Towards More Security in Data Exchange

Defining Unparsers with Context-Sensitive Encoders
for Context-Free Grammars

Lars Hermerschmidt, Stephan Kugelmann, Bernhard Rumpe
Software Engineering
RWTH Aachen

http://www.se-rwth.de/

About Me

Background

- Penetration Tester
- Now Software Engineering

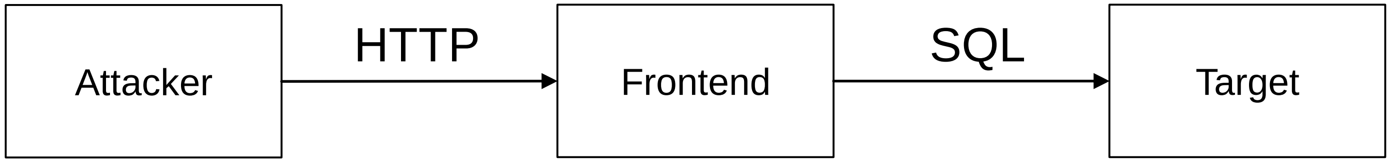
Research Focus

- Model-Driven Software Development
  - Textual Modeling Languages
- Security Architecture

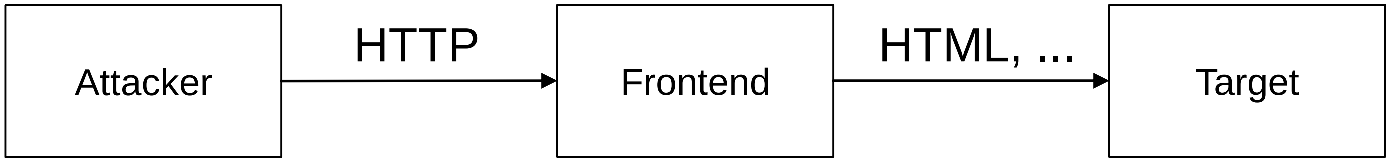
Why is Cross Site Scripting (XSS) Protection so hard to get right?

Injection Attacks

SQL Injection



XSS: plenty of different contexts where JavaScript can be used

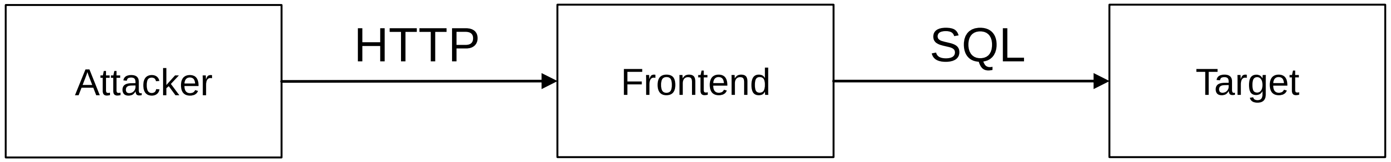


Lars Hermerschmidt
Chair of Software Engineering

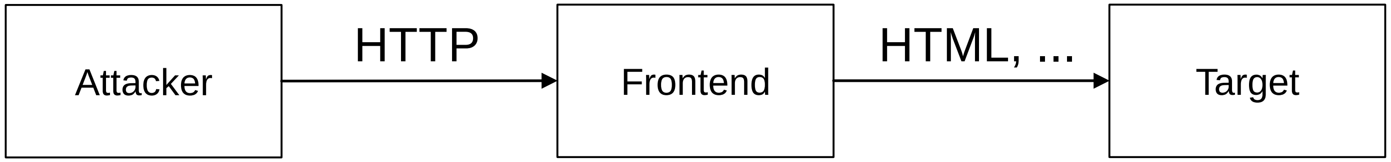
RWTH Aachen

Injection Attacks

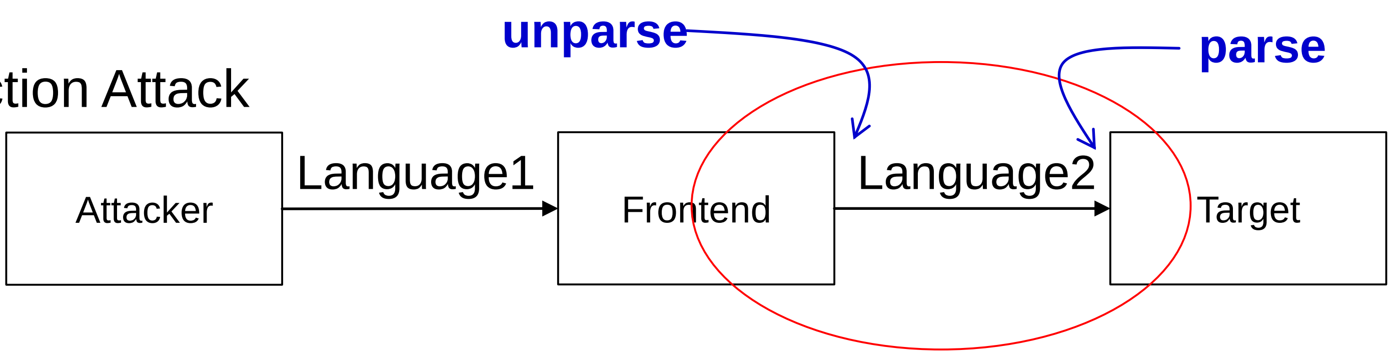
SQL Injection



XSS: plenty of different contexts where JavaScript can be used



Injection Attack



State of the art

- ✓ In general: Do not trust user data, sanitize or encode it

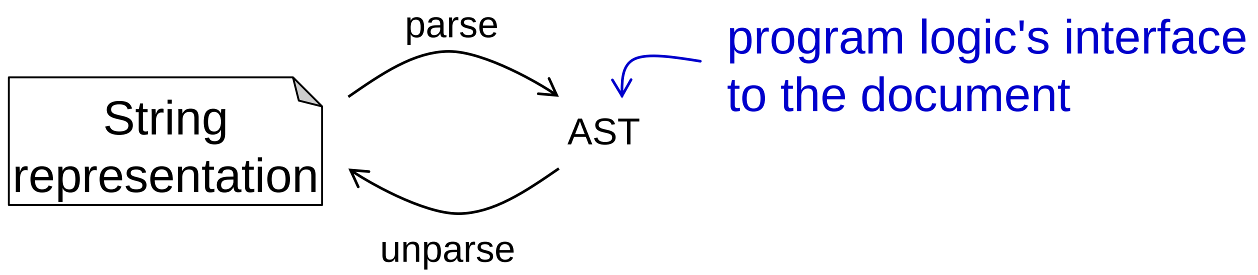
- ✓ SQL: Prepared Statements
- ✓ HTML, JavaScript, CSS
  - context aware encoding (HTML, <script>, JavaScript in HTML attribute, ...)
  - apply encoding automatically

[Weinberger2011]

- What about all the other languages?
  - Enterprise backend communication e.g. SAP systems
  - Cyber Physical Systems like cars, industrial control systems
  - new or custom formats

Lars Hermerschmidt
Chair of Software Engineering
RWTH Aachen

It happens during unparsing



Correct roundtrip

$$\forall \text{ AST } x : parse(unparse(x)) = x$$

Injection: malicious AST m containing control tokens within terminals

$$\nexists d : parse(d) = m$$

correct roundtrip for malicious AST m

$$parse_{decode}(unparse_{encode}(m)) = m$$

Defining Context-sensitive encoding

MontiCoder

- Generate (un)parser with context-sensitive (en/de)coder
- Define encoding per token in the grammar

MG

```
Element = "tags" LCURLY TagsToken RCURLY;
token LCURLY = "{";
token RCURLY = "}";
token TagsToken = (~('{' | '}' | ' '))+;

encodeTable TagsToken = {
      "{" -> "&#x0123;",
      "}" -> "&#x0125;",
      "&" -> "&#x0038;",
      " " -> "&#x0020;"
};
```

production rule

Lars Hermerschmidt
Chair of Software Engineering
RWTH Aachen

Language Composition

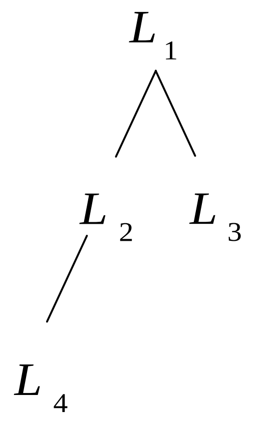
- One grammar per language (enables reuse, lowers complexity)
- Replace terminal from super-language with start symbol of sub-language
  - ➢ enables embedding of JavaScript in HTML
  - Encoding specified separately for each language

Unparsing

- Start Encoding in the most nested language
- Control characters from L_2 get encoded when used in L_4

Parsing

- Start parsing super-language
- Run decoder on tokens
- Run subparser



Reducing Language Features

Use Case: Include rich user input e.g. HTML into output

- Option 1: Reduce output language
  - Change production rules to match only tokens with special names, define encoding
  - not elegant, but more secure

- Option 2: Reduce input language
  - Copy input into output AST
  - Program logic <u>must not alter this input</u>

Using MontiCoder

Language Developer

1. Define output grammar and encoding table
2. Generate parser and unparser which include Context-Sensitive (de/en)coding

Language user a.k.a. application developer

1. Construct an AST for the output document
   a) Create parsable template
   b) Parse template to preinitialized AST
2. Add untrusted user data to AST nodes
3. Run generated MontiCoder unparser

Case Study: HTML and JavaScript

- Implemented grammars and encoding tables for HTML and JavaScript
- Web Application uses generated unparser

- Performed XSS Scan with OWASP ZAP and FuzzDB
  - found no XSS
- Manual penetration test
  - found error in one encoding table definition (<script> = <Script>)
  - added options: case-insensitive, ignore whitespaces

Conclusion

- Injection attacks arise from unparsing without encoding

- Encoding is a language property
  - defined by encoding table per grammar token

- MontiCoder: Derive context-sensitive encoder from it's definition within the grammar
  - NOT yet another HTML, JavaScript encoder

- Templates considered harmful
  - Directly putting untrusted data into output
  - Context within the output is lost
- Stop using IO APIs which have no idea of correct encoding
  - e.g. System.out.printl()

Thank You

Comments? Questions?