Type-Specific Languages to Fight Injection Attacks

Darya Kurilova, Cyrus Omar, Ligia Nistor, Benjamin Chung, Alex Potanin†, and Jonathan Aldrich

Carnegie Mellon University and Victoria University of Wellington

Problem
To create a webpage displaying results of a database query, a developer would prefer to write:

```javascript
let webpage = HTML = '<html>
  <body>
    <h1>Results for {keyword}</h1>
    <ul>
      {query_results(db, WHERE title = {keyword})}
    </ul>
  </body>
</html>
```

However, in a programming language without support for HTML and SQL literals, the developer has to write instead:

```javascript
let webpage = HTML = '<html>
  <body>
    <h1>Results for " + keyword + "</h1>
    <ul>
      query_results(db, WHERE title = " + keyword + ")
    </ul>
  </body>
</html>
```

To make it simpler, the developer ends up writing the following unsafe code!

```javascript
let webpage = HTML = parse_html(
  '<html>
  <body>
    <h1>Results for " + keyword + "</h1>
    <ul>
      query_results(db, WHERE title = " + keyword + "">
    </ul>
  </body>
</html>
```

Example Exploits

Cross-Site Scripting (XSS) – #3 in OWASP Top 10
- provide ""<script> malicious script here </script>" as keyword in line 4 (right above) to execute a malicious script

Injection Attack – #1 in OWASP Top 10
- provide "'; DROP TABLE products --" as keyword in line 8 (right above) to erase the products table from the database

Approach*

- Shift responsibility for parsing certain generic literal forms into the typechecker.
- Typechecker, in turn, defers responsibility to user-defined types, by treating the body of the literal as a term of the type-specific language (TSL) associated with the type it is being checked against.
- TSL rewriting the term to use only general-purpose notations and can contain expressions of the host language.
- TSLs are implemented in the Wyvern programming language.

What a Software Developer Writes

```javascript
1 let keyword : String = user_input
2 let webpage : HTML = HTML
3 <html>
4  <body>
5   <h1>Results for {keyword}</h1>
6   <ul>
7     query_results(db, WHERE title = {keyword})
8   </ul>
9 </body>
</html>
```

What a TSL Designer Writes

```javascript
1 case type HTML
2  TextNode of String Grammar TSL
3 HTML.Element of Attributes * HTML Quoted Wyvern
4 BodyElement of Attributes * HTML TSL Delimiters
5 HtmlElement of Attributes * HTML Type- specific languages
6 UMLElement of Attributes * HTML
7 ...
8 metadata = new : HasTSL
9 val parser = :
10  start ::= '<html>' child::start '</html>'
11    'HTMLHtmlElement(([], child))'
12  start ::= '<body>' child::start '</body>'
13    'HTMLBodyElement(([], child))'
14  start ::= '<h1>' child::start '</h1>'
15    'HTMLH1Element(([], child))'
16  start ::= '<ul>' child::start '</ul>'
17    'HTMLULElement(([], child))'
18  start ::= '{<i>EXP\}'
19    'let : HTML'
20 ...
```

TSL Delimiters

- "TSL code here, "inner backticks" must be doubled"
- "TSL code here, "inner single quotes" must be doubled"
- {TSL code here, [inner brackets] must be balanced}
- <TSL code here, [inner angle brackets] must be balanced>
- ~ Forward reference for a multiline TSL code that is presented as an indented block starting on the next line

Injection Attacks Are Prevented

Cross-Site Scripting (XSS)
- On line 5 (of developer-written code), if HTMLTextNode() is omitted, a compilation error occurs. However, if HTMLTextNode() is present an arbitrary keyword of type String cannot be interpolated.

Injection Attack
- On line 8 (of developer-written code), we do not need to invoke conversion function as the keyword String is prepared to be a valid SQL parameter automatically.

Contributions

1. Type-specific languages (TSLs), a mechanism for safely composing language extensions that
   - associates the logic for parsing generic literal forms and expanding them into general-purpose syntax with a type, and
   - reduces the developer’s cognitive burden.
2. Using TSLs prevents injection attacks, such as cross-site scripting (XSS) and query injection attacks.