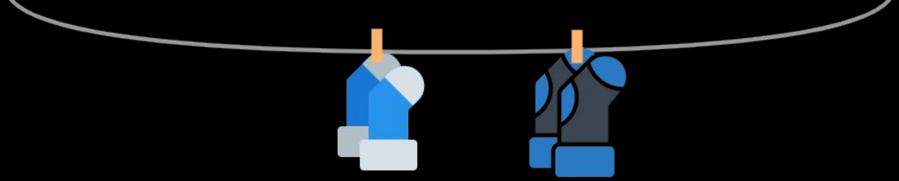


SOCKS overTURNed

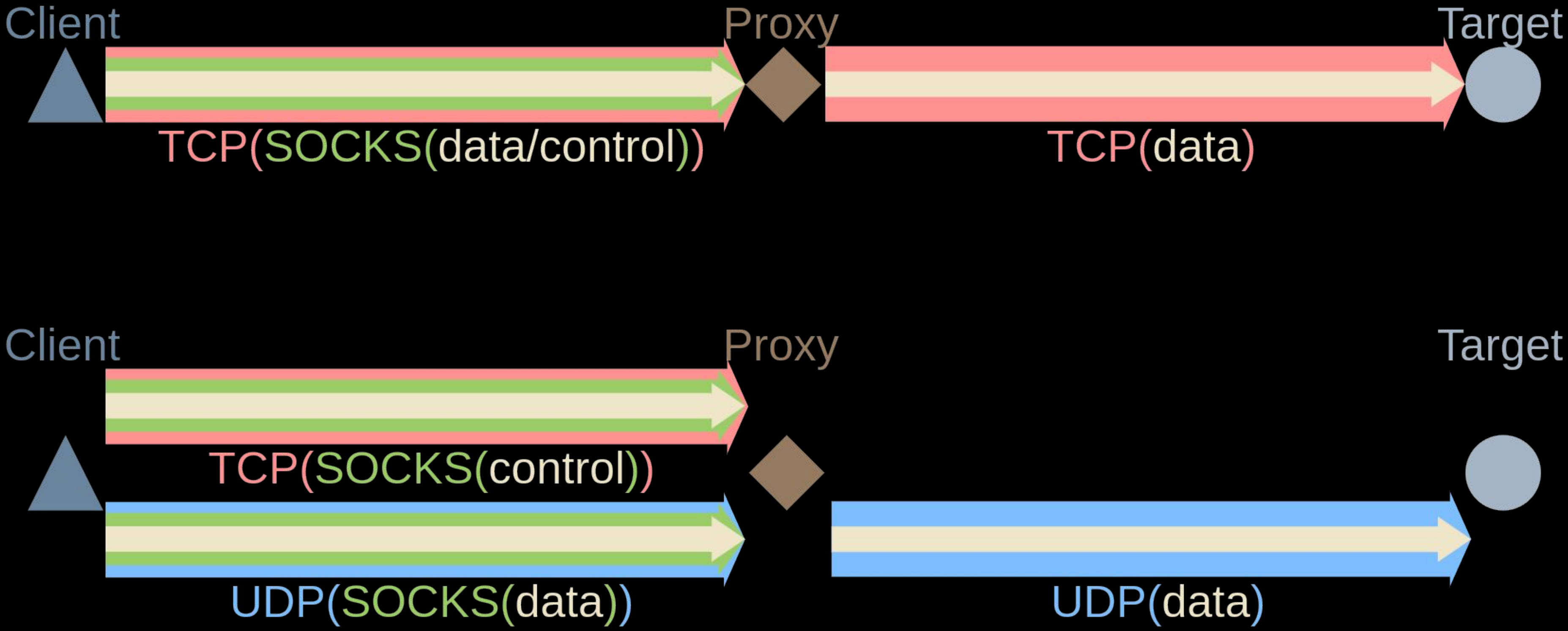
RP86: Using TURN relays as Proxies

Sean Liao

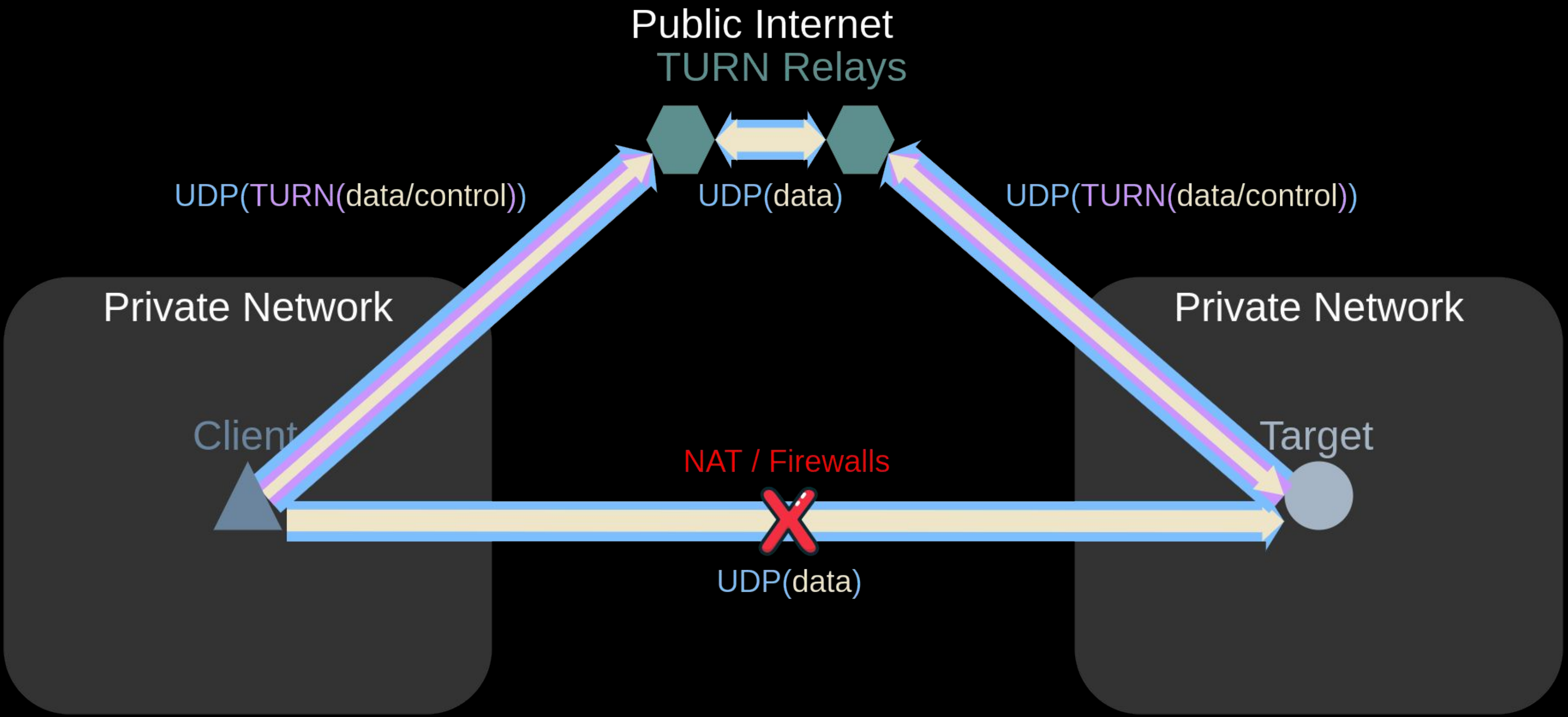
SOCKS

- Not short for anything
- Widely supported generic proxy protocol
- Layer 4
- SOCKS4 (1992) / SOCKS4a: TCP
- SOCKS5 (1996, RFC 1928): TCP & UDP

SOCKS


Client
Proxy
Target
TCP(SOCKS(data/control))
TCP(data)
Client
Proxy
Target
TCP(SOCKS(control))
UDP(SOCKS(data))
UDP(data)

TURN - Traversal Using Relays around NAT



Public Internet

TURN Relays

UDP(TURN(data/control))

UDP(data)

UDP(TURN(data/control))

Private Network

Private Network

Client

Target

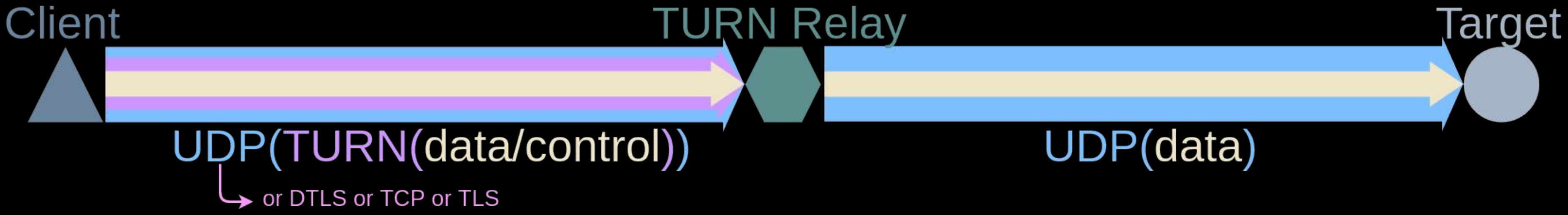
NAT / Firewalls

UDP(data)

TURN

- Traversal Using Relays around NAT
- Designed / used primarily for audio / video communications
- Extension of STUN (Session Traversal Utilities for NAT) protocol
- Implemented by web browsers as part of WebRTC
- Layer 4
- Base protocol (RFC 5766): UDP/TCP/TLS to proxy, UDP to destination
- RFC 6062: TCP to destination

TURN



Client

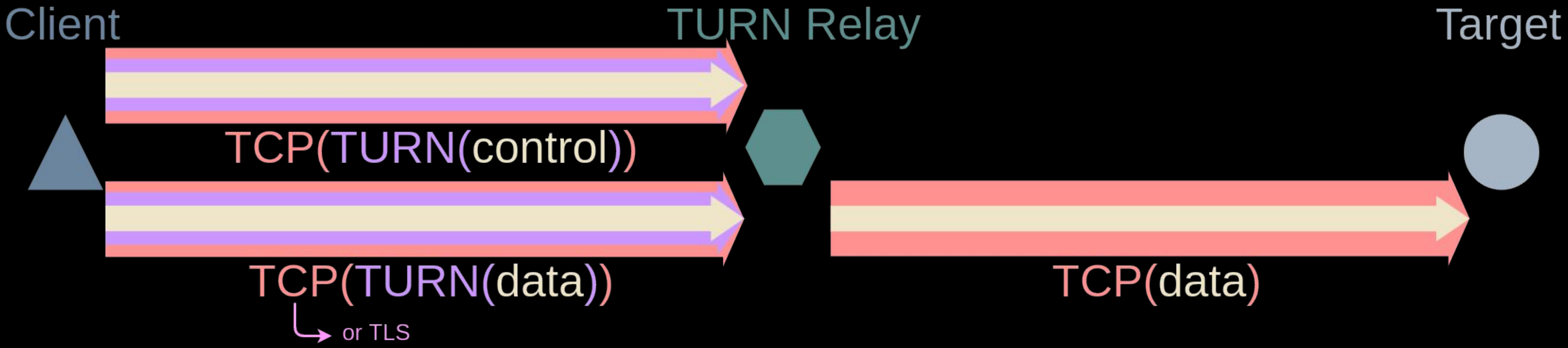
TURN Relay

Target

UDP(TURN(data/control))

or DTLS or TCP or TLS

UDP(data)



Client

TURN Relay

Target

TCP(TURN(control))

TCP(TURN(data))

or TLS

TCP(data)

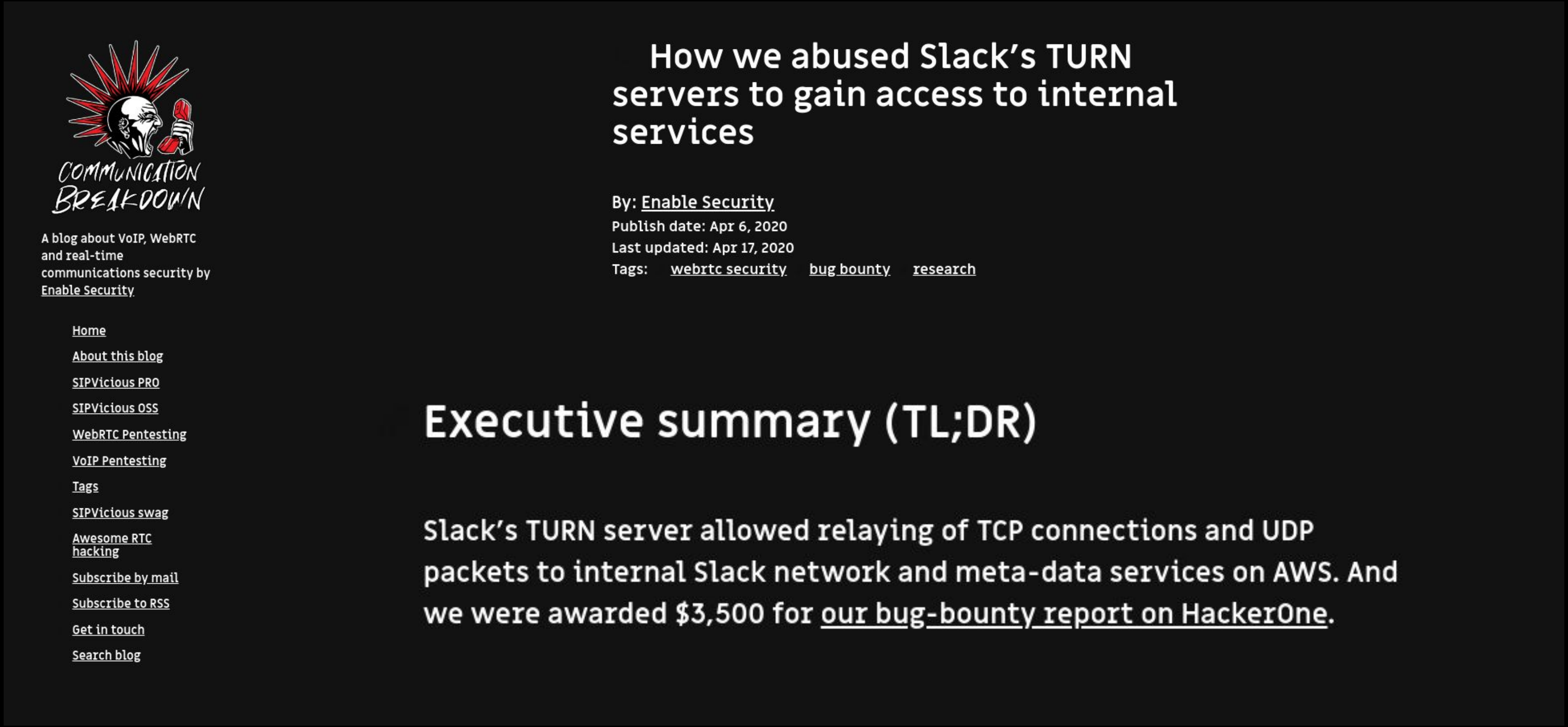
Basic TURN connection

```go
// setup client for udp
turnConfig := &turn.ClientConfig{
    TURNServerAddr: p.turnAddress,
    Conn:           udpconn, // raw udp socket listener started earlier
    Username:       p.turnUser,
    Password:       p.turnPass,
    Realm:          p.turnRealm,
}
client, _ := turn.NewClient(turnConfig)
client.Listen()

// allocate a udp port on TURN relay
relayConn, _ := client.Allocate()


// read data from remote
_, sourceAddr, _ := relayConn.ReadFrom(buffer)

// write data to remote
relayConn.WriteTo(buffer, destinationAddr)
```

Chained Together

- Let SOCKS clients talk to TURN relays
  - Mask originating address
  - Access private network connected to relay
  - Access outside network using whitelisted relay
- Forwarding
  - TURN relay makes connection to final destination
- Reverse Connection
  - Red teaming
  - Establish connection through relay to known (whitelisted) endpoint
  - Serve connections in reverse direction, open up internal network

Related Work - Forward / Slack / Enable Security



A blog about VoIP, WebRTC and real-time communications security by Enable Security

- Home
- About this blog
- SIPVicious PRO
- SIPVicious OSS
- WebRTC Pentesting
- VoIP Pentesting
- Tags
- SIPVicious swag
- Awesome RTC hacking
- Subscribe by mail
- Subscribe to RSS
- Get in touch
- Search blog

How we abused Slack's TURN servers to gain access to internal services

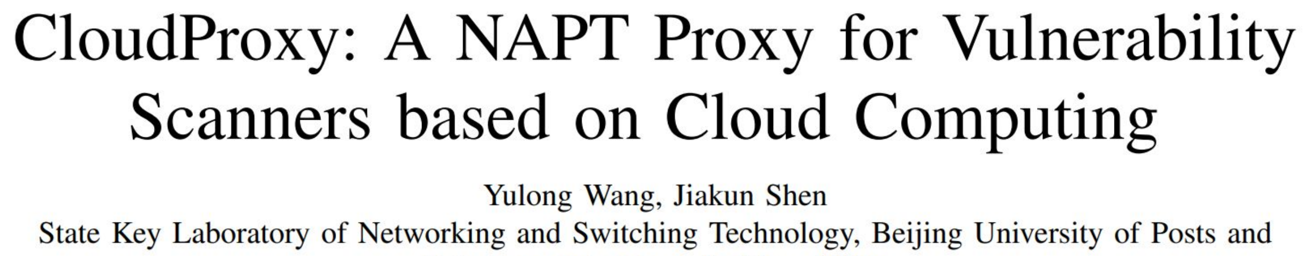
By: Enable Security
Publish date: Apr 6, 2020
Last updated: Apr 17, 2020
Tags: webrtc security bug bounty research

Executive summary (TL;DR)

Slack's TURN server allowed relaying of TCP connections and UDP packets to internal Slack network and meta-data services on AWS. And we were awarded $3,500 for our bug-bounty report on HackerOne.

https://www.rtcsec.com/2020/04/01-slack-webrtc-turn-compromise/

Related Work - Reverse / CloudProxy



CloudProxy: A NAPT Proxy for Vulnerability Scanners based on Cloud Computing

Yulong Wang, Jiakun Shen
State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications, Beijing, China
Email: {wyl, moretea_sjk}@bupt.edu.cn

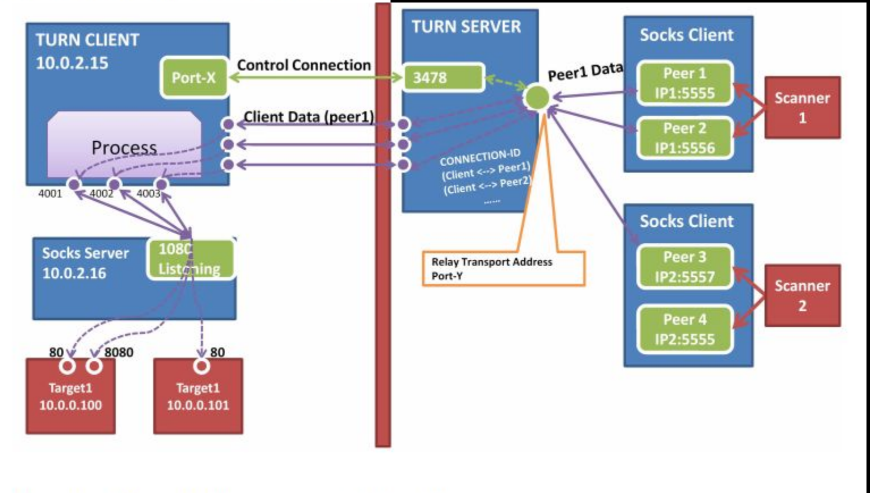
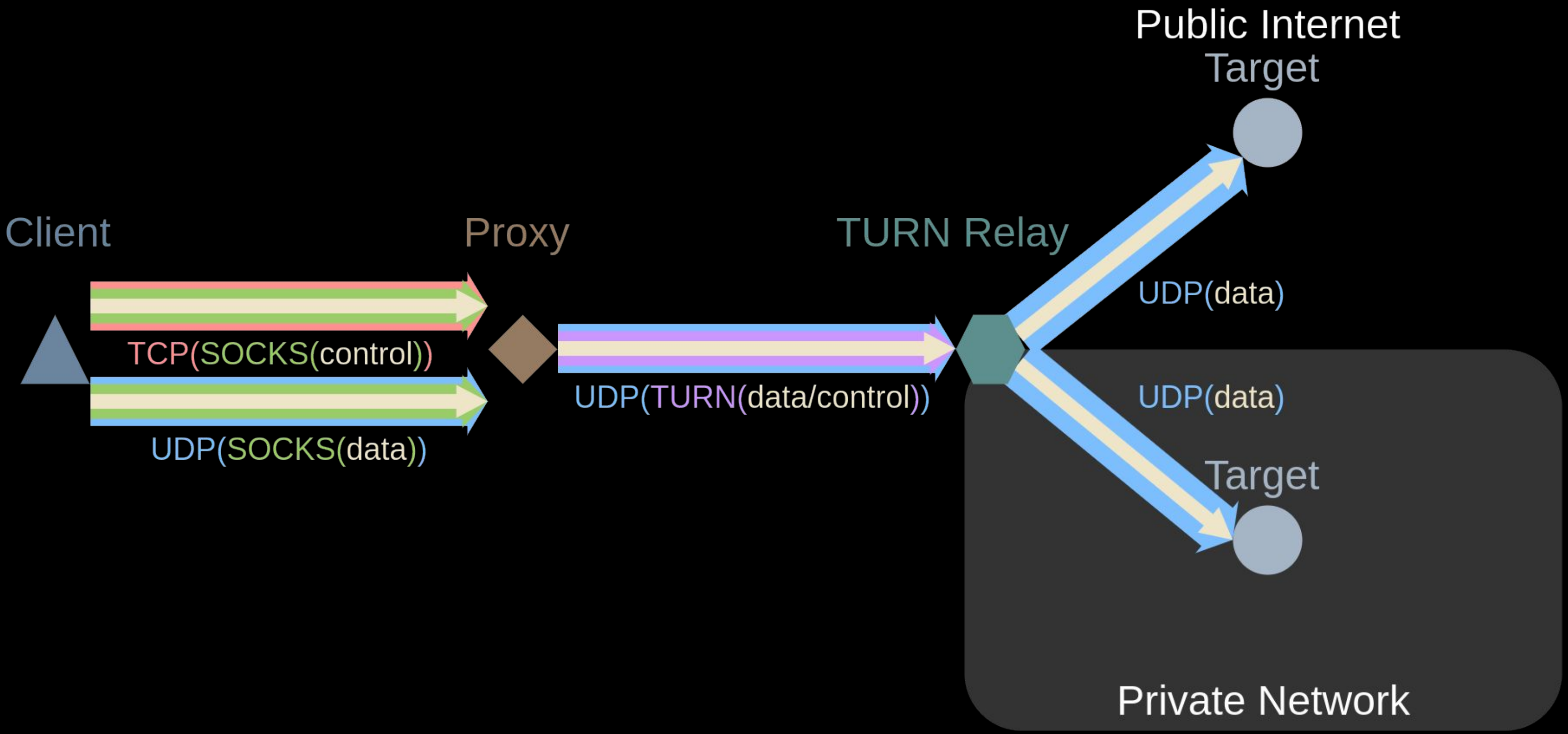


Fig. 3. Overall Structure of CloudProxy

https://pdfs.semanticscholar.org/4887/7c7dccc81fa24a1a7579c46c7eaadbf8e792.pdf

Forward UDP


Public Internet
Target
Client
Proxy
TURN Relay
UDP(data)
TCP(SOCKS(control))
UDP(TURN(data/control))
UDP(data)
UDP(SOCKS(data))
Target
Private Network

TURN UDP

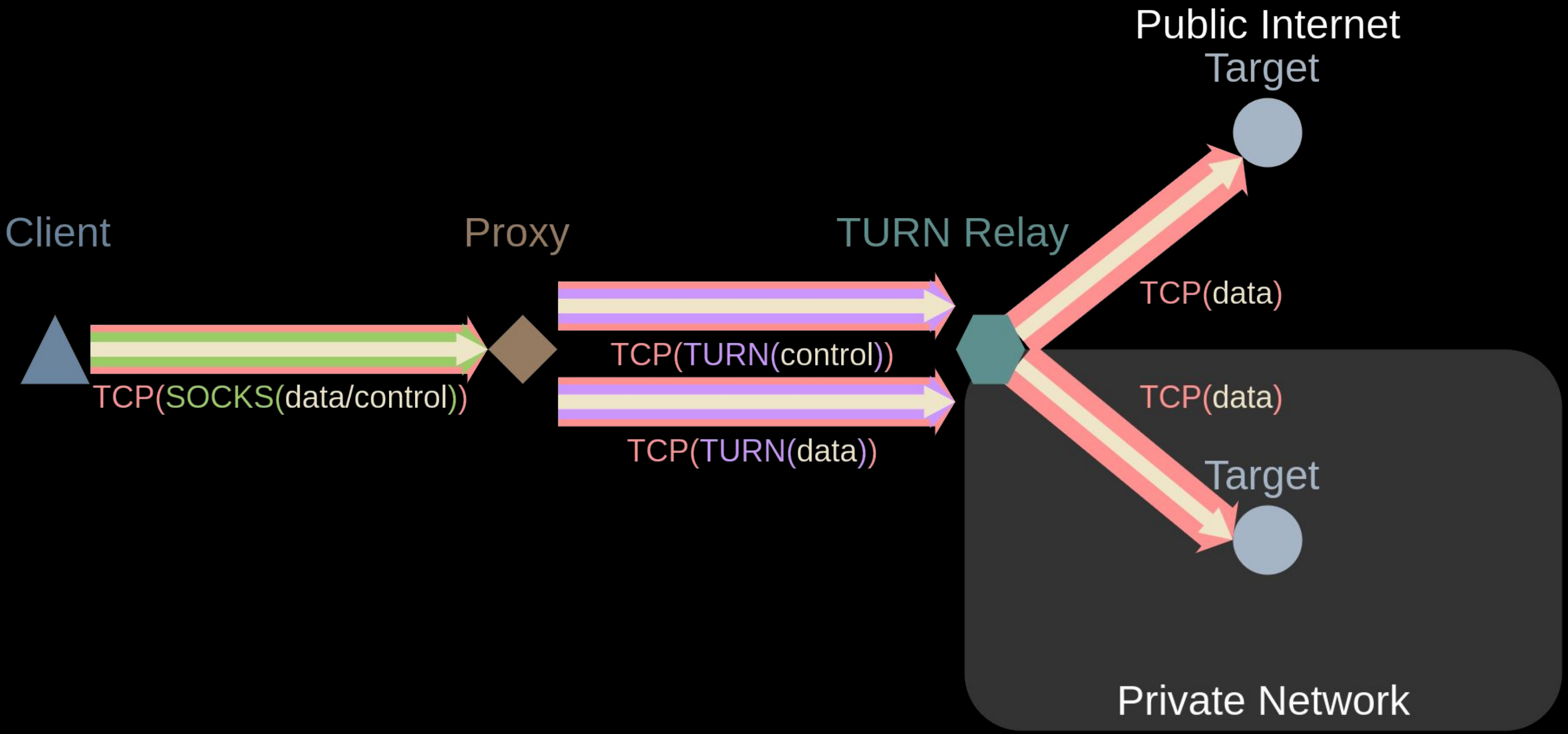
on incoming SOCKS packet

```
// retrieve existing session
uSess := conns[srcAddr.String()]

// start new session on demand
if uSess == nil {
    _, dconn, _ := f.Proxy.connectUDP()
    uSess = &session{dconn, srcAddr, srcConn}
    conns[srcAddr.String()] = uSess
    go uSess.handleIncoming()
}

// write to TURN relay
uSess.dconn.WriteTo(data, dstAddr)
```

on incoming TURN packet

```
// uSess.handleIncoming

// read packet
n, from, _ := dconn.ReadFrom(buf)

// wrap in SOCKS packet
datagram := socks5.NewDatagram(srcAddr, buf[:n])

// write to SOCKS client
srcConnconn.WriteToUDP(datagram.Bytes(), srcAddr)
```

Forward TCP


Public Internet
Target
Client
Proxy
TURN Relay
TCP(data)
TCP(TURN(control))
TCP(SOCKS(data/control))
TCP(data)
TCP(TURN(data))
Target
Private Network

TURN TCP - Implement RFC 6062

```
// Update client.Allocate
// make protocol configurable
proto.RequestedTransport{Protocol: c.transportProtocol},


// Have the TURN relay connect to a remote destination
func (c *Client) Connect(peer *net.TCPAddr) (ConnectionID, error) {
    msg := stun.New()
    stun.NewType(stun.MethodConnect,stun.ClassRequest).AddTo(msg)
    stun.XORMappedAddress{peer.IP, peer.Port}.AddToAs(msg, stun.AttrXORPeerAddress)
    // other fields omitted

    res := c.PerformTransaction()

    // extract connection ID from successful response
    var cid ConnectionID
    cid.GetFrom(res)
}
```

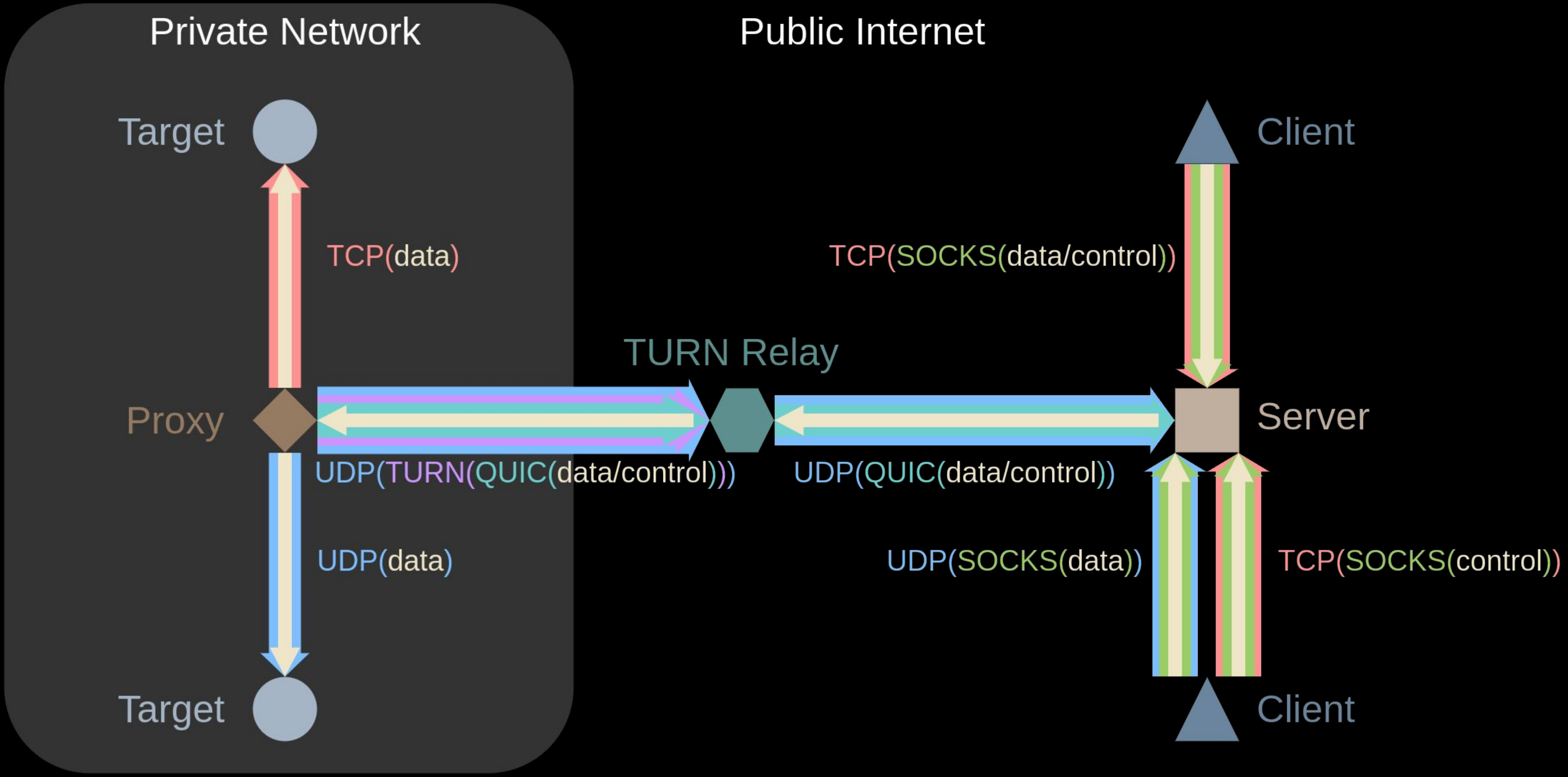
TURN TCP - Implement RFC 6062

```
// Associate an new tcp connection with a remote connection on the TURN relay
func (c *Client) ConnectionBind(dataConn net.Conn, cid ConnectionID) error {
    msg := stun.Build(
        stun.NewType(stun.MethodConnectionBind, stun.ClassRequest),
        cid,
        // other fields omitted
    )

    // write binding request
    dataConn.Write(msg.Raw)

    // read response, limit to response bytes only
    dataConn.Read(buf)

    // omitted verify success
}
```

TURN TCP

```
// same as before
// but specify transport protocol in turnConfig
controlConn, _ := net.Dial("tcp", turnAddress)
client, _ := turn.NewClient(turnConfig)
client.Listen()
client.Allocate()

// make relay connect to remote destination
connecttionID, _ := client.Connect(dstAddr)

// open new connection for data
dataConn, _ := net.Dial("tcp", turnAddress)
// associate connection with connect attempt
client.ConnectionBind(dataConn, connecttionID)

// read from TURN relay / destination
dataConn.Read(buffer)

// write to TURN relay / destination
dataConn.Write(buffer)
```

Reverse


Private Network
Public Internet
Target
TCP(data)
Proxy
UDP(TURN(QUIC(data/control)))
UDP(data)
Target
TURN Relay
UDP(QUIC(data/control))
Client
TCP(SOCKS(data/control))
Server
UDP(SOCKS(data))
TCP(SOCKS(control))
Client

Proxy

```
// connect to TURN relay with UDP
_, uconn, _ := proxy.connectUDP()

// connect with QUIC over TURN connection
qSession, _ := quic.Dial(uconn, serverAddr, serverHost, tlsConf, quicConf)

for {
    // wait for incoming connections from server
    stream, _ := qSession.AcceptStream(ctx)

    // extract protocol and destination address
    proto := readMessage(stream)
    addr := readMessage(stream)

    // serve connection
    switch proto {
        case "tcp":
            go serveTCP(addr, stream)
        case "udp":
            go serveUDP(addr, stream)
    }
}
```

Server

```
// accept incoming QUIC connection and start SOCKS servers
qListener, _ := quic.ListenAddr(serverAddr, tlsConf, nil)
for {
      qSession, _ := qListener.Accept(ctx)
      go func(){
            srv := socksServer()
            srv.ListenAndServe(&conn{qSession})
      }()
}

// example for incoming SOCKS/TCP to TCP/QUIC/TURN
stream, _ := conn.qSession.OpenStream()
writeMessage(stream, “tcp”)
writeMessage(stream, dstAddr)

// tell client connection is successful
reply := socks5.NewReply(socks5.RepSuccess, /* omitted */)
reply.WriteTo(clientconn)

// copy data between connections
go io.Copy(clientConn, stream)
io.Copy(stream, clientConn)
```

Problems

- TCP support
  - Only a single server implementation supports it: Coturn
  - No client (or server) library support in any language, implement it in Go
- TURN sessions
  - Single connection to host:port per session, problems with HTTP1, virtual domains
  - No closing connections
- DNS resolution
  - SOCKS library and TURN work with IP addresses
  - Split horizon DNS

Code

- Extended Library
  - pion/turn rfc6062 branch
  - https://github.com/pion/turn/tree/rfc6062
- Proxy code
  - https://github.com/seankhliao/uva-rp2/tree/master/cmd/proxy

Testing

- Find TURN relays in the wild
    - Use popular videoconferencing solutions
- Use own account / credentials
    - “insider”
- Patch Chromium to dump out credentials
    - Not the same as login credentials
    - Each service has its own way of transfering TURN credentials

Services

	UDP	TCP
Zoom	no TURN	
Google Meet	no TURN	
Cisco Webex (CiscoThinClient)	Drops after allocate	
GoToMeeting (Citrix)	“Wrong Transport Field”	
Slack (Amazon Chime)	Forbidden IP	X
Microsoft Teams / Skype	V	X
Facebook Messenger	V	X
Jitsi Meet	V	X
Riot.im (Matrix)	V	X
BlueJeans	V	X

Defense - Network / Firewall Operators

- TURN (RFC8155): Server auto discovery
  - mDNS / Anycast
  - Run your own STUN/TURN relay?
  - Clients need to support this
- Deep Packet Inspection / Network Flow Analysis (?)
- Push security to endpoints

Defense - TURN Operators

- Hiding Servers
  - Non default ports
  - Load balancers with TLS SNI (Server Name Indication)
- Authentication
  - “long-term credentials” are short term & on demand in practice
  - Requested over HTTP+JSON, XMPP, gRPC, …
  - Linked / additional auth
  - Verify realm
- Restricing Services
  - Limiting sessions
  - Disable unused protocols, ex. TCP
  - Block internal ranges
  - Block low ports
  - Architectural changes?
    - P2P Mesh vs MCU (Multipoint Conferencing Unit) vs SFU (Selective Forwarding Unit)

Conclusion / Future Work

- Can work
  - UDP works everywhere
  - Very little TCP support
- Red Teaming
  - Only need UDP and whitelisted server
- Difficult to protect against
  - Designed to tunnel through
- Credentials are hard to get
  - Reverse engineer credential exchange for stable credentials

- IPv6 support
- Masking traffic as audio / video
- Embed into applications / webpages
- Integrate into frameworks / Metasploit
- Reuse code from existing applications, browsers, meeting software
- Coopt WebRTC?