

DiaMon conference – 18 Nov. 2016

# Introduction to CTF 2

Common Trace Format

*Effici*OS

**Philippe Proulx**  
pproulx@efficios.com   
eepp 

# About the speaker

**Philippe Proulx**

pproulx@efficios.com 

[github.com/epp](https://github.com/epp) 

- Works at EfficiOS (company behind LTTng) in Montréal, Canada.
- Interests in tracing, file formats, protocols, embedded systems, and documentation.
- Author of barectf (tracer generator for bare-metal systems).
- Author of the CTF 2 proposal.

# Contents

1. What is CTF?
2. Why the major bump?
3. Major design goals of CTF 2
4. What's new in CTF 2?
5. Potential future extensions
6. Planned adoption
7. Resources & questions

# What is CTF?

## CTF: Common Trace Format

Data stream #1

Data stream #2

Data stream #3

Metadata stream

# What is CTF?

## CTF 1 metadata stream

```
// ...
event.header := struct {
    uint64 timestamp;
    uint16 id;
};
// ...
event {
    name = new_msg;
    id = 23;
    fields := struct {
        uint32 msg_id;
        string msg;
    } align(32);
};
// ...
```

## CTF data stream

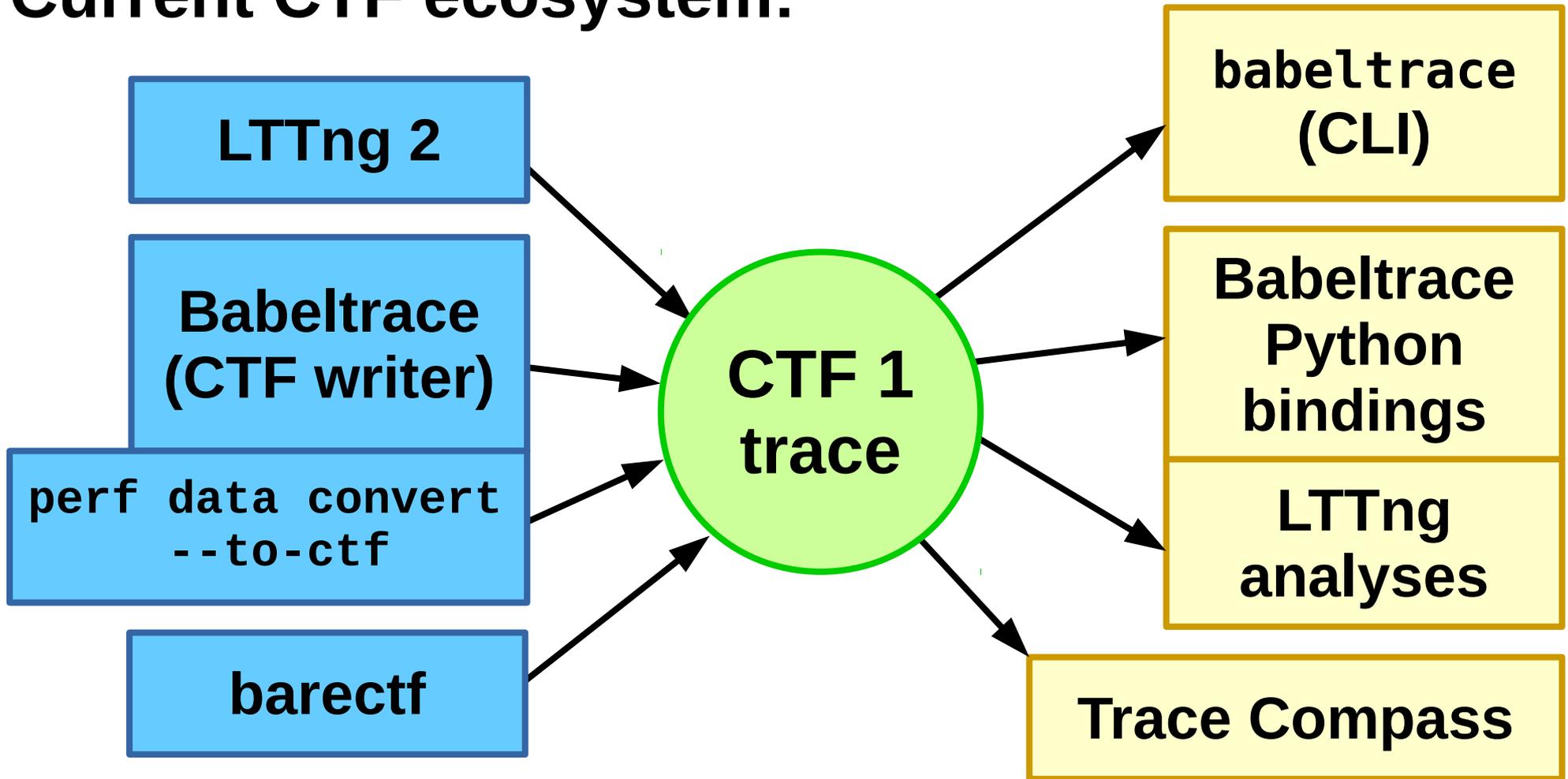
```
... 7d ee 9c b8 8b 99 d1
89 dd ed 84 c3 02 00 00
17 00 00 00 2d ff 00 00
48 65 6c 6c 6f 2c 20 57
6f 72 6c 64 21 00 2d ff
40 52 d9 8d ff 90 ff...
```

Encoded event record:

- Name: "new\_msg"
- timestamp: **15h47:11.2839912**
- msg\_id: **65325 (0xff2d)**
- msg: "**Hello, World!**"

# What is CTF?

## Current CTF ecosystem:



# Why the major bump?

## Major limitations of CTF 1 (current version):

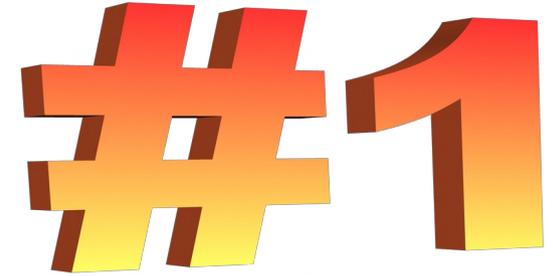
- Metadata stream is written in a custom language (TSDL) which is **complicated to parse correctly** without bugs.
- TSDL is strict: it has **no extension points** for custom user metadata and future enhancements.
- TSDL has **no specified support for event record namespacing and versioning**.
- CTF 1 has **no support for variable-length integer and union fields**, both of which can be very useful.

# Why the major bump?

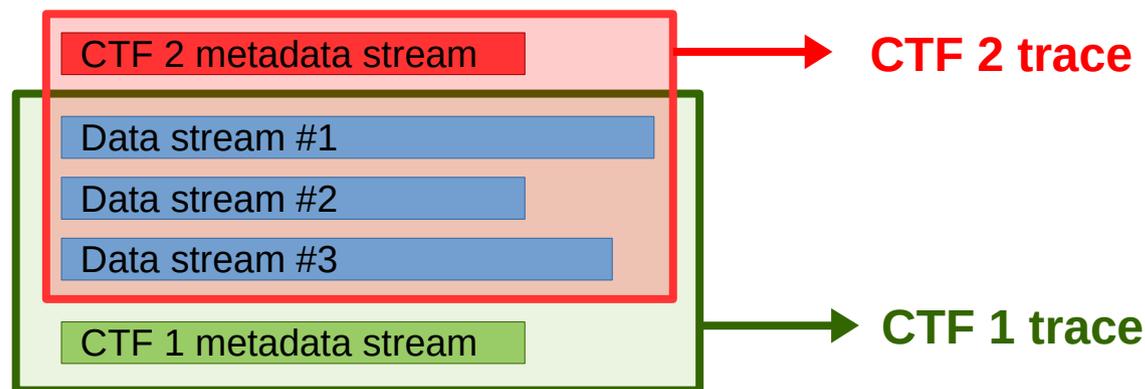
## CTF 1 $\neq$ CTF 2

- Metadata stream is *not* backward compatible. Its language is different.
- Data streams *are* backward compatible.
- A CTF 1-only consumer cannot decode a CTF 2 trace.
- A CTF 2-only consumer cannot decode a CTF 1 trace.

# Major design goals of CTF 2



CTF 2 data streams *must* be compatible with CTF 1 data streams.



# Major design goals of CTF 2

A large, stylized graphic of the number 2 preceded by a hash symbol (#). The graphic is rendered in a 3D style with a color gradient from red at the top to yellow at the bottom, and a dark brown shadow effect.

The CTF 2 streams *must* be as **efficient as possible** to produce by a tracer.

# Major design goals of CTF 2



CTF 2's model should be as close as possible to CTF 1's model.

# Major design goals of CTF 2



A CTF 2 trace should be **as easy as possible to consume.**

# Major design goals of CTF 2

A CTF 2 trace should  
to consume.

Parse *this*:

*Effici*OS

```
struct {
  typealias integer {size = 33;} :=
    some_int;
  enum : integer {
    size = 17;
    align = 0b100;
    byte_order = be; // big endian
    base = x;        /* "hex" */
    signed = true;
  } {
    INIT = 0x23d,
    /* best */ state = -50 ... 21,
  } state[17]
    [stream.packet.context.a.b.c];
  variant var <previous.selection> {
    some_int CHOICE0;
    struct {string z;}
      align(32) SOME_ENTRY[2];
  };
} align(64);
```

# Major design goals of CTF 2

# #5

A CTF 2 metadata stream *must* be **extensible** by producers and by future minor revisions of the specification (forward compatibility).

# Major design goals of CTF 2

# #6

CTF 2's specification should focus on how to encode and decode **data streams**.

~~integer base~~

~~event name~~

~~trace's environment~~

~~event's log level~~

# Major design goals of CTF 2



CTF 2's specification *must not* specify how to **transport** or **store** a trace.

```
trace dir/  
  metadata  
  stream0  
  stream1
```

# What's new in CTF 2?

DiaMon workgroup  
publishes CTF 2 documents



*Look at  
those pixels!*

# What's new in CTF 2?

## Terminology update

*Event*

*Event record class*

*Stream*

*Stream class*

*Clock*

*Clock class*

*Declaration*

*Field type*

*Event's fields*

*Event's payload*

...

...

# What's new in CTF 2?

Metadata stream is written in JSON

```
event {
  id = 23;
  name = "my_event";
  loglevel = 4;
  fields := struct {
    my_int intField;
    string stringField;
  } align(64);
};
```



```
{
  "fragment": "event-record-class",
  "user-attrs": {
    "diamon.org/ctf/ns/basic": {
      "name": "my_event",
      "log-level": 4
    }
  },
  "id": 23,
  "payload-field-type": {
    "field-type": "struct",
    "alignment": 64,
    "fields": [
      {
        "name": "intField",
        "field-type": "my_int"
      },
      {
        "name": "stringField",
        "field-type": {
          "field-type": "string"
        }
      }
    ]
  }
}
```

# What's new in CTF 2?

## Metadata stream is written in JSON

- Structure field names are strings → any name is allowed.
- Fields with special semantics to decode data streams can also have any name (e.g. *magic*, *stream\_id*, *id*, *events\_discarded*).
  - They are *tagged* as special fields instead of strictly relying on the name.

# What's new in CTF 2?

Metadata objects can contain namespaced user attributes

```
{
  "fragment": "event-record-class",
  "user-attrs": {
    "diamon.org/ctf/ns/basic": {
      "name": "my_event",
      "log-level": 4
    }
  },
  "id": 23,
  "payload-field-type": {
    "field-type": "struct",
    "alignment": 64,
    "fields": [
      {
        "name": "intField",
        "field-type": "my_int"
      },
      {
        "name": "stringField",
        "field-type": {
          "field-type": "string"
        }
      }
    ]
  }
}
```

```
{
  "field-type": "int",
  "size": 32,
  "alignment": 32,
  "byte-order": "le",
  "user-attrs": {
    "std": {
      "base": 16
    }
  }
}
```

```
{
  "field-type": "string",
  "user-attrs": {
    "my custom namespace": {
      "this is a special string": true,
      "style": {
        "fg": "#15cd49",
        "bg": "#c0c0c0"
      }
    },
    "other namespace": {
      "hide when": "Hello, World!"
    }
  }
}
```

# What's new in CTF 2?

New field type:  
*bit array*

- Parent field type of all **fixed-size field types**
  - Integer, boolean, floating point number, enumeration
- Contains the **size** and **byte order** properties
- *Not* an integer field type: has no sign property

# What's new in CTF 2?

New field type:  
*bit array*

## CTF 2 metadata stream

```
// ...  
{  
  "field-type": "bitarray",  
  "alignment": 16,  
  "size": 24,  
  "byte-order": "le"  
}  
// ...
```

## CTF data stream

```
... 7d ee 9c b8 8b 99 d1  
89 dd ed 84 c3 02 00 00  
17 00 00 00 2d ff 00...
```

Encoded field's value:

- **11000011 10000100 11101101**
- **0xc384ed**

# What's new in CTF 2?

## New field type: *null*

- Special field type which represents a 0-bit data field
- Still has an alignment property
- Cleaner than empty structure in some situations
  - E.g., as a variant field type's choice

# What's new in CTF 2?

New field type:  
*null*

## CTF 2 metadata stream

```
// ...  
{  
  "field-type": "null",  
  "alignment": 16  
}  
// ...
```

## CTF data stream

```
...7d ee 9c b8 8b 99 d1  
89 dd | ed 84 c3 02 00 00  
17 00 00 00 2d ff 00...
```

Encoded field's value:

- **null** (only possible value)

# What's new in CTF 2?

## New field type: *boolean*

- Inherit the properties of a bit array field type
  - Alignment, byte order, size in bits
- Specific meaning:
  - All the bits are 0  $\Rightarrow$  ***false***
  - Anything else  $\Rightarrow$  ***true***

# What's new in CTF 2?

New field type:  
*boolean*

## CTF 2 metadata stream

```
// ...  
{  
  "field-type": "bool",  
  "alignment": 16,  
  "size": 24,  
  "byte-order": "be"  
}  
// ...
```

## CTF data stream

```
... 7d ee 9c b8 8b 99 d1  
89 dd ed 84 c3 02 00 00  
17 00 00 00 2d ff 00...
```

Encoded field's value:

- As bit array: **0xc384ed**
- As boolean: **true**

# What's new in CTF 2?

New field type:  
*boolean*

## CTF 2 metadata stream

```
// ...  
{  
  "field-type": "bool",  
  "alignment": 16,  
  "size": 24,  
  "byte-order": "be"  
}  
// ...
```

## CTF data stream

```
...7d ee 9c b8 8b 99 d1  
89 dd 00 00 00 02 00 00  
17 00 00 00 2d ff 00...
```

Encoded field's value:

- As bit array: **0x000000**
- As boolean: **false**

# What's new in CTF 2?

## New field type: *variable-length bit array*

- Represents **little-endian base 128** (LEB128) data fields (used by DWARF and Google's protocol buffers)
- Smallest bit array data field can be encoded on a single byte; larger bit arrays use as many bytes as needed
- Saves data stream space in situations where a data field is usually small, but can sometimes be large
- Parent field type of the variable-length *integer* field type

# What's new in CTF 2?

## CTF 2 metadata stream

```
// ...  
{  
  "field-type": "varbitarray"  
}  
// ...
```

## CTF data stream

```
...7d ee 9c b8 8b 99 d1  
89 e5 8e 26 02 00 00 ab  
17 00 00 00 2d ff 00...
```

New field type:  
*variable-length bit array*

0xe5	0x8e	0x26
11100101	10001110	00100110
00100110	10001110	11100101
0100110	0001110	1100101

Encoded field's value:

- **0x98765**
- **0100110 0001110 1100101**

# What's new in CTF 2?

## New field type: *union*

- Represents data fields which can be decoded as different field types
- Chosen mechanism to **add field types** in future minor revisions of the CTF 2 specification: old consumers can always read/skip a data field
- Example: union of sequence of bytes (in CTF 2 spec.) *and* UTF-16 string (not in CTF 2 spec.)

# What's new in CTF 2?

## CTF 2 metadata stream

```
// ...
{
  "field-type": "union",
  "alignment": 16,
  "fields": [
    {"name": "as uint32",
     "field-type": "uint32"},
    {"name": "as string",
     "field-type": {
       "field-type": "string"
     }}
  ]
}
// ...
```

## New field type: *union*

## CTF data stream

```
... 7d ee 9c b8 8b 99 d1
89 e5 43 54 46 00 ed ab
17 00 00 00 2d ff 00...
```

Encoded field's value:

- **4609091** ("as uint32")
- **CTF** ("as string")

# Potential future extensions

- UTF-16 and UTF-32 string fields
- Fixed-point number fields
- Frequency scaling support
- Standard format string user attribute (most probably inspired by Python's *format()* string method)

# Planned adoption

- **Babeltrace (consumer and producer):** v2.1
- **LTTng:** ~v2.10/v2.11 if the discussion is active enough.
  - *Condition:* Babeltrace v2.1 *must* be released.
  - *Idea:* Implement a temporary hybrid mode where you can choose to generate either a CTF 1 or a CTF 2 trace. No interest so far.
- **barectf:** As soon as Babeltrace v2.1 is released.
- **Trace Compass:** Synchronized with LTTng producing CTF 2 traces.

# CTF 2 resources

- **Proposal:**

- <https://lists.linuxfoundation.org/pipermail/diamon-discuss/2016-October/000099.html>

- **HTML version:**

- <http://diamon.org/ctf/files/CTF2-PROP-1.0.html>

- **Other documents:**

- <http://diamon.org/ctf/files/CTF2-BASICATTRS-1.0.html>

- <http://diamon.org/ctf/files/CTF2-DOCID-1.0.html>

- <http://diamon.org/ctf/files/CTF2-FS-1.0.html>

- <http://diamon.org/ctf/files/CTF2-PMETA-1.0.html>

# Questions?

