scapy.layers.inet

IPv4 (Internet Protocol v4).

exception **scapy.layers.inet.BadFragments**(**args*, ***kwargs*) [source]

Bases: `ValueError`

class **scapy.layers.inet.DestIPField**(*name*, *default*) [source]

Bases: `IPField`, `DestField`

bindings: *Dict[Type[Packet], Tuple[str, Any]]*
= {<class 'scapy.contrib.ospf.OSPF_Hdr'>: [('224.0.0.5', {})], <class 'scapy.layers.inet.UDP'>: [('224.0.0.251', {'dport': 5353}), ('224.0.0.2', {'dport': 1985})]}

i2h(*pkt*, *x*) [source]

i2m(*pkt*, *x*) [source]

class **scapy.layers.inet.ICMP**(*_pkt*, */*, ***, *type=8*, *code=0*, *chksum=None*, *id=0*, *seq=0*, *ts_ori=61209461*, *ts_rx=61209461*, *ts_tx=61209461*, *gw='0.0.0.0'*, *ptr=0*, *reserved=0*, *length=0*, *addr_mask='0.0.0.0'*, *nexthopmtu=0*, *unused=None*, *extpad=b''*, *ext=None*) [source]

Bases: `Packet`

aliastypes

answers(*other*) [source]

fields_desc

▶ Display RFC-like schema

ICMP fields

type	`ByteEnumField`	`8`
code	`MultiEnumField` (Depends on 8)	`0`
chksum	`XShortField`	None
id	`XShortField` (Cond)	`0`
seq	`XShortField` (Cond)	`0`

ts_ori	`ICMPTimeStampField` (Cond)	`61209461`
ts_rx	`ICMPTimeStampField` (Cond)	`61209461`
ts_tx	`ICMPTimeStampField` (Cond)	`61209461`
gw	`IPField` (Cond)	`'0.0.0.0'`
ptr	`ByteField` (Cond)	`0`
reserved	`ByteField` (Cond)	`0`
length	`ByteField` (Cond)	`0`
addr_mask	`IPField` (Cond)	`'0.0.0.0'`
nexthopmtu	`ShortField` (Cond)	`0`
unused	`MultipleTypeField` (ShortField, IntField, StrFixedLenField)	`b''`
extpad	`_ICMPExtensionPadField` (Cond)	`b''`
ext	`_ICMPExtensionField` (Cond)	`None`

guess_payload_class(*payload*) [source]

hashret() [source]

mysummary() [source]

post_build(*p*, *pay*) [source]

post_dissection(*pkt*) [source]

class **scapy.layers.inet.ICMPEcho_am**(*self*) [source]

Bases: `AnsweringMachine`

Responds to ICMP Echo-Requests (ping)

function_name = *'icmpechod'*

is_request(*req*) [source]

make_reply(*req*) [source]

optam0: *Dict[str, Any]*

optam1: *Dict[str, Any]*

optam2: *Dict[str, Any]*

optsend: *Dict[str, Any]*

optsniff: *Dict[str, Any]*

print_reply(*req*, *reply*) [source]

---

class **scapy.layers.inet.ICMPExtension_Header**(*_pkt*, */*, ***, *version=2*, *reserved=0*, *chksum=None*) [source]

Bases: `Packet`

ICMP Extension per RFC4884.

Example:

```
pkt = IP(dst="127.0.0.1", src="127.0.0.1") / ICMP(
    type="time-exceeded",
    code="ttl-zero-during-transit",
    ext=ICMPExtension_Header() / ICMPExtension_InterfaceInformation(
        has_ifindex=1,
        has_ipaddr=1,
        has_ifname=1,
        ip4="10.10.10.10",
        ifname="hey",
    )
) / IPerror(src="12.4.4.4", dst="12.1.1.1") / \
    UDPerror(sport=42315, dport=33440) /  \
    b'\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00'
```

aliastypes

fields_desc

▶ Display RFC-like schema

ICMPExtension_Header fields

version	`BitField` (4 bits)	`2`
reserved	`BitField` (12 bits)	`0`
chksum	`XShortField`	None

guess_payload_class(*payload*) [source]

post_build(*p*, *pay*) [source]

show_indent= *0*

---

class **scapy.layers.inet.ICMPExtension_InterfaceInformation**(*_pkt*, */*, ***, *len=None*, *classnum=2*, *classtype=0*, *reserved=0*, *has_ifindex=0*, *has_ipaddr=0*, *has_ifname=0*, *has_mtu=0*, *ifindex=None*, *afi=None*, *reserved2=0*, *ip4=None*, *ip6=None*, *ifname_len=None*, *ifname=None*, *mtu=None*) [source]

Bases: `ICMPExtension_Object`

aliastypes

fields_desc

▶ Display RFC-like schema

ICMPExtension_InterfaceInformation fields

len	`ShortField`	None
classnum	`ByteEnumField`	2
classtype	`BitField` (2 bits)	0
reserved	`BitField` (2 bits)	0
has_ifindex	`BitField` (1 bit)	0
has_ipaddr	`BitField` (1 bit)	0
has_ifname	`BitField` (1 bit)	0
has_mtu	`BitField` (1 bit)	0
ifindex	`IntField` (Cond)	None
afi	`ShortField` (Cond)	None
reserved2	`ShortField` (Cond)	0
ip4	`IPField` (Cond)	None
ip6	`IP6Field` (Cond)	None
ifname_len	`FieldLenField` (Cond)	None
ifname	`StrLenField` (Cond)	None
mtu	`IntField` (Cond)	None

self_build(***kwargs*) [source]

---

class **scapy.layers.inet.ICMPExtension_Object**(*_pkt*, */*, ***, *len=None*, *classnum=0*, *classtype=0*) [source]

Bases: `Packet`

aliastypes

classmethod **dispatch_hook**(*_pkt=None*, **args*, ***kargs*) [source]

fields_desc

▶ Display RFC-like schema

ICMPExtension_Object fields

len	`ShortField`	None
classnum	`ByteEnumField`	0
classtype	`ByteField`	0

post_build(*p*, *pay*) [source]

classmethod **register_variant**() [source]

registered_icmp_exts
= {0: <class 'scapy.layers.inet.ICMPExtension_Object'>, 1: <class 'scapy.contrib.mpls.ICMPExtension_MPLS'>, 2: <class 'scapy.layers.inet.ICMPExtension_InterfaceInformation'>}

show_indent= *0*

---

class **scapy.layers.inet.ICMPTimeStampField**(*name: str*, *default: int | None*) [source]

Bases: `IntField`

any2i(*pkt*, *val*) [source]

i2repr(*pkt*, *val*) [source]

re_hmsm= *re.compile('([0-2]?[0-9])[Hh:](([0-5]?[0-9])([Mm:]([0-5]?[0-9])([sS:.]([0-9]{0,3}))?)?)?$')*

---

class **scapy.layers.inet.ICMPerror**(*_pkt*, */*, ***, *type=8*, *code=0*, *chksum=None*, *id=0*, *seq=0*, *ts_ori=61209461*, *ts_rx=61209461*, *ts_tx=61209461*, *gw='0.0.0.0'*, *ptr=0*, *reserved=0*, *length=0*, *addr_mask='0.0.0.0'*, *nexthopmtu=0*, *unused=None*, *extpad=b''*, *ext=None*) [source]

Bases: `ICMP`

aliastypes

answers(*other*) [source]

fields_desc

▶ Display RFC-like schema

ICMPerror fields

type	`ByteEnumField`	`8`
code	`MultiEnumField` (Depends on 8)	`0`
chksum	`XShortField`	`None`
id	`XShortField` (Cond)	`0`
seq	`XShortField` (Cond)	`0`
ts_ori	`ICMPTimeStampField` (Cond)	`61209461`
ts_rx	`ICMPTimeStampField` (Cond)	`61209461`
ts_tx	`ICMPTimeStampField` (Cond)	`61209461`
gw	`IPField` (Cond)	`'0.0.0.0'`
ptr	`ByteField` (Cond)	`0`
reserved	`ByteField` (Cond)	`0`
length	`ByteField` (Cond)	`0`
addr_mask	`IPField` (Cond)	`'0.0.0.0'`
nexthopmtu	`ShortField` (Cond)	`0`
unused	`MultipleTypeField` (ShortField, IntField, StrFixedLenField)	`b''`
extpad	`_ICMPExtensionPadField` (Cond)	`b''`
ext	`_ICMPExtensionField` (Cond)	`None`

mysummary() [source]

---

class **scapy.layers.inet.IP**(*_pkt, /, *, version=4, ihl=None, tos=0, len=None, id=1, flags=<Flag 0 ()>, frag=0, ttl=64, proto=0, chksum=None, src=None, dst=None, options=[]*) [source]

Bases: `Packet`, `IPTools`

aliastypes

answers(*other*) [source]

extract_padding(*s*) [source]

fields_desc

▶ Display RFC-like schema

IP fields

version	`BitField` (4 bits)	`4`
ihl	`BitField` (4 bits)	`None`
tos	`XByteField`	`0`
len	`ShortField`	`None`
id	`ShortField`	`1`
flags	`FlagsField`	`<Flag 0 ()>`
frag	`BitField` (13 bits)	`0`
ttl	`ByteField`	`64`
proto	`ByteEnumField`	`0`
chksum	`XShortField`	`None`
src	`SourceIPField`	`None`
dst	`DestIPField`	`None`
options	`PacketListField`	`[]`

`fragment`(*fragsize=1480*) [source]

Fragment IP datagrams

`hashret`() [source]

`mysummary`() [source]

`payload_guess`

Possible sublayers: `CARP` , `EIGRP` , `EtherIP` , `IGMP` , `IGMPv3` , `MPLS` , `OSPF_Hdr` , `PIMv2Hdr` , `RSVP` , `ICMP` , `IP` , `TCP` , `UDP` , `IPv6` , `AH` , `ESP` , `GRE` , `SCTP`

`post_build`(*p*, *pay*) [source]

`route`() [source]

`scapy.layers.inet.IPID_count`(*lst*, *funcID=<function <lambda>>*, *funcpres=<function <lambda>>*) [source]

Identify IP id values classes in a list of packets

lst: a list of packets funcID: a function that returns IP id values funcpres: a function used to summarize packets

class **scapy.layers.inet.IPOption**(*_pkt*, */*, ***, *copy_flag=0*, *optclass=0*, *option=0*, *length=None*, *value=b''*) [source]

Bases: `Packet`

aliastypes

classmethod **dispatch_hook**(*pkt=None*, **args*, ***kargs*) [source]

extract_padding(*p*) [source]

fields_desc

▶ Display RFC-like schema

IPOption fields

copy_flag	`BitField` (1 bit)	`0`
optclass	`BitEnumField`	`0`
option	`BitEnumField`	`0`
length	`FieldLenField`	None
value	`StrLenField`	`b''`

classmethod **register_variant**() [source]

registered_ip_options
= {0: <class 'scapy.layers.inet.IPOption_EOL'>, 1: <class 'scapy.layers.inet.IPOption_NOP'>, 2: <class 'scapy.layers.inet.IPOption_Security'>, 3: <class 'scapy.layers.inet.IPOption_LSRR'>, 4: <class 'scapy.layers.inet.IPOption_Timestamp'>, 7: <class 'scapy.layers.inet.IPOption_RR'>, 8: <class 'scapy.layers.inet.IPOption_Stream_Id'>, 9: <class 'scapy.layers.inet.IPOption_SSRR'>, 11: <class 'scapy.layers.inet.IPOption_MTU_Probe'>, 12: <class 'scapy.layers.inet.IPOption_MTU_Reply'>, 18: <class 'scapy.layers.inet.IPOption_Traceroute'>, 19: <class 'scapy.layers.inet.IPOption_Address_Extension'>, 20: <class 'scapy.layers.inet.IPOption_Router_Alert'>, 21: <class 'scapy.layers.inet.IPOption_SDBM'>}

class **scapy.layers.inet.IPOption_Address_Extension**(*_pkt*, */*, ***, *copy_flag=1*, *optclass=0*, *option=19*, *length=10*, *src_ext='0.0.0.0'*, *dst_ext='0.0.0.0'*) [source]

Bases: `IPOption`

aliastypes

fields_desc

▶ Display RFC-like schema

IPOption_Address_Extension fields

copy_flag	BitField (1 bit)	1
optclass	BitEnumField	0
option	BitEnumField	19
length	ByteField	10
src_ext	IPField	'0.0.0.0'
dst_ext	IPField	'0.0.0.0'

class **scapy.layers.inet.IPOption_EOL**(*_pkt*, */*, ***, *copy_flag=0*, *optclass=0*, *option=0*) [source]

Bases: IPOption

aliastypes

fields_desc

▶ Display RFC-like schema

IPOption_EOL fields

copy_flag	BitField (1 bit)	0
optclass	BitEnumField	0
option	BitEnumField	0

class **scapy.layers.inet.IPOption_LSRR**(*_pkt*, */*, ***, *copy_flag=1*, *optclass=0*, *option=3*, *length=None*, *pointer=4*, *routers=[]*) [source]

Bases: IPOption_RR

aliastypes

fields_desc

▶ Display RFC-like schema

IPOption_LSRR fields

copy_flag	BitField (1 bit)	1
optclass	BitEnumField	0
option	BitEnumField	3
length	FieldLenField	None
pointer	ByteField	4

routers	FieldListField	[]

class **scapy.layers.inet.IPOption_MTU_Probe**(*_pkt*, */*, ***, *copy_flag=0*, *optclass=0*, *option=11*, *length=4*, *mtu=0*) [source]

Bases: IPOption

aliastypes

fields_desc

▶ Display RFC-like schema

IPOption_MTU_Probe fields

copy_flag	BitField (1 bit)	0
optclass	BitEnumField	0
option	BitEnumField	11
length	ByteField	4
mtu	ShortField	0

class **scapy.layers.inet.IPOption_MTU_Reply**(*_pkt*, */*, ***, *copy_flag=0*, *optclass=0*, *option=12*, *length=4*, *mtu=0*) [source]

Bases: IPOption_MTU_Probe

aliastypes

fields_desc

▶ Display RFC-like schema

IPOption_MTU_Reply fields

copy_flag	BitField (1 bit)	0
optclass	BitEnumField	0
option	BitEnumField	12
length	ByteField	4
mtu	ShortField	0

class **scapy.layers.inet.IPOption_NOP**(*_pkt*, */*, ***, *copy_flag=0*, *optclass=0*, *option=1*) [source]

Bases: IPOption

aliastypes

fields_desc

▶ Display RFC-like schema

IPOption_NOP fields

copy_flag	`BitField` (1 bit)	`0`
optclass	`BitEnumField`	`0`
option	`BitEnumField`	`1`

---

class **scapy.layers.inet.IPOption_RR**(*_pkt*, */*, ***, *copy_flag=0*, *optclass=0*, *option=7*, *length=None*, *pointer=4*, *routers=[]*) [source]

Bases: `IPOption`

aliastypes

fields_desc

▶ Display RFC-like schema

IPOption_RR fields

copy_flag	`BitField` (1 bit)	`0`
optclass	`BitEnumField`	`0`
option	`BitEnumField`	`7`
length	`FieldLenField`	None
pointer	`ByteField`	`4`
routers	`FieldListField`	`[]`

get_current_router() [source]

---

class **scapy.layers.inet.IPOption_Router_Alert**(*_pkt*, */*, ***, *copy_flag=1*, *optclass=0*, *option=20*, *length=4*, *alert=0*) [source]

Bases: `IPOption`

aliastypes

fields_desc

▶ Display RFC-like schema

IPOption_Router_Alert fields

copy_flag	`BitField` (1 bit)	`1`
optclass	`BitEnumField`	`0`

option	BitEnumField	20
length	ByteField	4
alert	ShortEnumField	0

class **scapy.layers.inet.IPOption_SDBM**(*_pkt*, */*, ***, *copy_flag=1*, *optclass=0*, *option=21*, *length=None*, *addresses=[]*) [source]

Bases: IPOption

aliastypes

fields_desc

▶ Display RFC-like schema

IPOption_SDBM fields

copy_flag	BitField (1 bit)	1
optclass	BitEnumField	0
option	BitEnumField	21
length	FieldLenField	None
addresses	FieldListField	[]

class **scapy.layers.inet.IPOption_SSRR**(*_pkt*, */*, ***, *copy_flag=1*, *optclass=0*, *option=9*, *length=None*, *pointer=4*, *routers=[]*) [source]

Bases: IPOption_RR

aliastypes

fields_desc

▶ Display RFC-like schema

IPOption_SSRR fields

copy_flag	BitField (1 bit)	1
optclass	BitEnumField	0
option	BitEnumField	9
length	FieldLenField	None
pointer	ByteField	4
routers	FieldListField	[]

class **scapy.layers.inet.IPOption_Security**(*_pkt*, */*, ***, *copy_flag=1*, *optclass=0*, *option=2*, *length=11*, *security=0*, *compartment=0*, *handling_restrictions=0*, *transmission_control_code=b'xxx'*) [source]

Bases: `IPOption`

aliastypes

fields_desc

▶ Display RFC-like schema

IPOption_Security fields

copy_flag	`BitField` (1 bit)	`1`
optclass	`BitEnumField`	`0`
option	`BitEnumField`	`2`
length	`ByteField`	`11`
security	`ShortField`	`0`
compartment	`ShortField`	`0`
handling_restrictions	`ShortField`	`0`
transmission_control_code	`StrFixedLenField`	`b'xxx'`

class **scapy.layers.inet.IPOption_Stream_Id**(*_pkt*, */*, ***, *copy_flag=1*, *optclass=0*, *option=8*, *length=4*, *security=0*) [source]

Bases: `IPOption`

aliastypes

fields_desc

▶ Display RFC-like schema

IPOption_Stream_Id fields

copy_flag	`BitField` (1 bit)	`1`
optclass	`BitEnumField`	`0`
option	`BitEnumField`	`8`
length	`ByteField`	`4`
security	`ShortField`	`0`

class **scapy.layers.inet.IPOption_Timestamp**(*_pkt*, */*, ***, *copy_flag=0*, *optclass=2*, *option=4*, *length=None*, *pointer=9*, *oflw=0*, *flg=1*, *internet_address='0.0.0.0'*, *timestamp=0*) [source]

Bases: `IPOption`

aliastypes

fields_desc

▶ Display RFC-like schema

IPOption_Timestamp fields

copy_flag	`BitField` (1 bit)	`0`
optclass	`BitEnumField`	`2`
option	`BitEnumField`	`4`
length	`ByteField`	`None`
pointer	`ByteField`	`9`
oflw	`BitField` (4 bits)	`0`
flg	`BitEnumField`	`1`
internet_address	`IPField` (Cond)	`'0.0.0.0'`
timestamp	`IntField`	`0`

post_build(*p*, *pay*) [source]

---

class **scapy.layers.inet.IPOption_Traceroute**(*_pkt*, */*, ***, *copy_flag=0*, *optclass=0*, *option=18*, *length=12*, *id=0*, *outbound_hops=0*, *return_hops=0*, *originator_ip='0.0.0.0'*) [source]

Bases: `IPOption`

aliastypes

fields_desc

▶ Display RFC-like schema

IPOption_Traceroute fields

copy_flag	`BitField` (1 bit)	`0`
optclass	`BitEnumField`	`0`
option	`BitEnumField`	`18`
length	`ByteField`	`12`
id	`ShortField`	`0`
outbound_hops	`ShortField`	`0`
return_hops	`ShortField`	`0`

originator_ip	IPField	'0.0.0.0'

class **scapy.layers.inet.IPTools** [source]

Bases: object

Add more powers to a class with an “src” attribute.

hops() [source]

ottl() [source]

whois() [source]

whois the source and print the output

class **scapy.layers.inet.IPerror**(*_pkt*, */*, ***, *version=4*, *ihl=None*, *tos=0*, *len=None*, *id=1*, *flags=<Flag 0 ()>*, *frag=0*, *ttl=64*, *proto=0*, *chksum=None*, *src=None*, *dst=None*, *options=[]*) [source]

Bases: IP

aliastypes

answers(*other*) [source]

fields_desc

▶ Display RFC-like schema

IPerror fields

version	BitField (4 bits)	4
ihl	BitField (4 bits)	None
tos	XByteField	0
len	ShortField	None
id	ShortField	1
flags	FlagsField	<Flag 0 ()>
frag	BitField (13 bits)	0
ttl	ByteField	64
proto	ByteEnumField	0
chksum	XShortField	None
src	SourceIPField	None

dst	`DestIPField`	`None`
options	`PacketListField`	`[]`

mysummary() [source]

payload_guess

Possible sublayers: `ICMPerror`, `IPerror`, `TCPerror`, `UDPerror`, `SCTPerror`

class scopy.layers.inet.RandTCPOptions(*size=None*) [source]

Bases: `VolatileValue`

class scopy.layers.inet.TCP(*_pkt, /, *, sport=20, dport=80, seq=0, ack=0, dataofs=None, reserved=0, flags=<Flag 2 (S)>, window=8192, chksum=None, urgptr=0, options=b''*) [source]

Bases: `Packet`

aliastypes

answers(*other*) [source]

fields_desc

▶ Display RFC-like schema

TCP fields

sport	`ShortEnumField`	`20`
dport	`ShortEnumField`	`80`
seq	`IntField`	`0`
ack	`IntField`	`0`
dataofs	`BitField` (4 bits)	`None`
reserved	`BitField` (3 bits)	`0`
flags	`FlagsField`	`<Flag 2 (S)>`
window	`ShortField`	`8192`
chksum	`XShortField`	`None`
urgptr	`ShortField`	`0`
options	`TCPOptionsField`	`b''`

hashret() [source]

mysummary() [source]

payload_guess

Possible sublayers: `HSFZ`, `DoIP`, `SOMEIP`, `BGP`, `CRX1New`, `DiamG`, `ENIPTCP`, `LDP`, `ModbusADURequest`, `ModbusADUResponse`, `MQTT`, `OpenFlow`, `PostgresBackend`, `PostgresFrontend`, `RTR`, `RTSP`, `Skinny`, `SOCKS`, `STUN`, `TacacsHeader`, `DceRpc`, `DNS`, `HTTP`, `UDP`, `KerberosTCPHeader`, `KpasswdTCPHeader`, `LDAP`, `NBTSession`, `PPTP`, `Skinny`

post_build(*p*, *pay*) [source]

---

class **scapy.layers.inet.TCPAOValue**(*_pkt*, */*, ***, *keyid=None*, *rnextkeyid=None*, *mac=b''*) [source]

Bases: `Packet`

Value of TCP-AO option

aliastypes

fields_desc

▶ Display RFC-like schema

TCPAOValue fields

keyid	`ByteField`	None
rnextkeyid	`ByteField`	None
mac	`StrLenField`	b''

---

class **scapy.layers.inet.TCPOptionsField**(*name: str*, *default: I | None*, *fmt: str = 'H'*, *remain: int = 0*) [source]

Bases: `StrField`

getfield(*pkt*, *s*) [source]

i2h(*pkt*, *x*) [source]

i2m(*pkt*, *x*) [source]

islist= *1*

m2i(*pkt*, *x*) [source]

randval() [source]

class **scapy.layers.inet.TCP_client**(*self*, *ip*, *port*, *srcip=None*, ***kargs*) [source]

Bases: `Automaton`

Creates a TCP Client Automaton. This automaton will handle TCP 3-way handshake.

Usage: the easiest usage is to use it as a SuperSocket.

```
>>> a = TCP_client.tcplink(HTTP, "www.google.com", 80)
>>> a.send(HTTPRequest())
>>> a.recv()
```

Parameters:

- **ip** – the ip to connect to
- **port**
- **src** – (optional) use another source IP

CLOSED(**args: ATMT, **kargs: Any*)→ NewStateRequested [source]

ESTABLISHED(**args: ATMT, **kargs: Any*)→ NewStateRequested [source]

LAST_ACK(**args: ATMT, **kargs: Any*)→ NewStateRequested [source]

START(**args: ATMT, **kargs: Any*)→ NewStateRequested [source]

STOP(**args: ATMT, **kargs: Any*)→ NewStateRequested [source]

STOP_SENT_FIN_ACK(**args: ATMT, **kargs: Any*)→ NewStateRequested [source]

SYN_SENT(**args: ATMT, **kargs: Any*)→ NewStateRequested [source]

ack_of_fin_received(*pkt*) [source]

actions: *Dict[str, List[_StateWrapper]]*
= {'ack_of_fin_received': [], 'connect': [<function TCP_client.send_syn>], 'fin_received': [<function TCP_client.send_finack>], 'incoming_data_received': [<function TCP_client.receive_data>], 'outgoing_data_received': [<function TCP_client.send_data>], 'reset_received': [], 'stop_ack_timeout': [], 'stop_fin_received': [<function TCP_client.stop_send_ack>], 'stop_requested': [<function TCP_client.stop_send_finack>], 'syn_ack_timeout': [], 'synack_received': [<function TCP_client.send_ack_of_synack>]}

breakpoints: *Set[_StateWrapper]*

conditions: *Dict[str, List[_StateWrapper]]*
= {'CLOSED': [], 'ESTABLISHED': [], 'LAST_ACK': [], 'START': [<function TCP_client.connect>], 'STOP': [<function TCP_client.stop_requested>], 'STOP_SENT_FIN_ACK': [], 'SYN_SENT': []}

connect() [source]

eofs: *Dict[str, _StateWrapper] = {}*

fin_received(*pkt*) [source]

incoming_data_received(*pkt*) [source]

initial_states: *List[_StateWrapper]*
= [<function ATMT.state.<locals>.deco.<locals>._state_wrapper>]

intercepted_packet: *None | Packet*

interception_points: *Set[_StateWrapper]*

ioevents: *Dict[str, List[_StateWrapper]]*
= {'CLOSED': [], 'ESTABLISHED': [<function TCP_client.outgoing_data_received>], 'LAST_ACK': [], 'START': [], 'STOP': [], 'STOP_SENT_FIN_ACK': [], 'SYN_SENT': []}

ionames: *List[str] = ['tcp']*

iosupersockets: *List[SuperSocket] = [<function TCP_client.outgoing_data_received>]*

listen_sock: *SuperSocket | None*

master_filter(*pkt*) [source]

outgoing_data_received(*fd*) [source]

packets: *PacketList*

parse_args(*ip*, *port*, *srcip=None*, ***kargs*) [source]

receive_data(*pkt*) [source]

recv_conditions: *Dict[str, List[_StateWrapper]]*
= {'CLOSED': [], 'ESTABLISHED': [<function TCP_client.incoming_data_received>, <function TCP_client.reset_received>, <function TCP_client.fin_received>], 'LAST_ACK': [<function TCP_client.ack_of_fin_received>], 'START': [], 'STOP': [], 'STOP_SENT_FIN_ACK': [<function TCP_client.stop_fin_received>], 'SYN_SENT': [<function TCP_client.synack_received>]}

reset_received(*pkt*) [source]

send_ack_of_synack(*pkt*) [source]

send_data(*d*) [source]

send_finack(*pkt*) [source]

send_sock: *SuperSocket | None*

send_syn() [source]

states: *Dict[str, _StateWrapper]*
= {'CLOSED': <function ATMT.state.<locals>.deco.<locals>._state_wrapper>, 'ESTABLISHED': <function ATMT.state.<locals>.deco.<locals>._state_wrapper>, 'LAST_ACK': <function ATMT.state.<locals>.deco.<locals>._state_wrapper>, 'START': <function ATMT.state.<locals>.deco.<locals>._state_wrapper>, 'STOP': <function ATMT.state.<locals>.deco.<locals>._state_wrapper>, 'STOP_SENT_FIN_ACK': <function ATMT.state.<locals>.deco.<locals>._state_wrapper>, 'SYN_SENT': <function ATMT.state.<locals>.deco.<locals>._state_wrapper>}

stop_ack_timeout() [source]

stop_fin_received(*pkt*) [source]

stop_requested() [source]

stop_send_ack(*pkt*) [source]

stop_send_finack() [source]

stop_state(**args: ATMT*, ***kargs: Any*)→ NewStateRequested [source]

syn_ack_timeout() [source]

synack_received(*pkt*) [source]

tcplink= *<scapy.automaton._ATMT_to_supersocket object>*

threadid: *int | None*

timeout: *Dict[str, _TimerList]*
= {'CLOSED': [], 'ESTABLISHED': [], 'LAST_ACK': [], 'START': [], 'STOP': [], 'STOP_SENT_FIN_ACK': [<Timer 0.000000(1.000000)>], 'SYN_SENT': [<Timer 0.000000(1.000000)>]}

---

class **scapy.layers.inet.TCPerror**(*_pkt, /, *, sport=20, dport=80, seq=0, ack=0, dataofs=None, reserved=0, flags=<Flag 2 (S)>, window=8192, chksum=None, urgptr=0, options=b''*) [source]

Bases: `TCP`

aliastypes

answers(*other*) [source]

fields_desc

▶ Display RFC-like schema

TCPerror fields

sport	`ShortEnumField`	`20`
dport	`ShortEnumField`	`80`
seq	`MayEnd`	`0`
ack	`IntField`	`0`
dataofs	`BitField` (4 bits)	`None`
reserved	`BitField` (3 bits)	`0`
flags	`FlagsField`	`<Flag 2 (S)>`
window	`ShortField`	`8192`
chksum	`XShortField`	`None`
urgptr	`ShortField`	`0`
options	`TCPOptionsField`	`b''`

mysummary() [source]

---

class **scapy.layers.inet.TracerouteResult**(*res=None, name='Traceroute', stats=None*) [source]

Bases: `SndRcvList`

get_trace() [source]

graph(*ASres=<scapy.as_resolvers.AS_resolver_multi object>*, *padding=0*, ***kargs*) [source]

x.graph(ASres=conf.AS_resolver, other args): ASres=None : no AS resolver => no clustering ASres=AS_resolver() : default whois AS resolver (riswhois.ripe.net) ASres=AS_resolver_cymru(): use whois.cymru.com whois database ASres=AS_resolver(server="whois.ra.net") type: output type (svg, ps, gif, jpg, etc.), passed to dot's "-T" option # noqa: E501 target: filename or redirect. Defaults pipe to Imagemagick's display program # noqa: E501 prog: which graphviz program to use

graphASres

graphdef

graphpadding

hloc

make_graph(*ASres=None*, *padding=0*) [source]

nloc

padding

show() [source]

trace3D(*join=True*) [source]

Give a 3D representation of the traceroute. right button: rotate the scene middle button: zoom shift-left button: move the scene left button on a ball: toggle IP displaying double-click button on a ball: scan ports 21,22,23,25,80 and 443 and display the result

trace3D_notebook() [source]

Same than trace3D, used when ran from Jupyter notebooks

world_trace() [source]

Display traceroute results on a world map.

---

class **scapy.layers.inet.UDP**(*_pkt*, */*, ***, *sport=53*, *dport=53*, *len=None*, *chksum=None*) [source]

Bases: `Packet`

aliastypes

answers(*other*) [source]

extract_padding(*s*) [source]

fields_desc

▶ Display RFC-like schema

UDP fields

sport	ShortEnumField	53
dport	ShortEnumField	53
len	ShortField	None
chksum	XShortField	None

hashret() [source]

mysummary() [source]

payload_guess

Possible sublayers: PDUTransport, HSFZ, DoIP, SOMEIP, BFD, BIFT, CoAP, CRX1New, GENEVE, GTPHeader, GTP_U_Header, HICP, IKEv2, KNX, LDP, LTP, MPLS, MQTTSN, PFCP, ProfinetIO, RIPng, BTH, SebekHead, SOCKS5UDP, STAMPSessionReflectorTestUnauthenticated, STAMPSessionSenderTestUnauthenticated, STUN, VQP, Wireguard, BOOTP, _dhcp6_dispatcher, DNS, HSRP, ESP, Kerberos, Kpasswd, GRE, L2TP, CLDAP, _LLMNR, MGCP, MobileIP, NBNSHeader, NBTDatagram, NetflowHeader, NTP, Radius, RIP, SNMP, TFTP, VXLAN, ZEP2

post_build(*p*, *pay*) [source]

---

class **scapy.layers.inet.UDPerror**(*_pkt*, */*, **, *sport=53*, *dport=53*, *len=None*, *chksum=None*) [source]

Bases: UDP

aliastypes

answers(*other*) [source]

fields_desc

▶ Display RFC-like schema

UDPerror fields

sport	`ShortEnumField`	53
dport	`ShortEnumField`	53
len	`ShortField`	None
chksum	`XShortField`	None

mysummary() [source]

scapy.layers.inet.calc_tcp_md5_hash(*tcp: TCP*, *key: bytes*) → bytes [source]

Calculate TCP-MD5 hash from packet and return a 16-byte string

class **scapy.layers.inet.connect_from_ip**(*host*, *port*, *srcip*, *poison=True*, *timeout=1*, *debug=0*) [source]

Bases: `object`

Open a TCP socket to a host:port while spoofing another IP.

Parameters:
- **host** – the host to connect to
- **port** – the port to connect to
- **srcip** – the IP to spoof. the cache of the gateway will be poisonned with this IP.
- **poison** – (optional, default True) ARP poison the gateway (or next hop), so that it answers us (only one packet).
- **timeout** – (optional) the socket timeout.

Example - Connect to 192.168.0.1:80 spoofing 192.168.0.2:

```
from scapy.layers.http import HTTP, HTTPRequest
client = connect_from_ip("192.168.0.1", 80, "192.168.0.2")
sock = SSLStreamSocket(client.sock, HTTP)
resp = sock.sr1(HTTP() / HTTPRequest(Path="/"))
```

Example - Connect to 192.168.0.1:443 with TLS wrapping spoofing 192.168.0.2:

```
import ssl
from scapy.layers.http import HTTP, HTTPRequest
context = ssl.SSLContext(ssl.PROTOCOL_TLS_CLIENT)
context.check_hostname = False
context.verify_mode = ssl.CERT_NONE
client = connect_from_ip("192.168.0.1", 443, "192.168.0.2")
sock = context.wrap_socket(client.sock)
sock = SSLStreamSocket(client.sock, HTTP)
resp = sock.sr1(HTTP() / HTTPRequest(Path="/"))
```

close() [source]

scapy.layers.inet.defrag(*plist*)→ ([not fragmented], [defragmented], [source]

[[bad fragments], [bad fragments], ...])

scapy.layers.inet.defragment(*plist*)→ plist defragmented as much as possible [source]

scapy.layers.inet.fragleak(*target*, *sport=123*, *dport=123*, *timeout=0.2*, *onlyasc=0*, *count=None*) [source]

scapy.layers.inet.fragleak2(*target*, *timeout=0.4*, *onlyasc=0*, *count=None*) [source]

scapy.layers.inet.fragment(*pkt*, *fragsize=1480*) [source]

Fragment a big IP datagram

scapy.layers.inet.get_tcpao(*tcphdr: TCP*)→ TCPAOValue | None [source]

Get the TCP-AO option from the header

scapy.layers.inet.in4_chksum(*proto: int*, *u: IP*, *p: bytes*)→ int [source]

IPv4 Pseudo Header checksum as defined in RFC793

Parameters:
- **proto** – value of upper layer protocol
- **u** – upper layer instance
- **p** – the payload of the upper layer provided as a string

scapy.layers.inet.in4_pseudoheader(*proto: int*, *u: IP*, *plen: int*)→ bytes [source]

IPv4 Pseudo Header as defined in RFC793 as bytes

Parameters:
- **proto** – value of upper layer protocol
- **u** – IP layer instance
- **plen** – the length of the upper layer and payload

scapy.layers.inet.inet_register_l3(*l2*, *l3*) [source]

Resolves the default L2 destination address when IP is used.

scapy.layers.inet.overlap_frag(*p*, *overlap*, *fragsize=8*, *overlap_fragsize=None*) [source]

Build overlapping fragments to bypass NIPS

p: the original packet overlap: the overlapping data fragsize: the fragment size of the packet overlap_fragsize: the fragment size of the overlapping packet

scapy.layers.inet.report_ports(*target*, *ports*) [source]

portscan a target and output a LaTeX table report_ports(target, ports) -> string

scapy.layers.inet.sign_tcp_md5(*tcp: TCP*, *key: bytes*)→ None [source]

Append TCP-MD5 signature to tcp packet

scapy.layers.inet.tcp_pseudoheader(*tcp: TCP*)→ bytes [source]

Pseudoheader of a TCP packet as bytes

Requires underlayer to be either IP or IPv6

scapy.layers.inet.traceroute(*target*, *dport=80*, *minttl=1*, *maxttl=30*, *sport=<RandShort>*, *l4=None*, *filter=None*, *timeout=2*, *verbose=None*, ***kargs*) [source]

Instant TCP traceroute

Parameters:
- **target** – hostnames or IP addresses
- **dport** – TCP destination port (default is 80)
- **minttl** – minimum TTL (default is 1)
- **maxttl** – maximum TTL (default is 30)
- **sport** – TCP source port (default is random)
- **l4** – use a Scapy packet instead of TCP
- **filter** – BPF filter applied to received packets
- **timeout** – time to wait for answers (default is 2s)
- **verbose** – detailed output

Returns: an TracerouteResult, and a list of unanswered packets

scapy.layers.inet.traceroute_map(*ips*, ***kargs*) [source]

Util function to call traceroute on multiple targets, then show the different paths on a map.

Parameters:
- **ips** – a list of IPs on which traceroute will be called
- **kargs** – (optional) kwargs, passed to traceroute