Building on top of Scapy:
what could possibly go wrong?

Julien Bedel & Claire Vacherot

Pass The SALT 2022

whoami

Claire Vacherot

Senior pentester @ Orange Cyberdefense Lyon

- Industrial networks and devices security
- I like to write tools
- Speaker @ GreHack 2020, Defcon 2021

Best conference (with PTS)

TL;DR

Tried to fit Scapy

into our existing tool

Hard time, learned a lot

Previously…

Using or not using Scapy

Tricks, workarounds and headaches

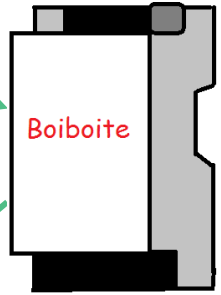
Wrap up

BOF (Boiboite Opener Framework)

Python library to discover, interact & test via several industrial network protocols

```python
from bof.layers.chicken import *

chickennet = ChickenNet().connect("192.168.1.242")
hello_req = ChickenPacket(type="hello request")
response, source = chickennet.sr(hello_req)
print("Remote IP:", response.ip_address)
chickennet.disconnect()
```

```
Remote IP: 192.168.1.242
```

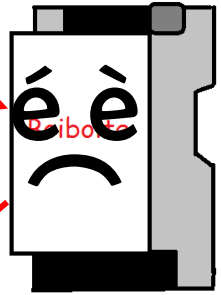


BOF (Boiboite Opener Framework)

Expected usage: misusing protocols, fuzzing

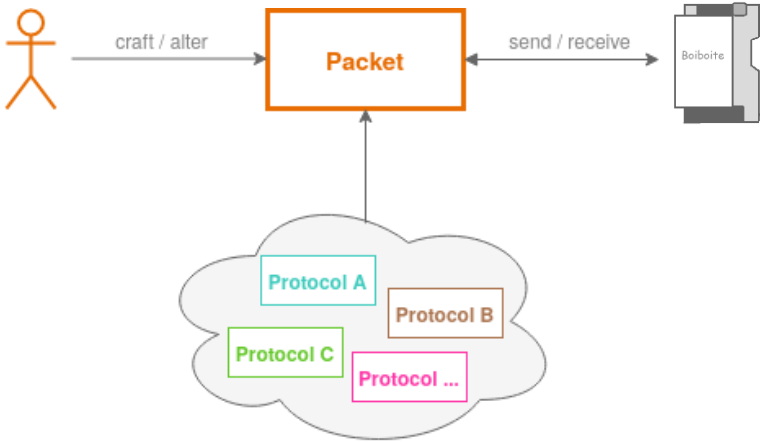
```
from bof.layers.chicken import *

chickennet = ChickenNet().connect("192.168.1.242")
hello_req = ChickenPacket(type="hello request")
hello_req.source_ip = "nope"
response, source = chickennet.sr(hello_req)
print("Remote IP:", response.ip_address)
chickennet.disconnect()
```

```
???
```



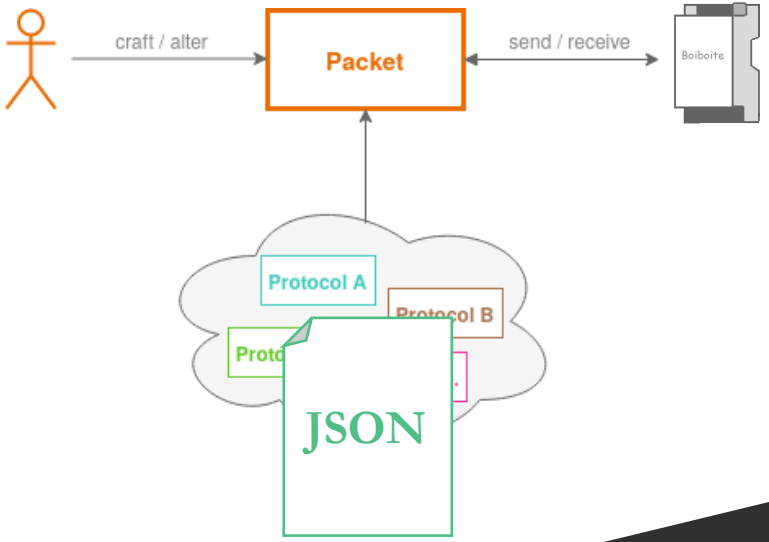
Design: First try

Requirements: Add protocols, alter packets, deviate from protocol specifications



Design: First try

Requirements: Add protocols, alter packets, deviate from protocol specifications



Design: First try

Requirements: Add protocols, alter packets, deviate from protocol specifications


```
"CRI": [
        {"name": "structure length", "type": "field", "size": 1, "is_length": true},
        {"name": "cri connection type code", "type": "field", "size": 1, "default": "03"},
        {"name": "connection data", "type": "depends:cri connection type code"}
    ],
```

Yes but…

- Field size in bytes

```
"CRI": [
        {"name": "structure length", "type": "field", "size": 1, "is_length": true},
        {"name": "cri connection type code", "type": "field", "size": 1, "default": "03"},
        {"name": "connection data", "type": "depends:cri connection type code"}
    ],
```

- Dirty workaround

```
{"name": "address type, hop count, extended frame format", "type": "field", "size": 1,
"bitsizes": "1, 3, 4"},
```

Yes but…

- Conditional fields

```
"CRI": [
        {"name": "structure length", "type": "field", "size": 1, "is_length": true},
        {"name": "cri connection type code", "type": "field", "size": 1, "default": "03"},
        {"name": "connection data", "type": "depends:cri connection type code"}
    ],
```

- Length fields to adapt
- Nested "depends"
- **A** depending on **B**, **B** depending on **C**, **C** depending on **A**

And also…

- Varying number of fields

```
... "repeat": true}
```

- Optional fields

```
... "optional": true}
```

- Fields with unpredictable sizes

```
???
```

- Type management (strings, integers, arrays, …)

```
??????????
```

And also…

https://github.com/Orange-Cyberdefense/bof/issues/13

JulienBedel commented on 14 Aug 2020 • edited

TL;DR

OPC UA sometimes receives arrays of strings.

Array data handling has not been thought and purposely implemented in BOF yet. However workarounds might exists to support array with BOF current functionalities.

- …predictable sizes ???
- Type management (strings, integers, arrays, …) ??????????

And also…

https://github.com/Orange-Cyberdefense/bof/issues/13

https://github.com/Orange-Cyberdefense/bof/issues/12

JulienBedel commented on 12 Aug 2020 • edited

TL;DR

In OPC UA, optional field presence is sometimes defined using an encoding mask. In that case, each digit of the mask represents if an optional field is present (1) or not (0).

This mechanism is not included in BOF yet, making it hard to represent frames with encoding mask based optional fields.

- Type management (strings, integers, arrays, …)

??????????

And also…

- Type managem

And also..

- Varying number
- Optional fields
- Fields with unp
- Type managem

Previously…

Using or not using Scapy

Tricks, workarounds and headaches

Wrap up

Scapy

Powerful interactive packet manipulation program

- Send, sniff and **dissect and forge** network packets
- Packets as **layers** that are stacked one upon another


```
>>> pkt = IP(dst="192.168.1.242")/TCP(dport=1664)/Chicken(sound="cluck cluck")
>>> pkt.show2()
###[ IP ]###
[...]
###[ TCP ]###
[...]
###[ Chicken ]###
 length    = 5
  type      = Bresse
  sound     = 'cluck cluck'
```

Layers

- Protocol implementations as **layers**
- A lot of existing ones, easy* to add new ones

** "If […] the protocol is not too brain-damaged […]"*

```
class Chicken(Packet):
    name = "Chicken"
    fields_desc = [
        ByteField("length", None),
        IntEnumField("type", 1, {1: "Bresse", 2: "Berry", 3: "Bastard"}),
        StrField("sound", "")
    ]
    def post_build(self, p, pay):
        p = (len(p)).to_bytes(1, byteorder='big') + p[1:]
        return p + pay
```

So close but so far away

Scapy has

Mandatory field types

Protocol specification reliance

Simple usage and clear syntax

BOF <u>also</u> needs

…optional field types

…not to rely on them

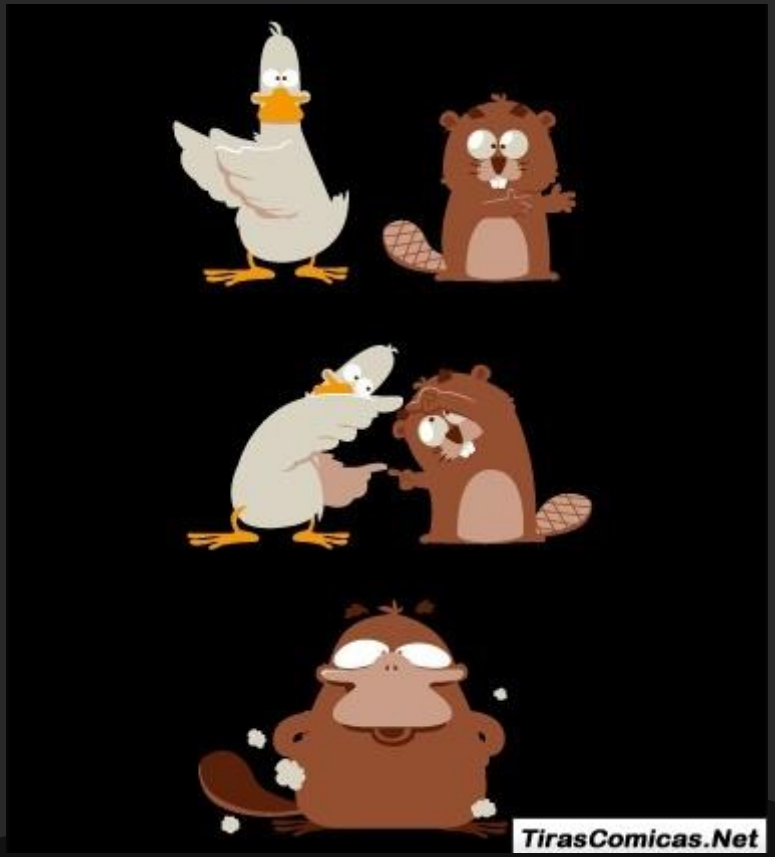
…a dedicated syntax

Thought about it a lot…

Keep BOF?

Keep Scapy?

Keep both?



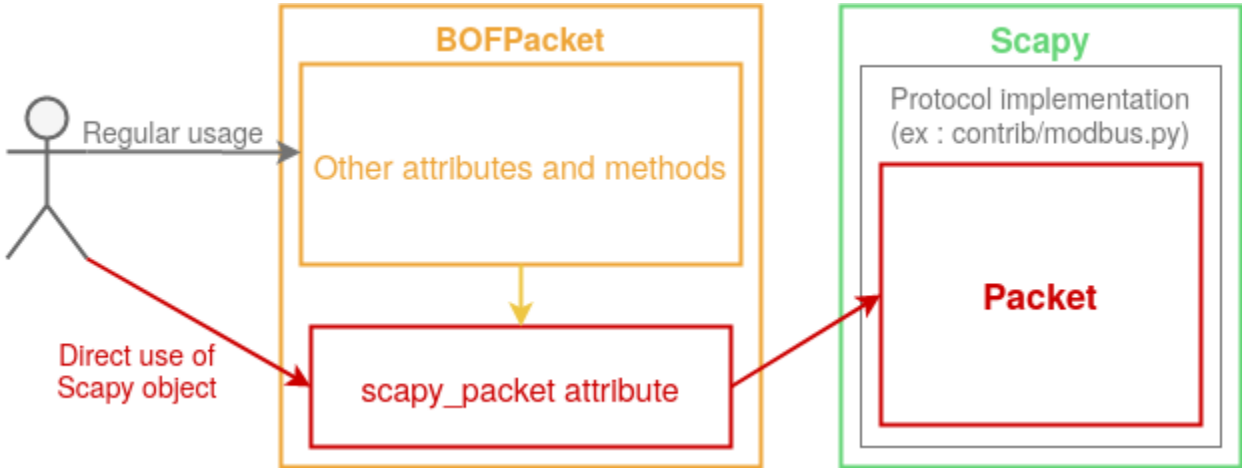
Thought about it a lot…

… and ended up writing a wrapper \o/

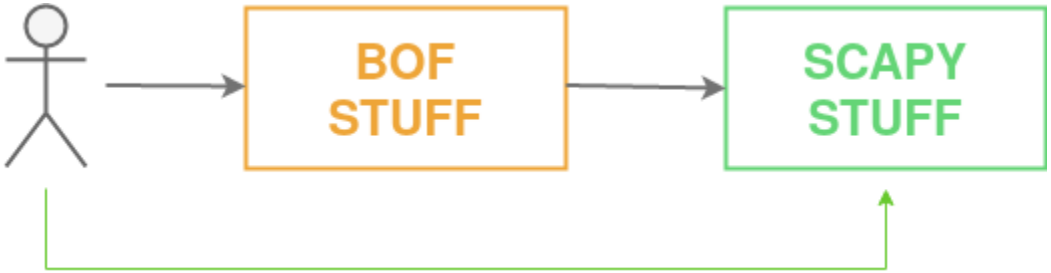
- Keep BOF usage / syntax
- Make use of Scapy's strength
- Without altering Scapy itself

To support updates

Result


BOFPacket
Scapy
Regular usage
Other attributes and methods
Protocol implementation
(ex : contrib/modbus.py)
Packet
Direct use of
Scapy object
scapy_packet attribute

Result



Previously…

Using or not using Scapy

Tricks, workarounds and headaches

Wrap up

BOF's behavior

```
pkt = bof.ChickenPacket()
pkt.type = 1
pkt.sound = "whatever"
print(raw(pkt))
pkt.show2()
```

```
b'\r\x00\x00\x00\x01whatever'
###[ Chicken ]###
  length    = 5
  type      = Bresse
  sound     = 'whatever'
```

BOF's behavior

Not saying that doing this makes sense

```
pkt = bof.ChickenPacket()
pkt.type = "yeah"
pkt.sound = "whatever"
print(raw(pkt))
pkt.show2()
```

```
b'\ryeahwhatever'
###[ Chicken ]###
  length    = 13
  type      = b'yeah'
  sound     = 'whatever'
```

Scapy's behavior

```
pkt = scapy.Chicken()
pkt.type = "yeah"
pkt.sound = "whatever"
print(raw(pkt))
pkt.show2()
```

```
ValueError: Incorrect type of value for field type:
struct.error('required argument is not an integer')
To inject bytes into the field regardless of the type, use RawVal. See help(RawVal)
```

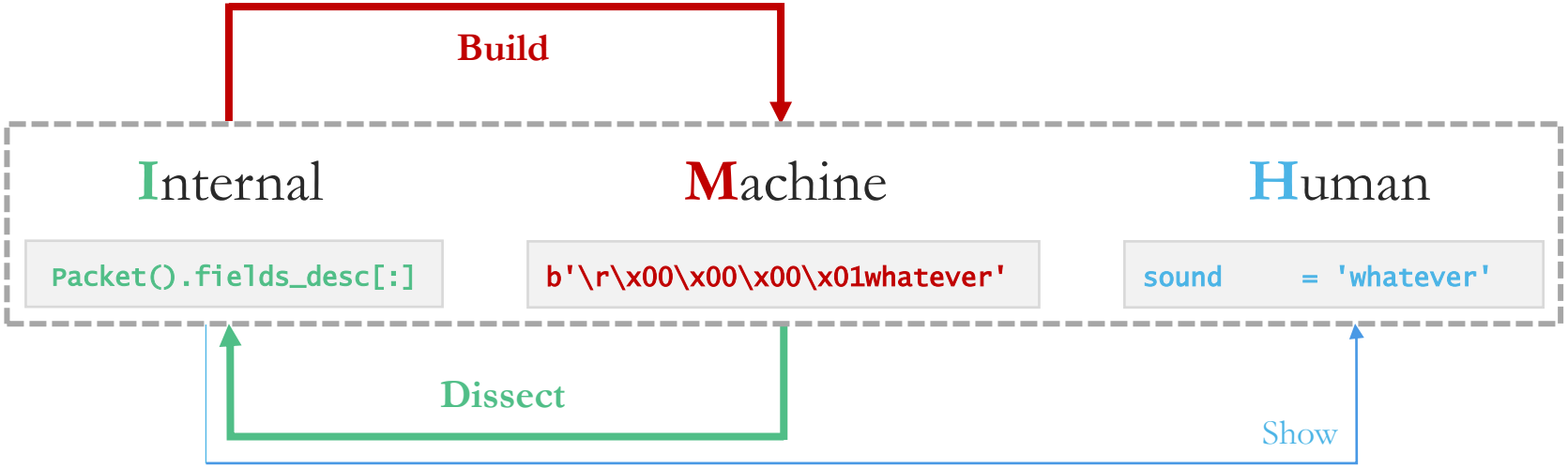
Scapy's behavior

- Good point but **RawVal** has fewer features (from scapy/fields.py):

```
class RawVal:
    def __init__(self, val=b""):
    def __str__(self):
    def __bytes__(self):
    def __len__(self):
    def __repr__(self):
```

```
class Field(Generic[I, M]):
__slots__ = ["name", "fmt", "default", "sz",
"owners", "struct"]
   def h2i(self, pkt, x):
    def i2h(self, pkt, x):
    def m2i(self, pkt, x):
    def i2m(self, pkt, x):
    def any2i(self, pkt, x):
    def i2repr(self, pkt, x):
    def addfield(self, pkt, s, val):
    def getfield(self, pkt, s):
    def copy(self):
    def randval(self):
    [...]
```

Quick reminder



https://scapy.readthedocs.io/en/latest/build_dissect.html

Scapy vs. BOF

Scapy has

Mandatory field types

Protocol specification reliance

I

Fields always calculated from the packet

This is probably why Scapy works so well, duh

BOF <u>also</u> needs

…optional field types

…not to rely on them

M

Fields sometimes disconnected from the packet

Scapy vs. BOF

Why not just change Machine representation?

Disconnected from Internal, breaks Human…

- Loose Scapy's capabilities

Scapy vs. BOF

Why not just change Machine representation?

Disconnected from Internal, breaks Human…

- Loose Scapy's capabilities

Let's mess with internals \o/

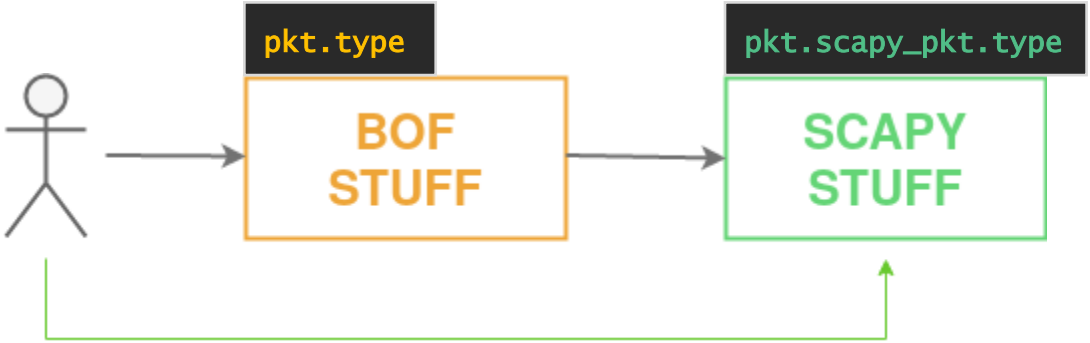
"It is a strange fate that we should suffer so much fear and doubt over so small a thing."

```
pkt = bof.ChickenPacket()
pkt.type = "yeah"
pkt.sound = "whatever"
print(raw(pkt))
pkt.show2()
```

```
b'\ryeahwhatever'
###[ Chicken ]###
  length    = 13
  type      = b'yeah'
  sound     = 'whatever'
```

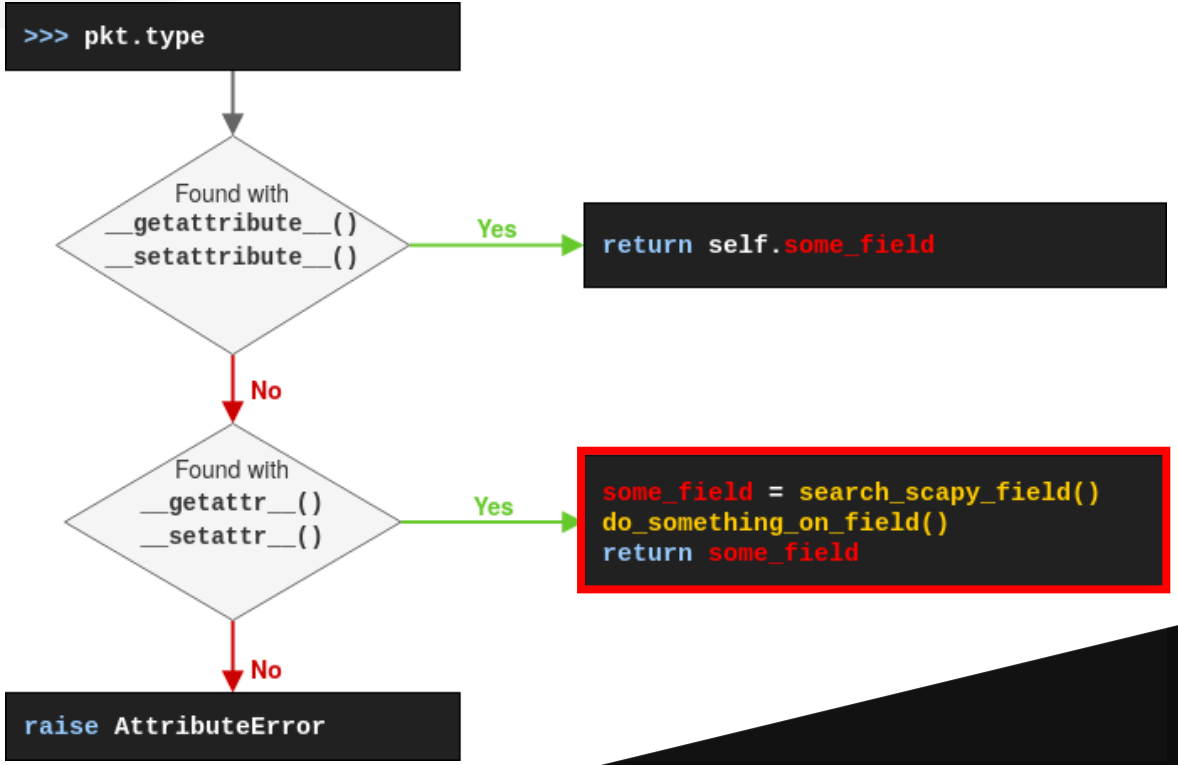
Interface with Scapy

```
pkt.type = "yeah"
```



Python internals Call to pkt's __getattr__() / __setattr__() method

Interface with Scapy



Not as straightforward

```
some_field = search_scapy_field()
```

Loop through fields for `some_field`

```
do_something_on_field()
```

`value` compatible with field?

- Yes : `somefield = value`
- No : ***dramatic music***

https://github.com/Orange-Cyberdefense/bof/blob/master/bof/packet.py

```python
    def _field_generator(self, start_packet:object=None, terminal=False) -> tuple:
        """Yields fields in packet/packetfields with their closest parent.

        This is where the worst of Scapy comes to life (and is translated to BOF).
        Brace yourselves, and welcome to hell.
        """
        start_packet = self.scapy_pkt if not start_packet else start_packet
        iterlist = [start_packet] if isinstance(start_packet, PacketField) else \
                   [start_packet, start_packet.payload]
        for packet in iterlist:
            for field in packet.fields_desc:
                if isinstance(field, MultipleTypeField):
                    field = field._find_fld()
                elif isinstance(field, ConditionalField) and field._evalcond(packet):
                    field = field.fld
                if isinstance(field, PacketField) or isinstance(field, Packet):
                    pkt = getattr(packet, field.name)
                    # if pkt = None, next call restarts at start_packet (1st line)
                    # and causes infinite loop, so we replace with empty packet.
                    yield from self._field_generator(pkt if pkt else Packet())
                if isinstance(field, Field):
                    # We must handle a tricky situation: Sometimes part of a packet
                    # is in payload, and Scapy will search for them only in identified
                    # fields. So if we don't find the field in the frame, we return
                    # the payload instead.
                    try:
                        start_packet.getfield_and_val(field.name)
                    except ValueError:
                        yield field, start_packet.payload # Not in packet, give payload
                    else:
                        yield field, start_packet # Found the packet
```

Dynamically changing types

```
pkt.type = "yeah"
```

```
fields_desc = [
        ByteField("length", None),
        IntEnumField("type", 1, {1: "Bresse", 2: "Berry", 3: "Bastard"}),
        StrField("sound", "")
```

1. Loop through fields for `pkt.type`
2. `value` compatible with field?
   - ~~Yes : `pkt.type = value`~~
   - No : **Replace `fields_desc[1]` with `ByteField("yeah")` ???**

Yes but…

- `fields_desc` as class attribute

```
class Sandwich(Packet):
fields_desc = [
        "Salad",
        "Tomato",
        "Onion"
    ]

first, second = Sandwich(), Sandwich()
second.fields_desc[1] = "Camembert"
```

```
first:   Salad, Camembert, Onion
second:  Salad, Camembert, Onion
```

Dynamically changing types

1. `Clone` Scapy Packet object (!= `copy`)
2. Replace `fields_desc[1]` with `ByteField("yeah")` in new class

Chicken

```
ByteField("length"),
IntEnumField("type"),
StrField("sound")
```

Chicken_<randint>

```
ByteField("length"),
ByteField("type"),
StrField("sound")
```

`self._clone()`

~~It's useless but~~ it works!

```
pkt = ChickenPacket()
pkt.type = "yeah"
pkt.sound = "whatever"
print(raw(pkt))
pkt.show2()
```

```
b'\ryeahwhatever'
###[ Chicken ]###
  length    = 13
  type      = b'yeah'
  sound     = 'whatever'
```

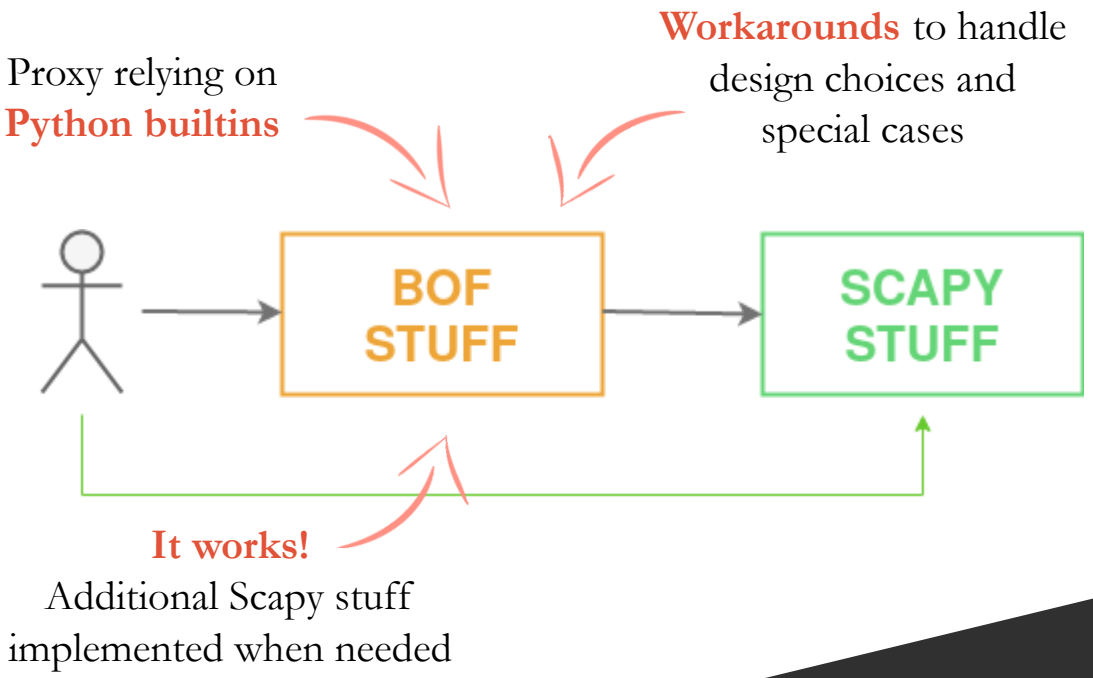
Others things that we can do

Not saying that doing this makes sense

- Add, remove, resize fields in packets
- More ways to access and update fields
- Proxy with additional attributes, methods and properties

```
>>> pkt.ip_address
'192.168.1.1'
>>> pkt["ip_address"]
b'\xc0\xa8\x01\x01'
```

From BOF to Scapy



Previously…

Using or not using Scapy

Tricks, workarounds and headaches

Wrap up

Wrap up

Use of previous work

and taking time for design

may have saved us time

Wrap up

Don’t just use the tools

understand their true power

make the most of it

(== RTFM because Scapy = ❤️ and Python = ❤️)

Thank you
Enjoy PTS!