Statically Typed String Sanitation Inside a Python

Nathan Fulton Cyrus Omar Jonathan Aldrich

Problem

Web applications must ultimately command systems, such as web browsers and database engines, using strings. Strings derived from improperly sanitized user input are therefore a potential vector for command injection attacks.

Milieu

To address the largest security threat facing today's web applications:

- **Developers** use libraries and frameworks which ultimately **ground out to operations on strings**.
- **Researchers** propose information flow and taint analyses, which are are often **attack-specific**, can become complicated, and do not generalize to arbitrary validation tasks (e.g., "is this a unix file path?").

Approach

We introduce **regular expression types** for classifing strings and equip these types with standard operations. Our approach makes it possible to specify and verify correctness of conventional implementations of input sanitation procedures.

References

N. Fulton, C. Omar, and J. Aldrich.
Statically typed string sanitation inside a python.
First International Workshop on Privacy and Security in Programming (PSP). ACM, 2014.

Two Illustrative Excerpts

String **concatenation** is typed using regular expression concatenation:

$$\textsc{S-T-Concat}\quad \frac{\Psi \vdash e_1 : \mathsf{stringin}[r_1] \qquad \Psi \vdash e_2 : \mathsf{stringin}[r_2]}{\Psi \vdash \mathsf{rconcat}(e_1; e_2) : \mathsf{stringin}[r_1 \cdot r_2]}$$

$$\textsc{S-E-Concat}\quad \frac{e_1 \Downarrow \mathsf{rstr}[s_1] \qquad e_2 \Downarrow \mathsf{rstr}[s_2]}{\mathsf{rconcat}(e_1; e_2) \Downarrow \mathsf{rstr}[s_1 s_2]}$$

Substring operations pattern match on the head of a string. Regular expression derivatives provide a natural approximation (ltl is roughly the derivative of lhd):

$$\textsc{S-T-Case}\quad \frac{\Psi \vdash e_1 : \mathsf{stringin}[r] \qquad \Psi \vdash e_2 : \sigma \qquad \Psi, x : \mathsf{stringin}[\mathsf{lhd}(r)], y : \mathsf{stringin}[\mathsf{ltl}(r)] \vdash e_3 : \sigma}{\Psi \vdash \mathsf{rstrcase}(e_1; e_2; x, y.e_3) : \sigma}$$

$$\textsc{S-E-Case-}\epsilon\quad \frac{e_1 \Downarrow \mathsf{rstr}[\epsilon] \qquad e_2 \Downarrow v_2}{\mathsf{rstrcase}(e_1; e_2; x, y.e_3) \Downarrow v_2}$$

$$\textsc{S-E-Case-Concat}\quad \frac{e_1 \Downarrow \mathsf{rstr}[as] \qquad [\mathsf{rstr}[a], \mathsf{rstr}[s]/x, y]e_3 \Downarrow v_3}{\mathsf{rstrcase}(e_1; e_2; x, y.e_3) \Downarrow v_3}$$

This system (λ_{RS}) also contains **replacement**, **checked casts** and **dynamically checked coercions**. See the paper for details[1].

Implementation Example

We are working toward a regular string types library for the extensible programming language Atlang.

```
@fn
def sanitize(s : stringin[r'.*']):
  return (s.replace(r'"', '&quot;')
           .replace(r'<', '&lt;')
           .replace(r'>', '&gt;'))

@fn
def query(s : stringin[r'[^"]*']):
  return 'SELECT * FROM users WHERE ' +
         'name="' + s + '"'
@fn
def div(s : stringin[r'[^<>]*']):
  return '<div>Results for '+s+'</div>'

@fn
def main():
  input = sanitize(user_input())
  results = db_execute(query(input))
  return div(input) + format(results)
```

Carnegie Mellon University

nathanfu@cs.cmu.edu · comar@cs.cmu.edu · jonathan.aldrich@cs.cmu.edu