LangSec for AppSec folks

German OWASP Day 2025

Dr. Lars Hermerschmidt

Product Owner Security Engineering

REWE DIGITAL

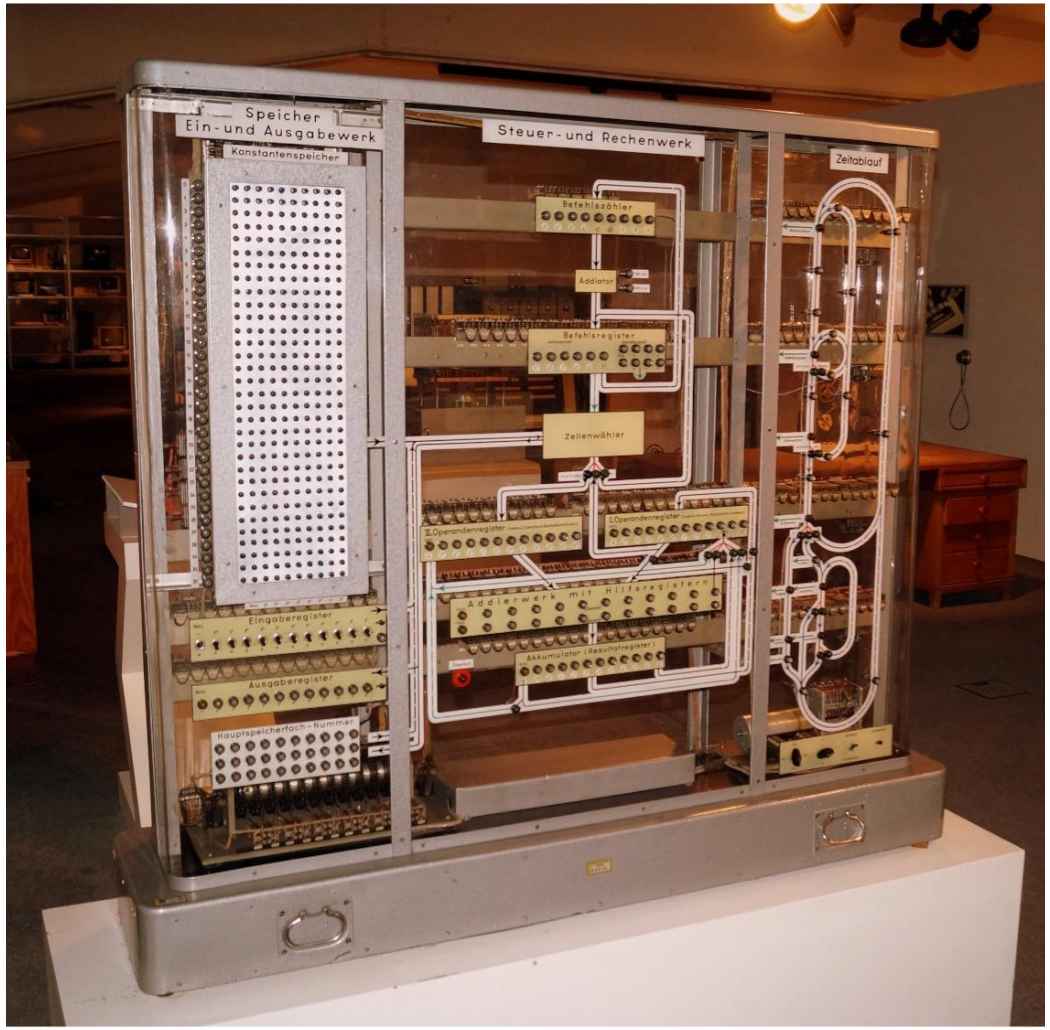
Industry

- Building Security Champions programs since 2017
- Coach and trainer for software security
- Help in DevOps transformation

Research

- LangSec, eliminating injections
- Threat Modeling automation
- Evidence based software security programs
  - security capability model Security Belts

LangSec



Software is Input

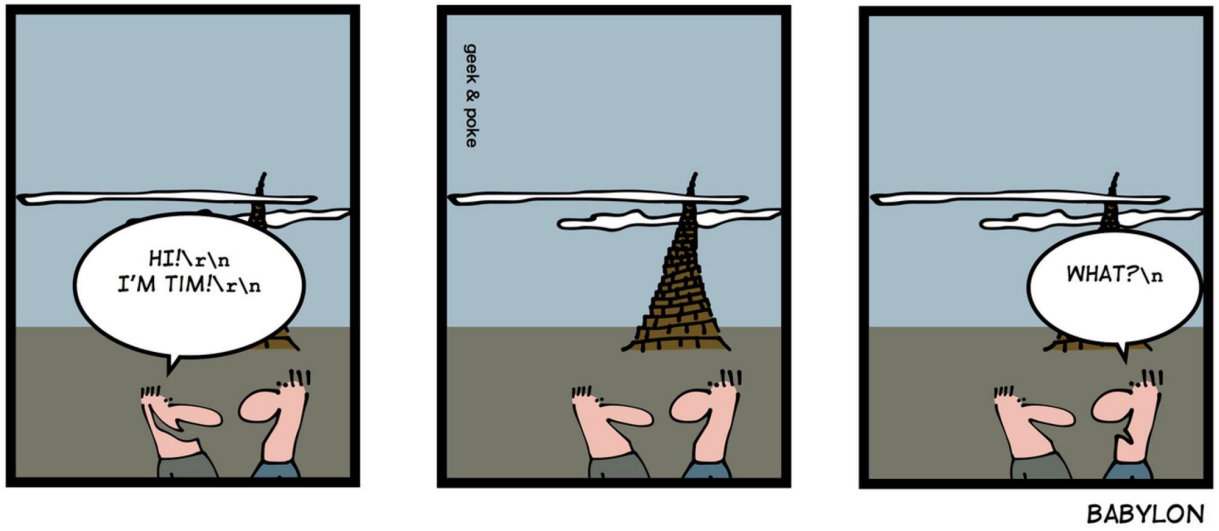
Von Neumann architecture

- General purpose machine
- Software is loaded and executed by the machine
- Software is stored like data

Input drives Software

- „Datenverarbeitung“ is an illusion
- Software reacts on input
- Attackers can use all offered processing capabilities

The View from the Tower of Babel

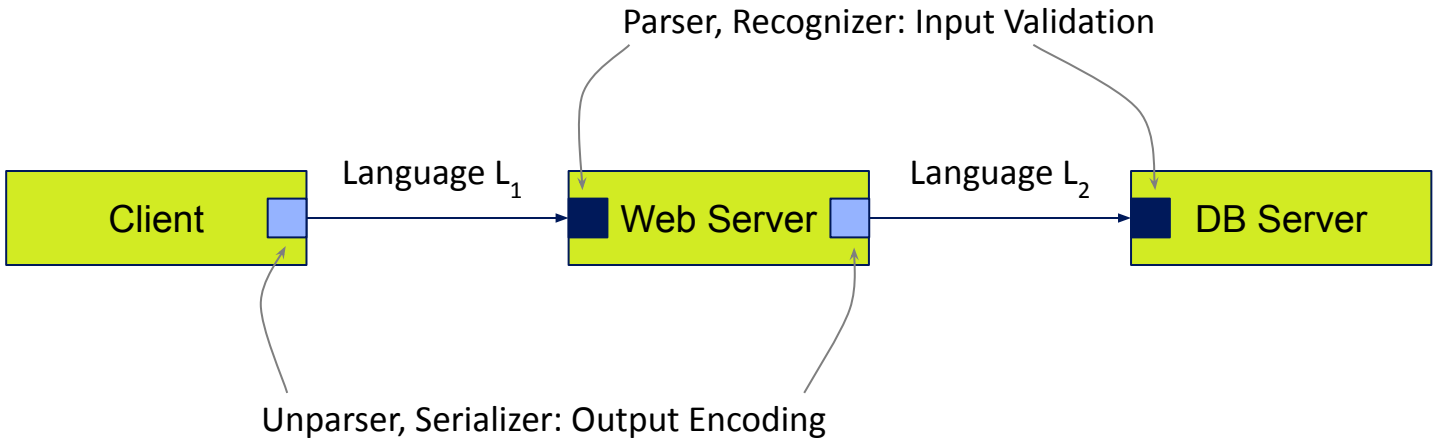


LangSec attributes the insecurity of software to ad hoc programming of input handling

- Treat all input as formal language
- Input handling code is a (un)parser for that language
- That (un)parser implementation must match the grammar used to define the language

Constructing secure (un)parsers is as complex as secure cryptography, so reuse known good libraries

Definitions - Where is that Language?



German OWASP Day 2025

01 Parser, Recognizer: Input Validation

Protection Against Input

Common Bad Advices: “Don’t trust input data”

Input Validation

- Validate all inputs against a specification
- Accept only those input fulfilling the specification
- Reject all others, NEVER try to guess/heal bad input

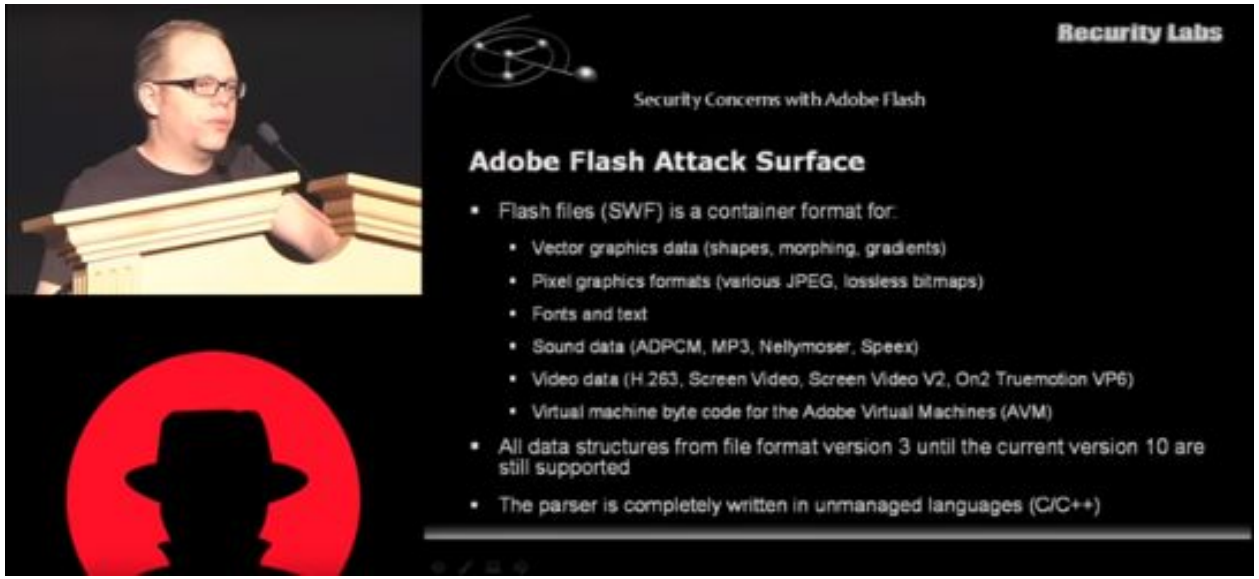
Input specification

Reject Known bad with deny list or black list: `^[^a-z]+$`

Only accept known inputs with whitelist: `^[0-9]+$`

Beware: Regular Expressions can only recognize regular languages (no nesting)

FX: Blitzableiter (2010)



FX exploited countless vulnerabilities in Flash Adobe did not manage to patch them

Blitzableiter a rigor parser used in front of the vulnerable Adobe Flash Plugin:

1. Recognize only valid Flash
2. Create a new Flash file from the accepted content
3. Pass the new Flash file to the Flash Plugin

Blitzableiter defended all exploitation attempts

FX found LangSec

The technique is now called Content Disarm and Reconstruction

German OWASP Day 2025

Antipattern: Shotgun Parser

```java
/**
 * This will parse the stream and populate the PDDocument object. This will close the k
 * done parsing.
 *
 * @param lenient activate leniency if set to true
 * @throws InvalidPasswordException If the password is incorrect.
 * @throws IOException If there is an error reading from the stream or corrupt data is
 */
public PDDocument parse(boolean lenient) throws IOException
{
    setLenient(lenient);
    // set to false if all is processed
    boolean exceptionOccurred = true;
    try
    {
        // PDFBOX-1922 read the version header and rewind
        if (!parsePDFHeader() && !parseFDFHeader())
        {
            throw new IOException( "Error: Header doesn't contain versioninfo" );
        }

        if (!initialParseDone)
        {
            initialParse();
        }
        exceptionOccurred = false;
        PDDocument pdDocument = createDocument();
        pdDocument.setEncryptionDictionary(getEncryption());
        return pdDocument;
    }
    finally
    {
        if (exceptionOccurred && document != null)
        {
            IOUtils.closeQuietly(document);
            document = null;
        }
    }
}
```

https://svn.apache.org/viewvc/pdfbox/trunk/pdfbox/src/main/java/org/apache/pdfbox/pdfparser/PDFParser.java?revision=1878544&view=co

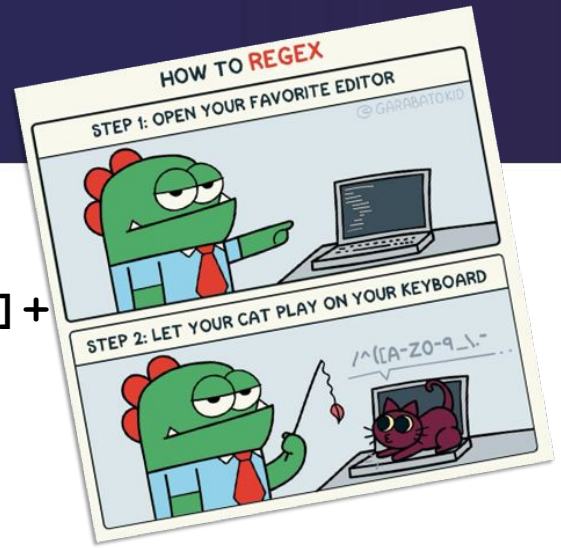
- Hand-written parser
- The recognition logic is spread all over the application
- State is allocated before the input is fully recognized

- Apache Tika uses Apache PDFBox Parser
- Hand-written parser
- Does it comply to some PDF grammar?

Don’t implement parsers in $YourFavoredProgrammingLanguage!

German OWASP Day 2025

Writing Secure Parsers is hard




```
^([a-zA-Z0-9])(([\-.]|[_]+)?([a-zA-Z0-9]+))*(@){1}[a-z0-9]+
,3})|([a-z]{2,3}[.]{1}[a-z]{2,3}))$
```

Regular Expression Catastrophic Backtracking Denial of Service

Polyglot: A files that is accepted by multiple parsers as different files

Parser Differential Bug

- Check with parser A
- Execute with parser B

github.com/Polydet/polyglot-database

README MIT license

Polyglots database

This repository is a data set of polyglot files. It's purpose is to be used to test

Please note that this repository uses https://git-lfs.github.com/.

	7z	afsk	agc	apk	avi	bmp
7ZIP+JAR.7z	x					
7ZIP+RAR-1.7z	x					
7ZIP+RAR-2.7z	x					
7ZIP+ZIP.7z	x					
AVI+HTML.avi					x	
AVI+ZIP.avi					x	
BMP+HTML+JAR.bmp						x
BMP+JAR.bmp						x
DOCX+ELF+JAR+PDF+RAR-1.zip						

github.com/corkami/mitra

README MIT license

Mitra

A tool to generate weird files (parasites, polymocks, polyglots, near-polyglots, crypto-polyglots) for many formats.

Loosely named after Μιθραδάτης, a polyglot historical figure (it's pronounced `mɪtrə`).

What's new.

How to use

- `mitra.py file1.pdf /dev/null -f --pad 2` gives you a standard PDF with an extra range of arbitrary data.
- `mitra.py file1.png file2.dcm` gives you a working PNG/DICOM polyglot.

Check Corkami mini or tiny PoCs for input files, and the formats repository for some extra technical info.

Features

It tries different layouts: Stacks (appended data), Cavities (blank space), Parasites (comments), Zippers (mutual comments).

It stores the offsets where the payloads 'switch sizes' between parenthesis for ambiguous ciphertexts.

Preventing Shotgun Parsers

Standardized language

- Find existing good (Un)Parser
- Review Parser with LangSec knowledge
- Work on Parsetree to access the data

Your own language

- Define language through grammar
- Generate (Un)Parser or use (Un)Parser Combinator
- Work on Parsetree to access the data

Don’t fear the grammar

CSV Example Grammar

```
29  grammar CSV;
30
31  csvFile: hdr row+ ;
32  hdr : row ;
33
34  row : field (',' field)* '\r'? '\n' ;
35
36  field
37      : TEXT
38      | STRING
39      |
40      ;
41
42  TEXT   : ~[,\n\r"]+ ;
43  STRING : '"' ('""'|~'"')* '"' ; // quote-quote is an escaped quote
```

S

P

Keywords (part of T)

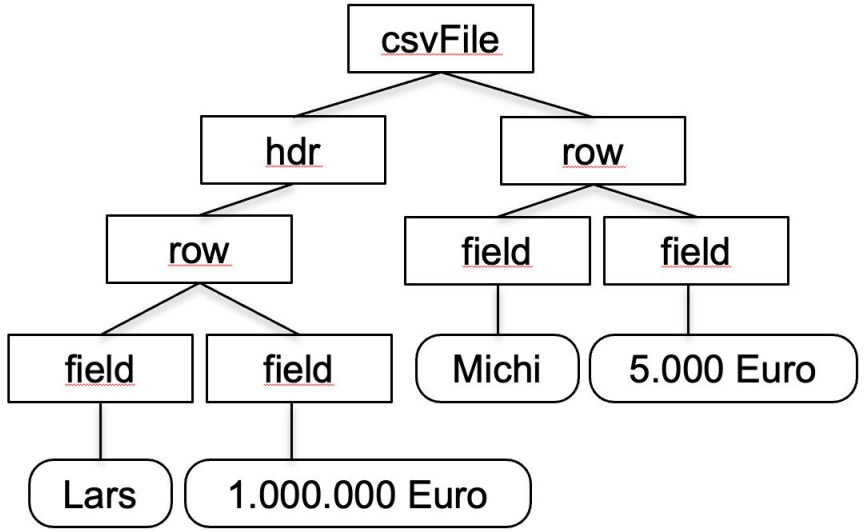
Grammar G = (N, T, P, S)
- **N**onterminal Symbols
- **T**erminal Symbols
- **P**roductions
- **S**tart Symbol

https://github.com/antlr/grammars-v4/blob/master/csv/CSV.g4

CSV Example

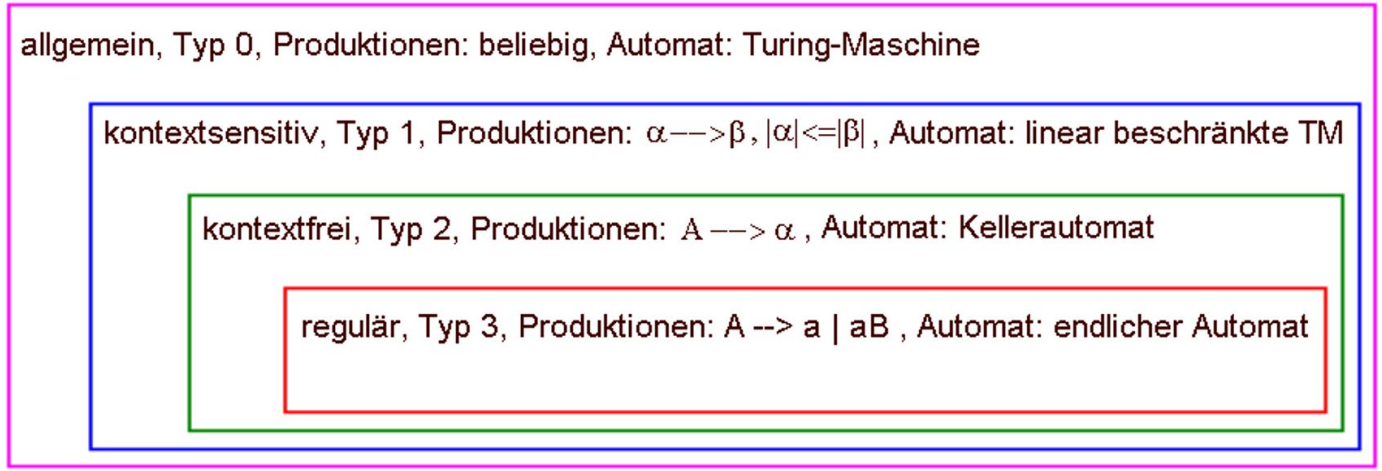
Lars,1.000.000 Euro
Michi,5.000 Euro

Parsing



Computational Cliff

Chomsky Hierarchy for Transport Languages



Context free Grammar: Halting problem decidable and computable

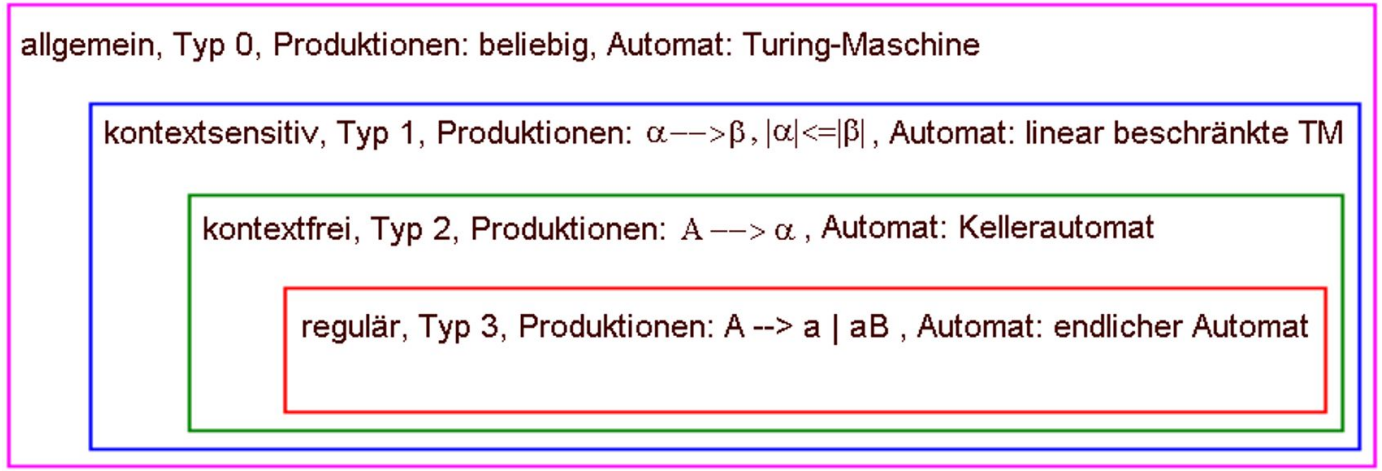
- „One can construct a parser and proof it‘s correctness.“
- Proof is challenging (ASN.1 compiler has been verified using Coq)

Above Context free Grammar = Remote Code Execution

- Parser has to be a Turing Machine
- Natural Languages are mildly context-sensitive = LLMs are doomed

Computational Cliff

Chomsky Hierarchy for Transport Languages



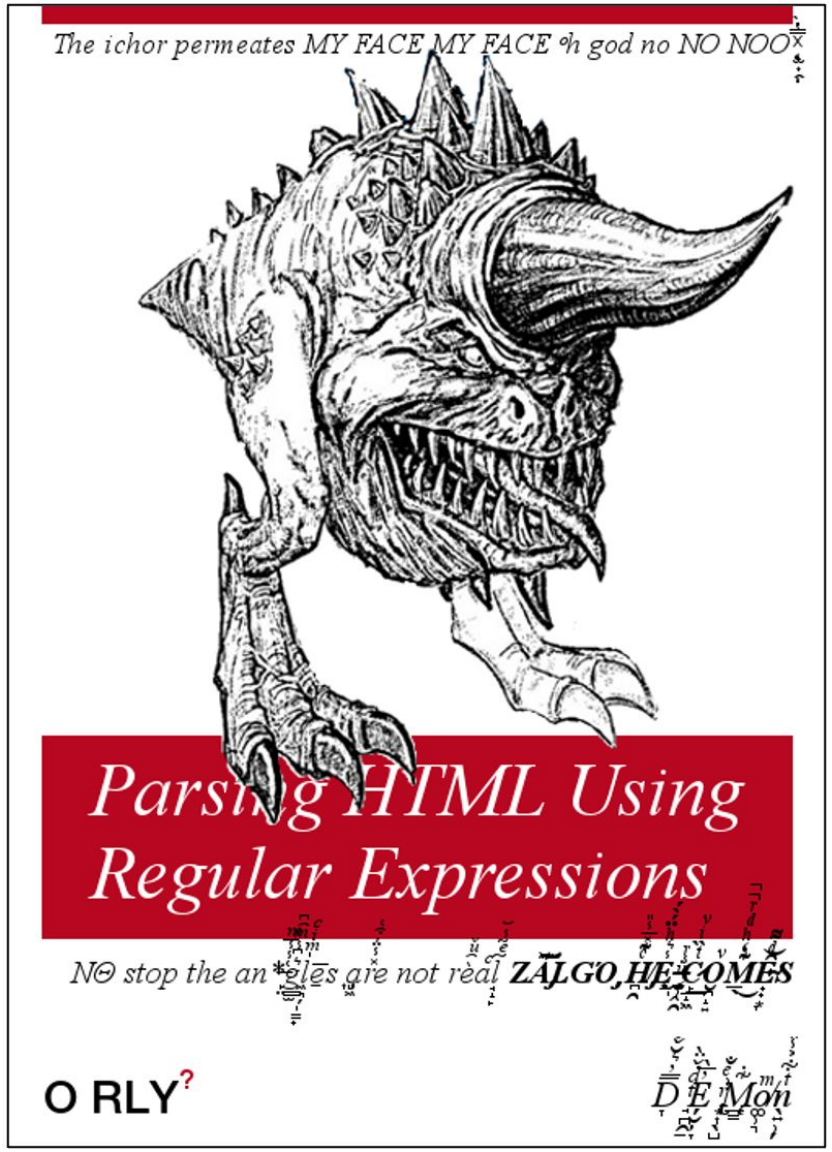
Context free Grammar: Halting problem decidable and computable

- „One can construct a parser and proof it‘s correctness.“
- Proof is challenging (ASN.1 compiler has been verified using Coq)

Above Context free Grammar = Remote Code Execution

- Parser has to be a Turing Machine
- Natural Languages are mildly context-sensitive = LLMs are doomed

Never use regular Expressions to parse more complex languages



Do not specify formats that are more complex than context free

Syntax and Semantics

Example: XML External Entity Injection (XXE)

- XML parsers load external entities from filesystem or network 🤷(language designers thought that would be a good idea)
- A Webservice processing this XML would return the /etc/passwd file to the caller

```
1  <?xml version="1.0" encoding="ISO-8859-1"?>
2  <!DOCTYPE foo [
3    <!ELEMENT foo ANY >
4    <!ENTITY xxe SYSTEM "file:///etc/passwd" >]>
5  <foo>&xxe;</foo>
```

Syntax

- Parsing enforces syntactical correctness on input
- Programs can decide which part of that input to process
  - Filtering of instructions is possible

Semantics

- Processing Input results in semantic action
  - Execute parsed instructions on Turing Machine
- (unintentionally) exposing semantic actions may lead to vulnerabilities, e.g., Log4Shell

Rice's theorem, Code Scanners

Programming Languages are <u>awesome</u>

- Turing complete
- Open to solve all problems

Programming Languages are <u>scary</u>

- Turing complete
- Unintended functionality possible

> Rice's theorem: “non-trivial, semantic properties of programs are undecidable”

It is impossible to predict what code or scripts are doing before executing them.

Implications for programming languages

- All static code analysis searching for bugs have to solve an undecidable problem and will fail.
- Dynamic analysis, i.e., security scanners is no better.

Rice's theorem, Code Scanners, and Antivirus

Programming Languages are <u>awesome</u>

- Turing complete
- Open to solve all problems

Programming Languages are <u>scary</u>

- Turing complete
- Unintended functionality possible

Rice's theorem: “non-trivial, semantic properties of programs are undecidable”

It is impossible to predict what code or scripts are doing before executing them.

Implications for programming languages

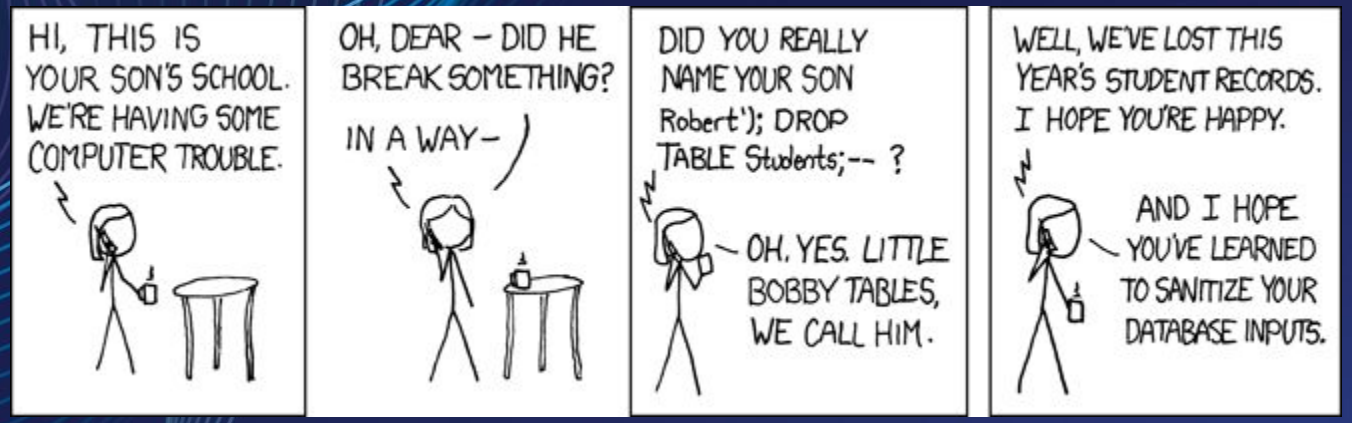
- All static code analysis searching for bugs have to solve an undecidable problem and will fail.
- Dynamic analysis, i.e., security scanners is no better.

Implications for file formats

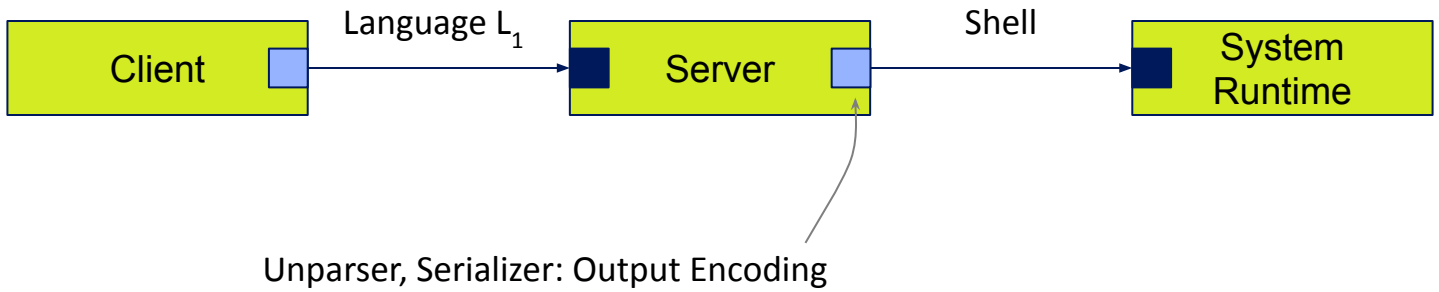
- Antivirus does not work.
- Executing in a sandbox and observing is no better.
- Solution: Remove semantic actions, i.e., scripts from file formats.

German OWASP Day 2025

02 Unparser, Serializer: Output Encoding



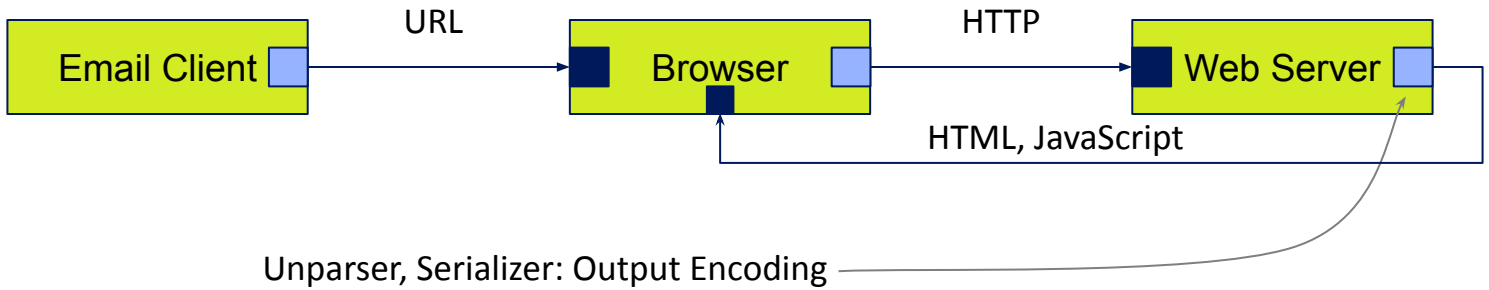
Protection Against Input during Output Creation




```
1    String btype = request.getParameter("backuptype");
2    String cmd = new String("cmd.exe /K \"c:\\util\\rmanDB.bat "
3    +btype+
4    "&&c:\\utl\\cleanup.bat\"")
5
6    System.Runtime.getRuntime().exec(cmd);
```

https://cwe.mitre.org/data/definitions/77.html

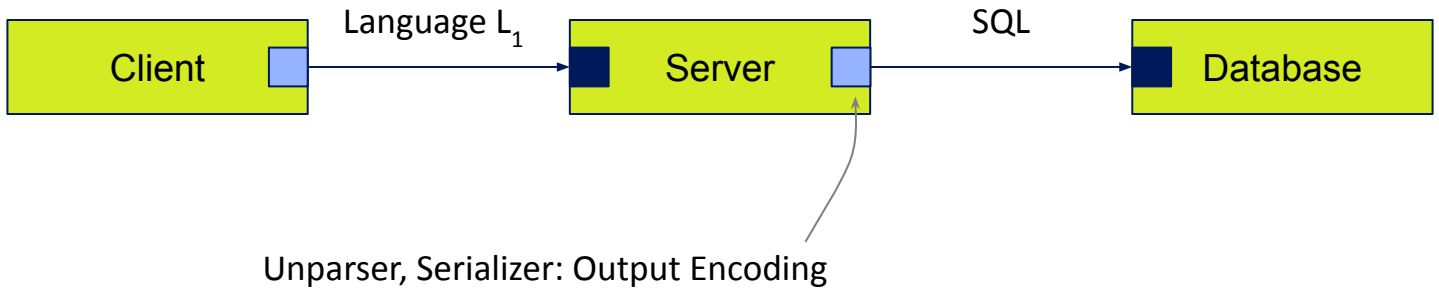
Protection Against Input during Output Creation




```
1    $username = $_GET['username'];
2    echo '<div class="header"> Welcome, ' . $username . '</div>';
```

https://cwe.mitre.org/data/definitions/79.html

Protection Against Input during Output Creation



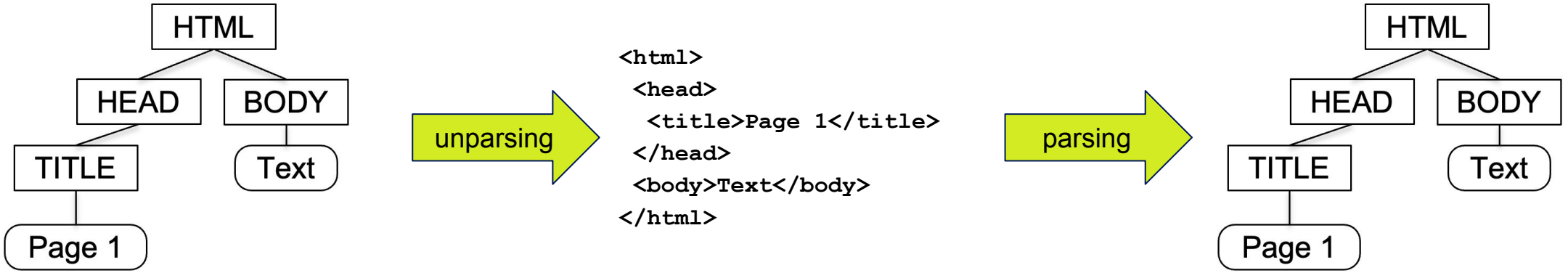

```
1   String firstname = req.getParameter("firstname");
2   String lastname = req.getParameter("lastname");
3
4   String query = "SELECT id, firstname, lastname FROM authors WHERE "
5       + "firstname = "+firstname+" and lastname = "+lastname;
6   PreparedStatement pstmt = connection.prepareStatement( query );
7   try
8   {
9       ResultSet results = pstmt.execute( );
10  }
```

https://owasp.org/www-community/attacks/SQL_Injection

String concatenation leads to

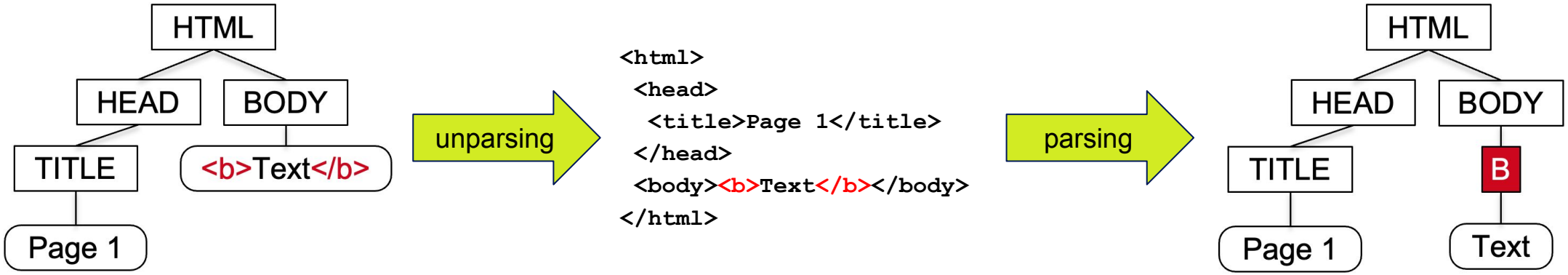
Injections

(Un)Parser Roundtrip



German OWASP Day 2025

(Un)Parser Roundtrip



Injections from a Language Perspective

“malicious input” is made of keywords from the grammar

Keywords separate data from code

Injection vulnerabilities are rooted in language design

All non-trivial languages with keywords are injectable

Languages using length field seem to be secure

- “Total Length” in IPv4 Header
- IEEE 802.3 MAC

```
29  grammar CSV;
30
31  csvFile: hdr row+ ;
32  hdr : row ;
33
34  row : field (',' field)* '\r'? '\n' ;
35
36  field
37      : TEXT
38      | STRING
39      |
40      ;
41
42  TEXT   : ~[,\n\r"]+ ;
43  STRING : '"' ('""'|~'"')* '"' ; // quote-quote is an escaped quote
```

Keywords (part of T)

https://github.com/antlr/grammars-v4/blob/master/csv/CSV.g4

German OWASP Day 2025

Vulnerable CSV Implementation

```java
@RestController
public class BadController {

  @GetMapping("/bad/contributions/{name}")
  public String getContribution(@PathVariable("name") String name) {
    return name + ',' + "10.000 Euro";
  }

  @GetMapping("/bad/bill")
  public String printBill(@RequestParam("contribution") String contribution) {
    String[] fields = contribution.split(regex: ",");

    StringBuilder out = new StringBuilder();
    out.append(str: "--------------------- eBon ---------------------\n");
    out.append("Name: " + fields[0] + "\n");
    out.append("Sum: " + fields[1] + "\n");
    return out.toString();
  }

}
```

Insecure unparser: String concatenation

Bad parser: Regular Expression for a context free language

German OWASP Day 2025

Good CSV Implementation

```java
@RestController
public class GoodController {

  @GetMapping("/good/contributions/{name}")
  public String getContribution(@PathVariable("name") String name) throws IOException {
    StringBuilder out = new StringBuilder();
    try (CSVPrinter printer = new CSVPrinter(out, CSVFormat.DEFAULT)) {
      printer.printRecord(name, "10.000 Euro");
    }
    return out.toString();
  }

  @GetMapping("/good/bill")
  public String printBill(@RequestParam("contribution") String contribution) throws IOException {
    CSVParser reader = CSVFormat.DEFAULT.parse(new StringReader(contribution));

    StringBuilder out = new StringBuilder();
    reader.forEach(record -> {
      out.append(str: "---------------------- eBon ----------------------\n");
      out.append("Name: " + record.get(0) + "\n");
      out.append("Sum: " + record.get(1) + "\n");
    });
    return out.toString();
  }
}
```

Unparser from a library
Work with the Unparser API

Parser from a library
Get data from the parser API

Recipe for Secure Input Handling

Use (Un)Parser Libraries that do encoding automatically correct

- Don’t write encoding code yourself
- Context-sensitive Encoding, e.g. in HTML, is too complex to get it right

When using an existing language (HTML, SQL, CSV, …)

- Find existing good (Un)Parser.
  - Fuzz-Test encoding
  - Review (Un)Parser with LangSec knowledge (Shotgun vs. Grammar based)
- Work on Parsetree to access the data

When creating your own communication language: Really? Why not JSON, or Cap’n Proto?

- Define language through grammar
- Generate (Un)Parser
- Work on Parsetree to access the data

In a perfect world…

Solving LangSec in Programming Languages

Interact with a language

- Make using good (un)parsers for existing languages easy
- Make constructing (un)parsers easy
- Build in (un)parser construction capabilities into programming languages core

Remove byte operations

- Remove simple Input Output functions like byte wise read and println()
- Remove string concatenation

Meanwhile at Google

SafeCoding - Eliminate Bug Classes

Bug Classes

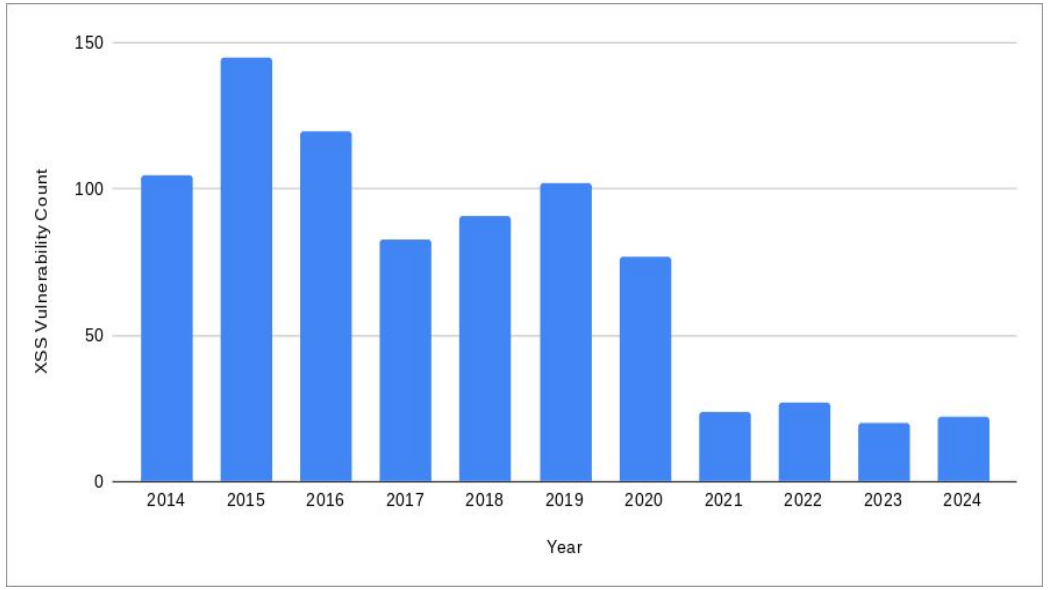
Encryption: Easy to build an application without encryption-in-transit

Injection: The web's core building blocks (HTML, JS, URLs) allow easily mixing code and data

Isolation: Cross-origin interactions between different applications are allowed by default

Solution

Providing a platform to developers that is Safe



https://bughunters.google.com/blog/6644316274294784/secure-by-design-google-s-blueprint-for-a-high-assurance-web-framework

AppSec should learn from LangSec

Input Validation is a misleading term

- Correct unparsing prevents Injections
- Strict parsing does not solve the root cause of injections, it makes programs robust against input

Computers (including AI) can not determine what programs do

- SAST and DAST try to solve an undecidable problem and fail.
- Antivirus does not work.

Eliminate Bug Classes

Communication language above deterministic context free introduce Remote Code Execution by design

“Rolling your own (un)parser is like rolling your own crypto algo”