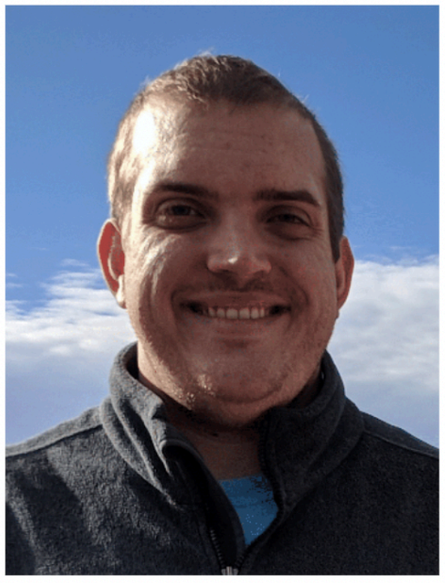
Certified Parsing of Dependent Regular Grammars

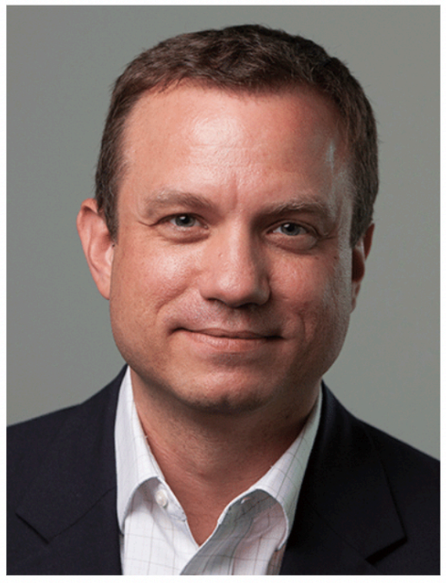


John Sarracino

Cornell University

Gang Tan

PennState



Greg Morrisett

Cornell University

Cornell Tech

Format Parsing is Hard

Name	Description
CVE-2021-38644	Microsoft MPEG-2 Video Extension Remote Code Execution Vulnerability
CVE-2021-38381	Live555 through 1.08 does not handle MPEG-1 or 2 files properly. Sending two successive RTSP SETUP commands for the same track causes a Use-After-Free and daemon crash.
CVE-2021-36937	Windows Media MPEG-4 Video Decoder Remote Code Execution Vulnerability
CVE-2021-21862	Multiple exploitable integer truncation vulnerabilities exist within the MPEG-4 decoding functionality of the GPAC Project on Advanced Content library v1.0.1. A specially crafted MPEG-4 input can cause an improper memory allocation resulting in a heap-based buffer overflow that causes memory corruption The implementation of the parser used for the “Xtra” FOURCC code is handled. An attacker can convince a user to open a video to trigger this vulnerability.
CVE-2021-21861	An exploitable integer truncation vulnerability exists within the MPEG-4 decoding functionality of the GPAC Project on Advanced Content library v1.0.1. When processing the 'hdlr' FOURCC code, a specially crafted MPEG-4 input can cause an improper memory allocation resulting in a heap-based buffer overflow that causes memory corruption. An attacker can convince a user to open a video to trigger this vulnerability.
CVE-2021-21860	An exploitable integer truncation vulnerability exists within the MPEG-4 decoding functionality of the GPAC Project on Advanced Content library v1.0.1. A specially crafted MPEG-4 input can cause an improper memory allocation resulting in a heap-based buffer overflow that causes memory corruption. The FOURCC code, 'trik', is parsed by the function within the library. An attacker can convince a user to open a video to trigger this vulnerability.
CVE-2021-21859	An exploitable integer truncation vulnerability exists within the MPEG-4 decoding functionality of the GPAC Project on Advanced Content library v1.0.1. The stri_box_read function is used when processing atoms using the 'stri' FOURCC code. An attacker can convince a user to open a video to trigger this vulnerability.
CVE-2021-21858	Multiple exploitable integer overflow vulnerabilities exist within the MPEG-4 decoding functionality of the GPAC Project on Advanced Content library v1.0.1. A specially crafted MPEG-4 input can cause an integer overflow due to unchecked addition arithmetic resulting in a heap-based buffer overflow that causes memory corruption. An attacker can convince a user to open a video to trigger this vulnerability.

MPEG CVEs, -Meridith Patterson

Format Parsing is Hard

Name	Description
CVE-2021-38644	Microsoft MPEG-2 Video Extension Remote Code Execution Vulnerability
CVE-2021-38381	Live555 through 1.08 does not handle MPEG-1 or 2 files properly. Sending two successive RTSP SETUP commands for the same track causes a Use-After-Free and daemon crash.
CVE-2	
CVE-2	ct on in a
CVE-20	Advanced Content library v1.0.1. When processing the 'hdlr' FOURCC code, a specially crafted MPEG-4 input can cause an improper memory allocation resulting in a heap-based buffer overflow that causes memory corruption. An attacker can convince a user to open a video to trigger this vulnerability.
CVE-2021-21860	An exploitable integer truncation vulnerability exists within the MPEG-4 decoding functionality of the GPAC Project on in a the
CVE-2	ode.
CVE-20	on Advanced Content library v1.0.1. A specially crafted MPEG-4 input can cause an integer overflow due to unchecked addition arithmetic resulting in a heap-based buffer overflow that causes memory corruption. An attacker can convince a user to open a video to trigger this vulnerability.

"Shotgun" parser in which the ***value*** of parsed data influences the ***future parsing action***

This is known as Dependent Parsing because parse results *depend* on prior parsed values

MPEG CVEs, -Meridith Patterson

Dependent Parsing: Netstrings

Netstrings: length-prefixed, colon and comma delimited encoding of ascii values

Dependent Parsing: Netstrings

Netstrings: length-prefixed, colon and comma delimited encoding of ascii values

Valid

5:hello,

12:hello world,

Dependent Parsing: Netstrings

Netstrings: length-prefixed, colon and comma delimited encoding of ascii values

Valid

5:hello,

12:hello world,

Invalid

3:hello,

5:hello

Dependent Parsing: Netstrings

Netstrings: length-prefixed, colon and comma delimited encoding of ascii values

Valid

5:hello,

12:hello world,

Invalid

3:hello,

5:hello

Netstrings are *dependent* because the body depends on the value of the prefix

Dependent Regular Grammars

- We extend regular languages with monadic bind.
- Enables reasoning about dependent parsers within a proof assistant.
- Implemented and formalized in Coq.

Outline

- Syntax and Semantics
- Parsing algorithm and correctness
- Netstring case study

Outline

- **Syntax and Semantics**
- Parsing algorithm and correctness
- Netstring case study

Dependent Parser Syntax

$$p ::= \emptyset \mid \ . \mid p|p \mid \texttt{pure}\ v \mid p \gg\!= f \mid p^*$$

Dependent Parser Syntax

Accept

Return

Kleene Star

$$p ::= \emptyset \mid . \mid p|p \mid \texttt{pure}\ v \mid p \gg\!= f \mid p^*$$

Reject

Union

Dependent Parser Syntax

Accept

Return

Kleene Star

$$p ::= \emptyset \mid . \mid p|p \mid \texttt{pure}\ v \mid p \gg\!= f \mid p^*$$

Reject

Union

Bind

Bind Semantics

p parses s to v: $(s, p) \rightarrowtail v$

Bind Semantics

p parses s to v: $(s, p) \rightarrowtail v$

PBIND

$$\frac{(s_l, p) \rightarrowtail x \qquad (s_r, f\ x) \rightarrowtail y}{(s_l +\!\!+ s_r, p \gg\!= f) \rightarrowtail y}$$

Bind Semantics

p parses s to v: $(s, p) \rightarrowtail v$

If p parses s_l to x,

PBIND

$$\frac{(s_l, p) \rightarrowtail x \qquad (s_r, f\ x) \rightarrowtail y}{(s_l +\!\!+ s_r, p \gg\!= f) \rightarrowtail y}$$

Bind Semantics

p parses s to v: $(s, p) \rightarrowtail v$

If p parses s_l to x,

and $f\ x$ parses s_r to y,

PBIND

$$\frac{(s_l, p) \rightarrowtail x \qquad (s_r, f\ x) \rightarrowtail y}{(s_l +\!\!+ s_r, p \gg\!= f) \rightarrowtail y}$$

Bind Semantics

p parses s to v: $(s, p) \rightarrowtail v$

If p parses s_l to x,

and $f\ x$ parses s_r to y,

$$\textsc{PBind}\ \frac{(s_l, p) \rightarrowtail x \qquad (s_r, f\ x) \rightarrowtail y}{(s_l +\!\!+ s_r, p \gg\!= f) \rightarrowtail y}$$

then $p \gg\!= f$ parses $s_l +\!\!+ s_r$ to y

Parsing an Even Digit

$$\text{digit} \gg\!= \lambda x \to \text{if } x\%2 \text{ then return } x \text{ else fail}$$

Parsing an Even Digit

Parse a digit

$$\text{digit} \gg\!= \lambda x \to \text{if } x\%2 \text{ then return } x \text{ else fail}$$

Parsing an Even Digit

Parse a digit

$$\text{digit} \gg\!= \lambda x \to \text{if } x\%2 \text{ then return } x \text{ else fail}$$

Accept if even

Parsing an Even Digit

Parse a digit

Otherwise reject

$$\text{digit} \gg\!= \lambda x \to \text{if } x\%2 \text{ then return } x \text{ else fail}$$

Accept if even

Outline

- **Syntax and Semantics**
- Parsing algorithm and correctness
- Netstring case study

Outline

- Syntax and Semantics
- **Parsing algorithm and correctness**
- Netstring case study

Parsing Algorithm

- How do we know that the actual parsing algorithm is implemented correctly?

- How do we know it terminates?

- We address both:
  1) we define the algorithm in Coq and prove correctness using the parsing semantics;
  2) we prove termination in Coq as well.

Parsing with Derivatives

We extend Brzozowski derivatives to Dependent Regular Grammars by defining two functions:

$|| \ p \ ||_\epsilon$

Finite set of return values of p

$D_c \ p$

Derivative of p with respect to c

We parse by iteratively applying derivatives and then building the set of return values.

Epsilon Interpretations

Specification: $v \in || p ||_\epsilon \iff p \text{ parses } \epsilon \text{ to } v$

Parser	$\|\| p \|\|_\epsilon$
return 5	{5}
fail	{}
return 2 | return 3 | .	{2,3}
return 5 $>\!\!>\!\!=$ $\lambda n \to$ return $n+1$	{6}
digit*	{[]}

Derivative Intuition

$$D_c\ p$$

Specification: p parses $c :: s$ to $v \iff D_c\ p$ parses s to v

Since the derivative preserves parse behavior, we can parse by iteratively applying the derivative operator.

Iterated Derivative

Parsing two characters on the string “xy”

$$. \gg\!= \lambda x \to . \gg\!= \lambda y \to \text{return } (x, y)$$

Iterated Derivative

Parsing two characters on the string “xy”

$$. \gg\!= \lambda x \to . \gg\!= \lambda y \to \text{return } (x, y)$$

D_x

$$\text{return } `x' \gg\!= \lambda x \to . \gg\!= \lambda y \to \text{return } (x, y)$$

Iterated Derivative

Parsing two characters on the string “xy”

$$. \gg\!= \lambda x \to . \gg\!= \lambda y \to \text{return } (x, y)$$

D_x

$$\text{return } `x' \gg\!= \lambda x \to . \gg\!= \lambda y \to \text{return } (x, y)$$

simplify

$$. \gg\!= \lambda y \to \text{return } (`x', y)$$

Iterated Derivative

Parsing two characters on the string “xy”

$$. \gg\!= \lambda x \to . \gg\!= \lambda y \to \text{return } (x, y)$$

D_x

$$\text{return } `x' \gg\!= \lambda x \to . \gg\!= \lambda y \to \text{return } (x, y)$$

simplify

$$. \gg\!= \lambda y \to \text{return } (`x', y)$$

D_y, simplify

$$\text{return } (`x', `y')$$

Outline

- Syntax and Semantics
- **Parsing algorithm and correctness**
- Netstring case study

Outline

- Syntax and Semantics
- Parsing algorithm and correctness
- **Netstring case study**

Case study: Netstrings

Proof of concept verified parser for a dependent format (known as calc-regular languages).

```
Definition net_str :=
  num >>= fun n =>
  char ":" $> ((repeat n pAny) <$ char ",").
```

Case study: Netstrings

Proof of concept verified parser for a dependent format (known as calc-regular languages).

Clear, concise!!

```
Definition net_str :=
  num >>= fun n =>
  char ":" $> ((repeat n pAny) <$ char ",").
```

Case study: Netstrings

Proof of concept verified parser for a dependent format (known as calc-regular languages).

Clear, concise!!

```
Definition net_str :=
  num >>= fun n =>
  char ":" $> ((repeat n pAny) <$ char ",").

Theorem net_str_spec:
  forall s v,
  Parses s net_str v <->
  exists s',
    s = s' ++ (":" :: v ++ ("," :: nil)) /\
    Parses s' num (length v).
```

Case study: Netstrings

Proof of concept verified parser for a dependent format (known as calc-regular languages).

Clear, concise!!

Strong spec!!

```
Definition net_str :=
  num >>= fun n =>
  char ":" $> ((repeat n pAny) <$ char ",").

Theorem net_str_spec:
  forall s v,
  Parses s net_str v <->
  exists s',
    s = s' ++ (":" :: v ++ ("," :: nil)) /\
    Parses s' num (length v).
```

Future directions

- Interpreting parsers in Coq is slow (too slow for realistic input sizes).

- I want to collaborate with **YOU** on verifying a security property of a real-world parser!

- Interpreting parsers in Coq is slow (too slow for realistic input sizes).

- I want to collaborate with **YOU** on verifying a security property of a real-world parser!

$$p ::= \emptyset \mid \ . \mid p|p \mid \texttt{pure}\ v \mid p \gg\!= f \mid p^*$$

Thanks! Questions?

jsarracino@cornell.edu

https://github.com/jsarracino/dependent-regular-grammars

Backup slides

Dependent

Format parsing is hard

Name	Description
CVE-2021-38644	Microsoft MPEG-2 Video Extension Remote Code Execution Vulnerability
CVE-2021-38381	Live555 through 1.08 does not handle MPEG-1 or 2 files properly. Sending two successive RTSP SETUP commands for the same track causes a Use-After-Free and daemon crash.
CVE-2021-36937	Windows Media MPEG-4 Video Decoder Remote Code Execution Vulnerability
CVE-2021-21862	Multiple exploitable integer truncation vulnerabilities exist within the MPEG-4 decoding functionality of the GPAC roject on Advanced Content library v1.0.1. A specially crafted MPEG-4 input can cause an improper memory allocati lting in a heap-based buffer overflow that causes memory corruption The implementation of the parser used for the “Xtra” FOURCC code is handled. An attacker can convince a user to open a video to trigger this vulnerability.
CVE-2021-21861	An exploitable integer truncation vulnerability exists within the MPEG-4 decoding functionality of the GPAC Pro on Advanced Content library v1.0.1. When processing the 'hdlr' FOURCC code, a specially crafted MPEG-4 in an improper memory allocation resulting in a heap-based buffer overflow that causes memory corruption. An a can convince a user to open a video to trigger this vulnerability.
CVE-2021-21860	An exploitable integer truncation vulnerability exists within the MPEG-4 decoding functionality of the GPAC Proje on Advanced Content library v1.0.1. A specially crafted MPEG-4 input can cause an improper memory allocatio lting in a heap-based buffer overflow that causes memory corruption. The FOURCC code, 'trik', is parsed by the fu he library. An attacker can convince a user to open a video to trigger this vulnerability.
CVE-2021-21859	An exploitable integer truncation vulnerability exists within the MPEG-4 decoding functionality of the GPAC Pro on Advanced Content library v1.0.1. The stri_box_read function is used when processing atoms using the 'str de. An attacker can convince a user to open a video to trigger this vulnerability.
CVE-2021-21858	Multiple exploitable integer overflow vulnerabilities exist within the MPEG-4 decoding functionality of the GPAC P oject on Advanced Content library v1.0.1. A specially crafted MPEG-4 input can cause an integer overflow due to un addition arithmetic resulting in a heap-based buffer overflow that causes memory corruption. An attacker a user to open a video to trigger this vulnerability.

Parser Semantics

PANY

$$\frac{}{(c :: \epsilon, \ .) \rightarrowtail c}$$

PPURE

$$\frac{(s, p_l) \rightarrowtail v}{(s, p_l | p_r) \rightarrowtail v}$$

PALTL

$$\frac{(s, p_r) \rightarrowtail v}{(s, p_l | p_r) \rightarrowtail v}$$

PALTR

$$\frac{}{(\epsilon, \texttt{pure}\ v) \rightarrowtail v}$$

PSTAR0

$$\frac{}{(\epsilon, p^*) \rightarrowtail []}$$

PSTARITER

$$\frac{s \neq \epsilon \qquad (s, p) \rightarrowtail x \qquad (s', p^*) \rightarrowtail xs}{(s +\!\!+ s', p^*) \rightarrowtail x :: xs}$$

PBIND

$$\frac{(s_l, p) \rightarrowtail x \qquad (s_r, f\ x) \rightarrowtail y}{(s_l +\!\!+ s_r, p \gg\!= f) \rightarrowtail y}$$

To ensure well-behaved parser semantics, Star interior is productive

Parser Syntax

Traditional

$$\mathbf{r} ::= \mathbf{1} \mid \mathbf{0} \mid c \mid \mathbf{r_l} \oplus \mathbf{r_r} \mid \mathbf{r} \mathbin{@} \mathbf{F} \mid \mathbf{r}^* \mid \mathbf{r_l} \# \mathbf{r_r}$$

Dependent

$$p ::= \emptyset \mid \; . \mid p|p \mid \texttt{pure}\ v \mid p \gg\!= f \mid p^*$$

Interestingly, bind subsumes map and concat

Epsilon and Derivative Definitions

$$||\ p\ ||_\epsilon$$

$$
\begin{aligned}
\|\emptyset\|_\epsilon &= [] \\
\|\cdot\|_\epsilon &= [] \\
\|p_l|p_r\|_\epsilon &= \|p_l\|_\epsilon +\!\!+ \|p_r\|_\epsilon \\
\|\texttt{pure}\ v\|_\epsilon &= [v] \\
\|p^*\|_\epsilon &= [[]] \\
\|p \gg\!= f\|_\epsilon &= \texttt{concat\_map}\ (\lambda x \Rightarrow \|f\ x\|_\epsilon)\ \ \|p\|_\epsilon
\end{aligned}
$$

$$D_c\ p$$

$$
\begin{aligned}
\mathbf{D}_c\ \emptyset &= \emptyset \\
\mathbf{D}_c\ \cdot &= \texttt{pure}\ c \\
\mathbf{D}_c\ p_l|p_r &= \mathbf{D}_c\ p_l|\mathbf{D}_c\ p_r \\
\mathbf{D}_c\ \texttt{pure}\ v &= \emptyset \\
\mathbf{D}_c\ p^* &= (\mathbf{D}_c\ p)\ \$\ p^* \\
\mathbf{D}_c\ p \gg\!= f &= (\mathbf{D}_c\ p \gg\!= f)|\bigvee_{x \in \|p\|_\epsilon} \mathbf{D}_c\ (f\ x)
\end{aligned}
$$