub-regex, and modifying one sub-regex by subtracting the first set of the other one.

Case Exhibition

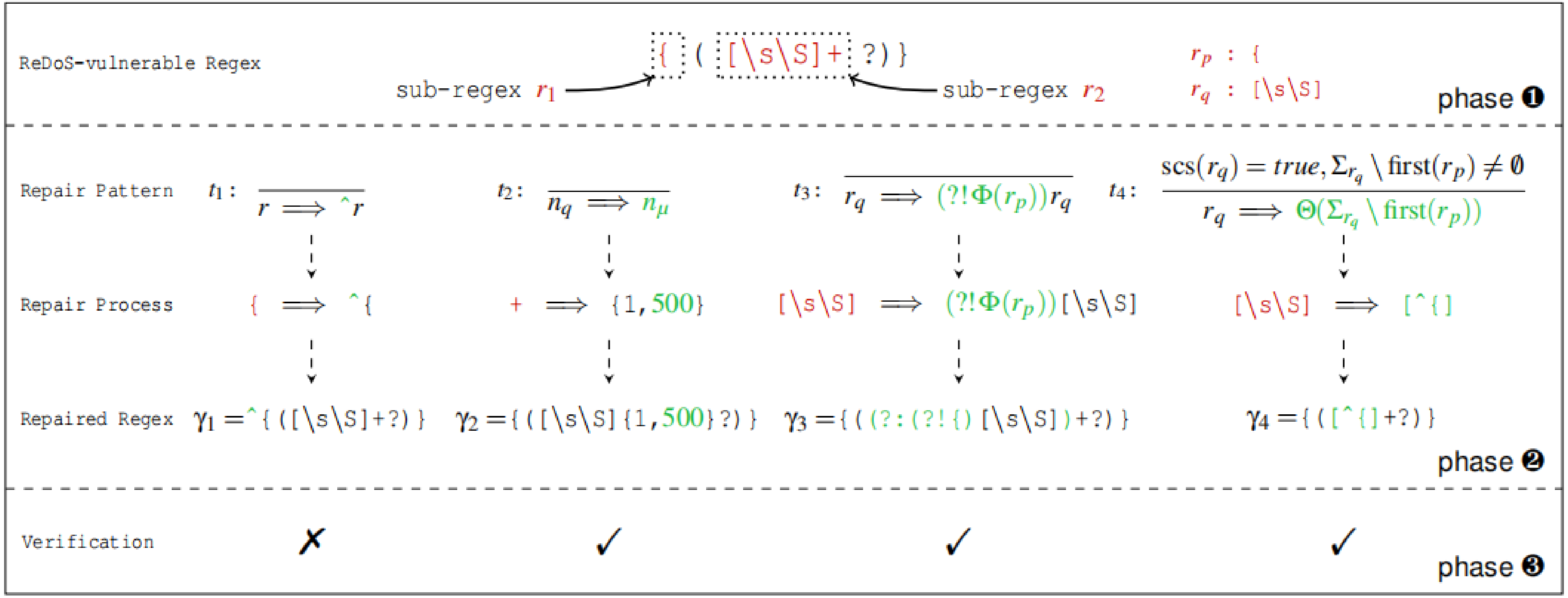


Figure 2: An Example for Repairing the ReDoS-vulnerable Regex {([\s\S]+?)}.

- The NPM package nodejs-tmpl (6,858,130 weekly downloads) used this regex, which aims to match the simple string formatting using {}.

Experiment Setup

- Evaluation Datasets
  - ➢ Our evaluation was conducted on the ReDoS-vulnerable regexes collected from two widely-used sources: (i) the SOLA-DA benchmark and (ii) real-world CVEs.

Table 9: The ReDoS-vulnerable Regex Sets for Evaluation.

Benchmark	#Regex	Description
SOLA-DA [34]	34	ReDoS-vulnerable regexes in NPM modules found by the Software Lab at TU Darmstadt
CVE [9]	414	ReDoS-vulnerable regexes extracted from 70 ReDoS-related CVEs in recent three years
Total	448	

Experiment Setup

- Evaluation Approaches
  - We selected three state-of-the-art tools belonging to three paradigms, i.e., regex engine substitution (RE2), input length restriction (LLI), and regex repair (FlashRegex).

- Evaluation Metrics
  - A defense is considered successful if it (i) passes all the given test cases, and (ii) is free from ReDoS attack.
  - The success defense rate is calculated by dividing the number of successful defenses by the total number of vulnerable regexes under defense.

Comparing State-of-the-art tools

Table 10: Success Defense Rate Across Automated Tools.

Tool	SOLA-DA	CVE	Total
RE2	18 (52.94%)	35 (8.45%)	53 (11.83%)
LLI(100)	22 (64.71%)	45 (10.87%)	67 (14.96%)
LLI(500)	26 (76.47%)	18 (4.35%)	44 (9.82%)
LLI(5000)	26 (76.47%)	19 (4.59%)	45 (10.04%)
FlashRegex	4 (11.76%)	91 (21.98%)	95 (21.20%)
RegexScalpel	**33 (97.06%)**	**410 (99.03%)**	**443 (98.88%)**
#Regex	34	414	448

Regex engine substitution

Input length restriction

Input length restriction

Our method

- RegexScalpel can effectively defend 98.88% of vulnerable regexes, compared with 21.20% achieved by the best work.

Comparing Maintainers' Repairs

Table 14: Success Defense Rate of the Repairs by Maintainers and RegexScalpel.

	SOLA-DA	CVE	Total
Manual Repair	14 (66.67%)	305 (77.81%)	319 (77.23%)
RegexScalpel	**21 (100%)**	**388 (98.98%)**	**409 (99.03%)**
#Regex	21	392	413

Input length restriction

Our method

- Among the repaired vulnerable regexes handcrafted by the maintainers, only 77.23% are ReDoS free. RegexScalpel outperforms manual fixing, and successfully repairs 99.03% of regexes.

Usefulness to Maintainers

Table 17: Demographics of New Vulnerabilities Repaired by RegexScalpel.

No.	Project	Disclosure Date	CVE ID	#Vuln. Regex
#1	Python	Jan 30th, 2021	CVE-2021-3733	1
#2	NLTK	Sep 5th, 2020	–	1
#3	pylint	Sep 3rd, 2020	–	2
#4	mpmath	Oct 8th, 2021	CVE-2021-29063	1
#5	browserslist	Apr 28th, 2021	CVE-2021-23364	6
#6	code-server	Sep 17th, 2021	CVE-2021-3810	1
#7	ansi-regex	Sep 12th, 2021	CVE-2021-3807	1
#8	nth-check	Sep 17th, 2021	CVE-2021-3803	1
#9	nodejs-tmpl	Sep 15, 2021	CVE-2021-3777	1
#10	jspdf	Feb 12th, 2021	CVE-2021-23353	1
			Total	16

- RegexScalpel detected and repaired 16 new ReDoS regexes in ten popular projects.
- All the 16 repairs were accepted by the maintainers and merged into subsequent project releases, resulting in 8 confirmed CVEs.

Semantics Preservation

- We used the following equation to calculate the semantic similarities between the repaired regexes and the original ones.

$$Sim(r_1, r_2) = \frac{|\mathcal{L}(r_1) \cap \mathcal{L}(r_2)|}{|\mathcal{L}(r_1) \cup \mathcal{L}(r_2)|}$$

$$Sim(r_1, r_2) = \lim_{\lambda \to +\infty} \frac{|\mathcal{L}(r_1)^{\leq \lambda} \cap \mathcal{L}(r_2)^{\leq \lambda}|}{|\mathcal{L}(r_1)^{\leq \lambda} \cup \mathcal{L}(r_2)^{\leq \lambda}|}$$

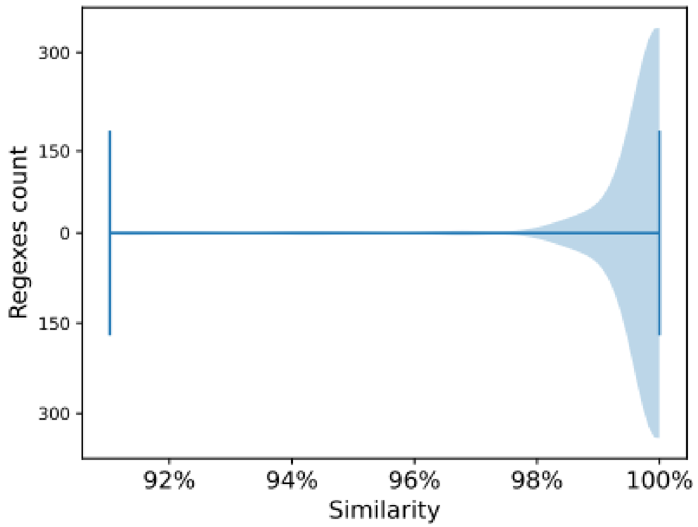


Figure 3: Semantic similarities between the vulnerable regexes and the repaired ones.

- For the benchmarks, most similarities go beyond 98%. On average, the semantic similarity is 99.57%, meaning that the semantics of regexes are well-preserved after the repair.

Summary

- We proposed RegexScalpel, which can defend ReDoS attacks by automatically localizing and repairing vulnerable regexes.
- The evaluation exhibits the remarkable effectiveness of RegexScalpel. It achieves 98.88% successful repair ratio, compared with 21.20% achieved by the best existing work.
- RegexScalpel helped to repair 16 ReDoS vulnerabilities in the ten real-world projects and got confirmed by the maintainers, resulting in 8 confirmed CVEs.
- RegexScalpel can synthesize repaired regexes preserving the semantics of the original ones and keeping the semantics as close as possible to the original ones.

THANKS

Q&A

Presenter: Yecheng Sun

E-mail: sunyc@ios.ac.cn