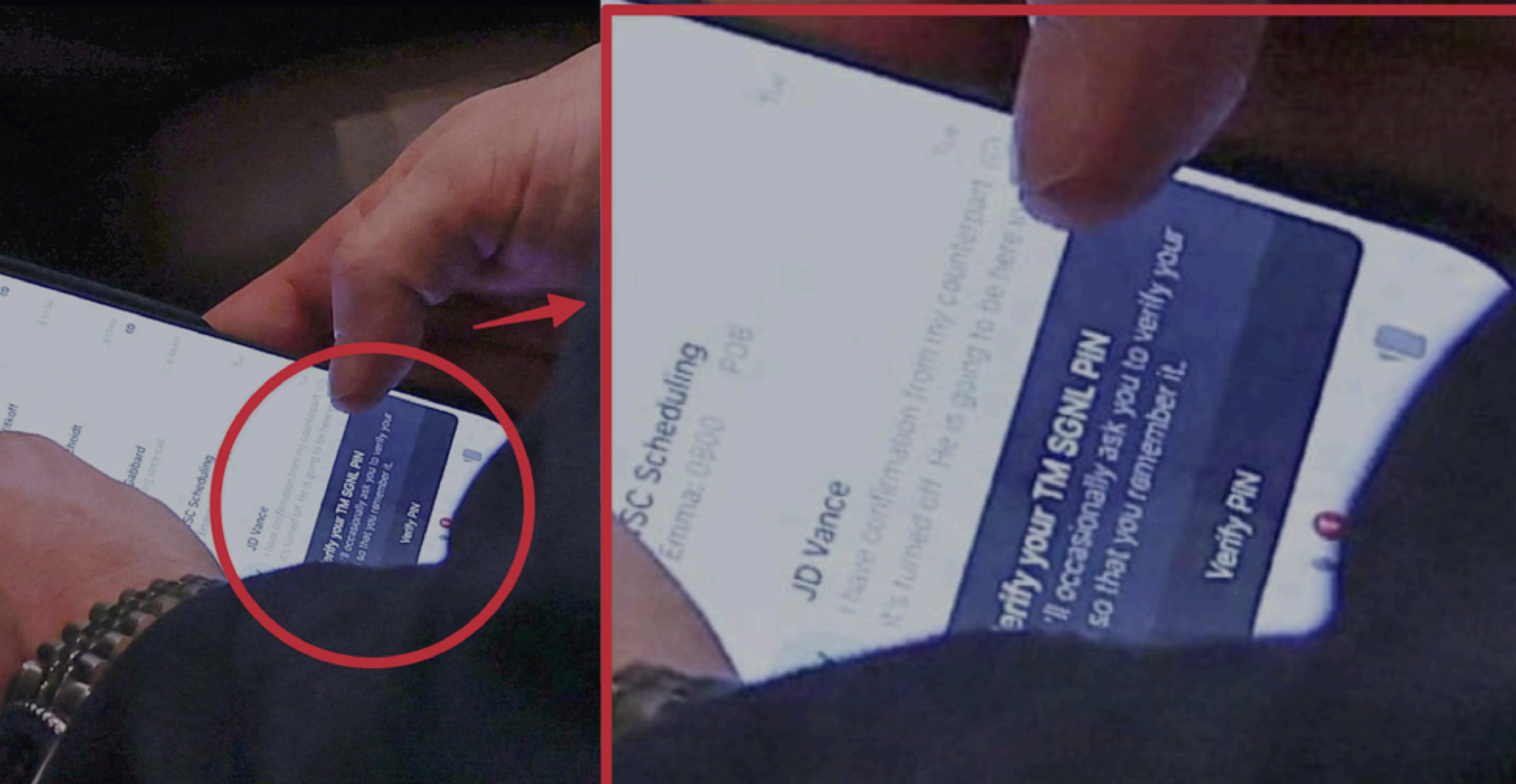
Carving JSON in heap dumps

👋 Hunter Domson @ Reporters United

How it started



404 Media: Mike Waltz Accidentally Reveals Obscure App the Government Is Using to Archive Signal Messages

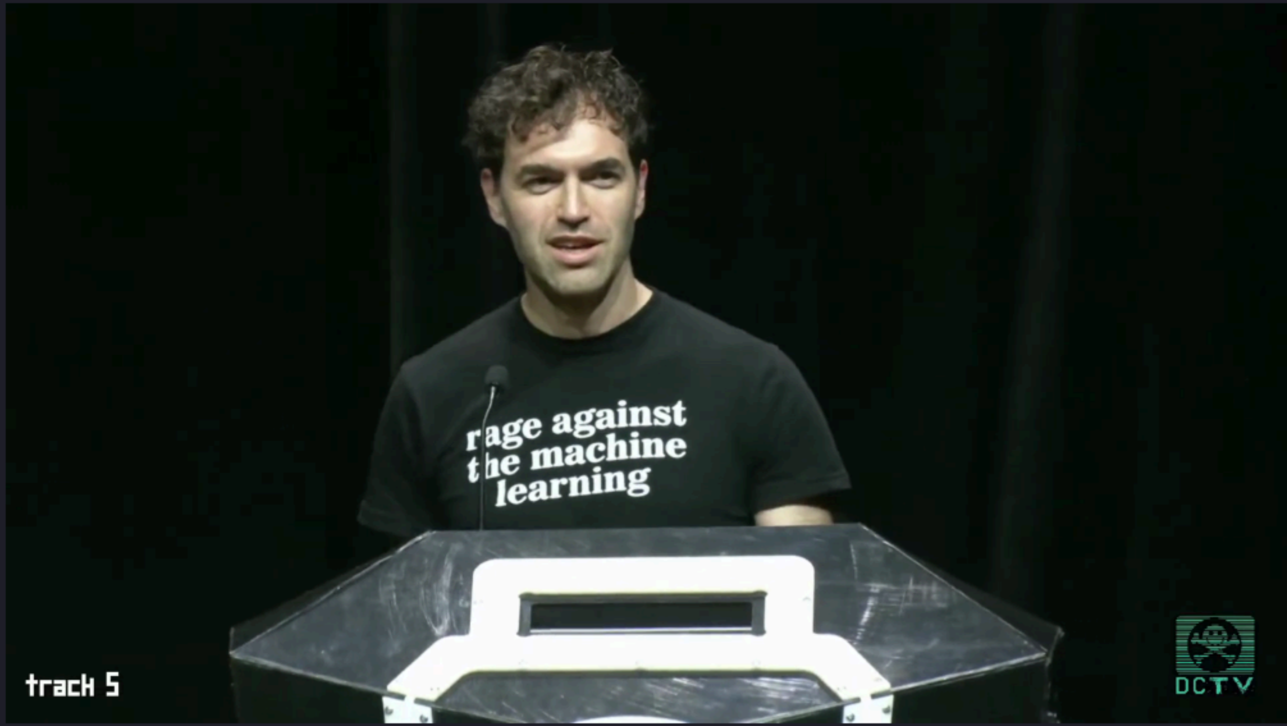
On May 1st, 2025, Mike Waltz (Trump's national security advisor) looked at his phone.

On May 5th, 2025, Telemessage shut down.

In between, Telemessage was hacked twice. DDoSecrets hosts heap dumps from their archival service.

Full saga in

"We are currently clear on OPSEC"



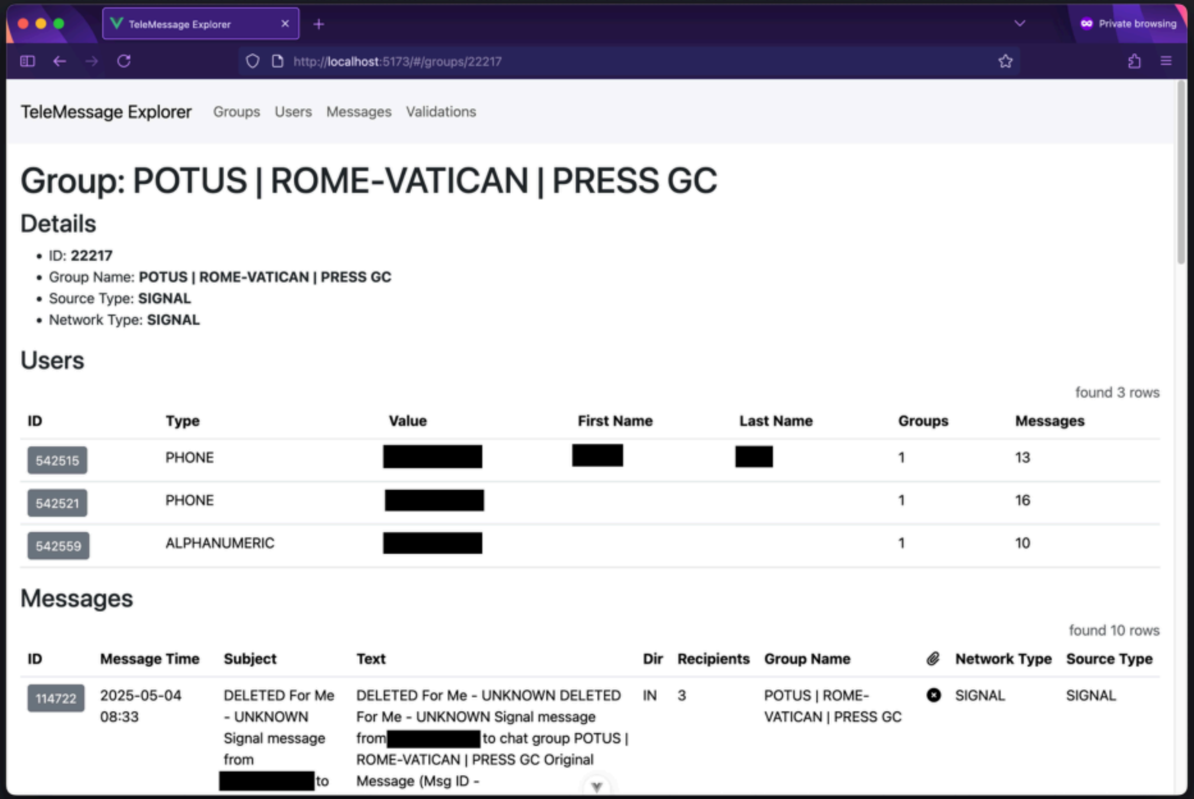
Still of Micah Lee's presentation in DEFCON 33

Heap dumps you said?

- Thousands server heap dumps, ~400 GB in total
- Basically copies of server memory every few minutes
- Messages lying around in JSON format

Macro","groupId":"17169071900-1521081435@g.us","groupMessage":true,"text":"<Kaf
kafied>:/shared/cloud/apigateway/attachments/2025_05_04_01_tm_tmp/str1452741252 4
630134620.txt"},"kafkafied":true}",\"type\":\"PHONE\"}":{"firstName":"[redacted]","la
stName":"[redacted]"},"{\"value\":\"[redacted]\",\"type\":\"PHONE\"}":{"firstName"
:"[redacted]","lastName":"[redacted]"},"{\"value\":\"[redacted]\",\"type\":\"PHONE\"}":{"fir
stName":"[redacted]","lastName":"[redacted]"},"{\"value\":\"[redacted]\",\"type\":\"PHON
E\"}":{"firstName":"[redacted]","lastName":"[redacted]"},"{\"value\":\"[redacted]\",\"typ
e\":\"PHONE\"}":{"firstName":"[redacted]","lastName":""}},"originalMessageData":null,
"ban":null,"acceptedPayloadIdentifier":"2f59a3f2-bb26-4369-847f-5024dfab1d32","g
roupName":"GD Macro","groupId":"17169071900-1521081435@g.us","groupMessage":true
,"text":"Just spoke to a D staffer on the senate side - 2 cosponsors (Alsobrooks
and gillibrand) did not sign the opposition letter so they think the bill still
has a good chance of passage the senate with 5 more Ds supporting it."},"kafkaf
ied":true}[redacted]\",\"type\":\"PHONE\"}":{"firstName":"[redacted]","lastName":""},"{\"v
alue\":\"[redacted]\",\"type\":\"PHONE\"}":{"firstName":"[redacted]","lastName":
"[redacted]"},"{\"value\":\"[redacted]\",\"type\":\"PHONE\"}":{"firstName":"[redacted]
","lastName":"[redacted]"},"{\"value\":\"[redacted]\",\"type\":\"PHONE\"}":{"firstN
ame":"[redacted]","lastName":"[redacted]"},"{\"value\":\"[redacted]\",\"type\":\"PH
ONE\"}":{"firstName":"[redacted]","lastName":"[redacted]"},"{\"value\":\"[redacted]\",\"
type\":\"PHONE\"}":{"firstName":"[redacted]","lastName":"[redacted]"},"{\"value\":\"[redacted]
\",\"type\":\"PHONE\"}":{"firstName":"[redacted]","lastName":"[redacted]"},"{\"val
ue\":\"[redacted]\",\"type\":\"PHONE\"}":{"firstName":"ingrid","lastName":"mess

- Micah created Telemessage Explorer
- Used `strings(1)` to extract JSON strings.
- Bespoke Python code to parse message types
- Docker / PostgreSQL / Vue.js



We tried to search for Greek characters in the dataset, but didn't find any.

Turns out that:

```
$ echo τεστ | strings
$ echo τεστ | strings -e S
τεστ
```

Issues with `strings -e S`

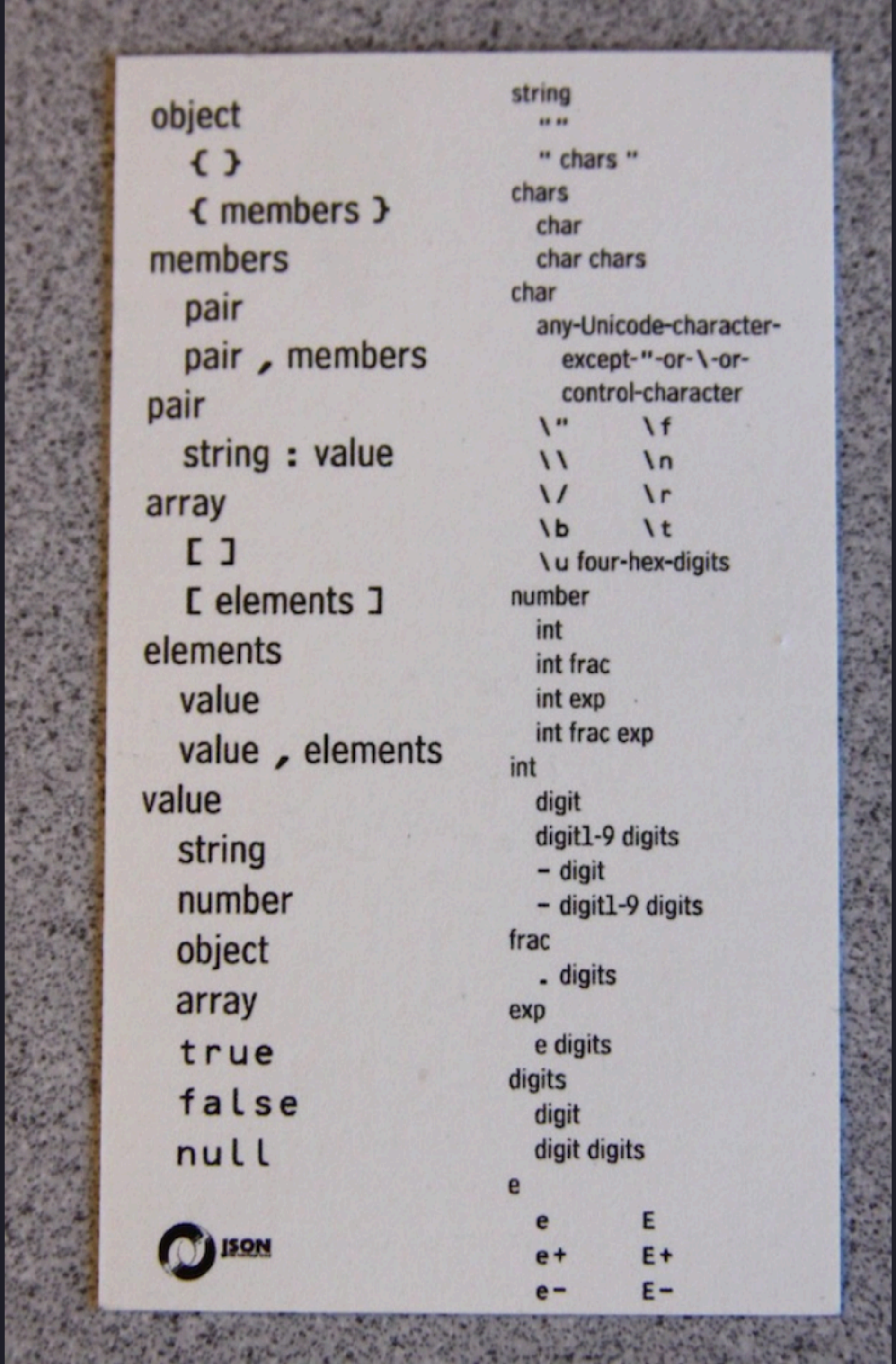
- ~160,000 Unicode characters
- Does not filter printable characters
- Mangled parsing if next to JSON string (happens often)

And what about?

- Multiple JSON strings on same run
- Multi-line JSON strings

No JSON carver out there 🙁

But can we create our own?



Back side of Douglas Crockford's business card

JSON Carver

- Written in Rust
- Less than 1,000 lines of code.
- Apache 2.0 / MIT

https://github.com/reportersunited/json-carver

How it works (1/4)

Quickly read from an I/O stream until it reaches a left square bracket (`[`), or a left curly bracket (`{`):

```
fn scout(&mut self) -> Result<Option<(usize, u8)>, io::Error> {
    loop {
        match memchr::memchr2(CHAR_LEFT_SQUARE_BRACKET, CHAR_LEFT_CURLY_BRACKET, available) {...}
    }
}
```

`memchr` powers `ripgrep` , uses SIMD/AVX2 where possible.

How it works (2/4)

Check the next legal character:

```
fn handle_left_square_bracket(&mut self) -> Result<Cause, io::Error> {
    self.jt.add_ident(CHAR_LEFT_SQUARE_BRACKET);
    for b in self.reader.mut_ref().bytes() {
        let b = b?;
        match b {
            CHAR_LEFT_SQUARE_BRACKET
            | CHAR_LEFT_CURLY_BRACKET
            | CHAR_RIGHT_SQUARE_BRACKET
            | CHAR_QUOT_MARK
            | CHAR_MINUS
            | CHAR_ZERO..=CHAR_NINE
            | CHAR_START_FALSE
            | CHAR_START_NULL
            | CHAR_START_TRUE => return Ok(Cause::Found(b)),
            CHAR_SPACE | CHAR_TAB | CHAR_NEWLINE | CHAR_CARRIAGE_RETURN => self.jt.advance(b),
            _ => return Ok(Cause::Corrupted(b)),
        }
    }
    Ok(Cause::Exhausted)
}
```

How it works (2/4)

Surprisingly, the code never needs to backtrack.
Here's all we need:

- Current character (`{`)
- Next character (`"`)
- In JSON key? (`true`)
- Indentation level (`2`)
- Indentations (`['[', '{']`) (4MiB array)

How it works (3/4)

Extend buffer at 4MiB intervals to store currently processed string:

```
fn advance(&mut self, mut b: u8) {
    // See how ripgrep handles the "very large lines" problem:
    // https://github.com/BurntSushi/ripgrep/issues/2959
    if self.replace_newlines && b == CHAR_NEWLINE {
        b = CHAR_SPACE;
    }
    if self.cur < self.processed.len() {
        self.processed[self.cur] = b;
    } else {
        self.processed.reserve(BUF_EXTEND_SIZE);
        self.processed.push(b);
    }
    self.cur += 1;
}
```

How it works (4/4)

Continue with next character, print string, or report corrupted:

```
ch = match res {
    Ok(Cause::Completed) => {
        return Ok(Cause::Completed);
    }
    Ok(Cause::Found(ch)) => ch,
    Ok(Cause::Corrupted(ch)) => {
        return Ok(Cause::Corrupted(ch));
    }
    Ok(Cause::Exhausted) => {
        return Ok(Cause::Exhausted);
    }
}
```

Is it fast?

Windows VM image (~82 GiB) - `strings(1)`

```
$ time strings /path/to/win.qcow > /tmp/strings.results
real    10m30.955s
user    9m59.717s
sys     0m28.554s
$ ls -lh /tmp/strings.results
-rw-r--r-- 1 user user 11G Jan 21 15:47 /tmp/strings.results
```

Windows VM image (~82 GiB) - `json-carver`

```
$ time json-carver -i /path/to/win.qcow -o /tmp/carve.results
real    1m44.513s
user    0m27.947s
sys     0m29.098s
$ ls -lh /tmp/carve.results
-rw-r--r-- 1 user user 247M Jan 21 15:34 /tmp/carve.results
```

6x faster than `strings(1)`

45x space efficiency

4 GB from `/dev/urandom` - `strings(1)`

```
$ time dd if=/dev/urandom bs=4MB count=1000 | strings > /dev/null
1000+0 records in
1000+0 records out
4000000000 bytes (4.0 GB, 3.7 GiB) copied, 39.0745 s, 102 MB/s

real    0m39.079s
user    0m31.588s
sys     0m9.423s
```

4 GB from `/dev/urandom` - `json-carver`

```
time dd if=/dev/urandom bs=4MB count=1000 | json-carver &>/dev/null
1000+0 records in
1000+0 records out
4000000000 bytes (4.0 GB, 3.7 GiB) copied, 9.20211 s, 435 MB/s

real	0m9.205s
user	0m1.663s
sys	0m8.270s
```

4x faster than `strings(1)`

4 GB from `/dev/urandom` - noop

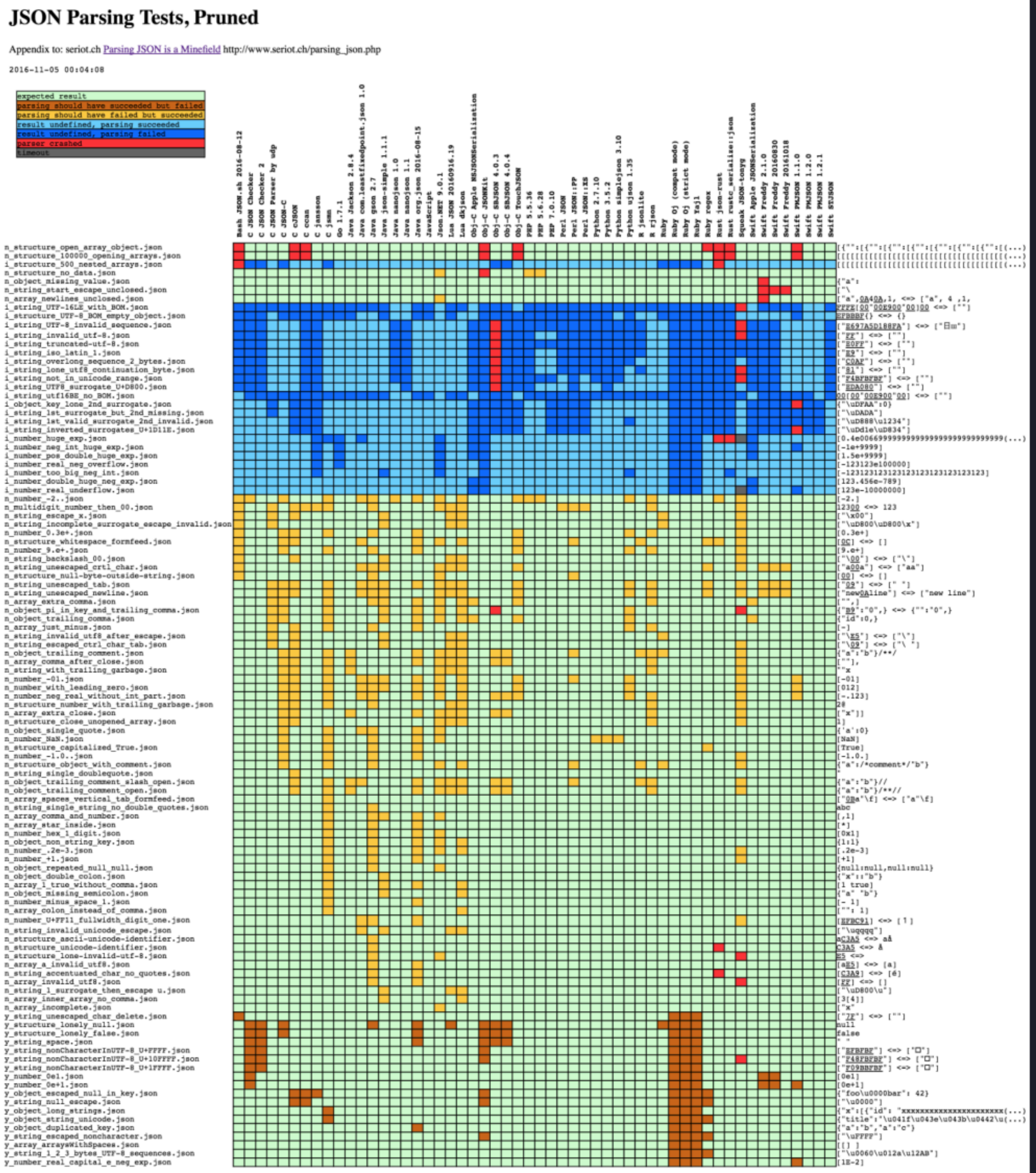
```
$ time dd if=/dev/urandom bs=4MB count=1000 > /dev/null
1000+0 records in
1000+0 records out
4000000000 bytes (4.0 GB, 3.7 GiB) copied, 6.89616 s, 580 MB/s

real    0m6.899s
user    0m0.002s
sys     0m6.888s
```

25% overhead over plain I/O

Is it accurate?

- RFC-8259 compliant
- Tested against JSONTestSuite



Any more features?

Return results in one line

```
$ echo -e "[\n1][\n1][\n2]" | json-carver --replace-newlines | sort -u
[ 1]
[ 2]
```

Reconstruct partially corrupted JSON

```
$ echo '{"valid": [1,2], "nope00000' | json-carver --fix-incomplete
corrupted,0,26,14
{"valid": [1,2]}
```

Ok, but was it worth it?

	Messages	Groups	Users
`strings(1)`	~60,000	~3,500	~44,500
`json-carver`	~100,000	~8,300	~145,000
Improvement	**x1.7**	**x2.4**	**x3.25**

Questions?

Credits

- DDoSecrets, for giving us access
- Micah Lee, for all the work on Telemessage Explorer

Resources

- https://www.reportersunited.gr/en/
- https://github.com/reportersunited/json-carver/