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
TURN - Traversal Using Relays around NAT
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
Basic TURN connection

// 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

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

https://pdfs.semanticscholar.org/4887/7c7dccc81fa24a1a7579c46c7eaadb8e792.pdf
Forward UDP

Client

TCP(SOCKS(control))

UDP(SOCKS(data))

Proxy

UDP(TURN(data/control))

TURN Relay

UDP(data)

Public Internet Target

UDP(data)

Target

Private Network
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
srcConn.dconn.WriteToUDP(datagram.Bytes(), srcAddr)
Forward TCP

Client → Proxy → TURN Relay → Public Internet Target

TCP(SOCKS(data/control)) → TCP(TURN(control)) → TCP(TURN(data)) → TCP(data)

Client

Proxy

TURN Relay

Public Internet Target

TCP(data)

Target

Private Network
// Update client.Allocate
// make protocol configurable

// 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
connectionID, _ := client.Connect(dstAddr)

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

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

// write to TURN relay / destination
dataConn.Write(buffer)
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)
    }
}
// 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 */) // omitted */
reply.WriteTo(clientconn)

// copy data between connections
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](https://github.com/pion/turn/tree/rfc6062)

- **Proxy code**
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 transferring TURN credentials
<table>
<thead>
<tr>
<th>Services</th>
<th>UDP</th>
<th>TCP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Zoom</td>
<td>no TURN</td>
<td></td>
</tr>
<tr>
<td>Google Meet</td>
<td>no TURN</td>
<td></td>
</tr>
<tr>
<td>Cisco Webex (CiscoThinClient)</td>
<td></td>
<td>Drops after allocate</td>
</tr>
<tr>
<td>GoToMeeting (Citrix)</td>
<td></td>
<td>“Wrong Transport Field”</td>
</tr>
<tr>
<td>Slack (Amazon Chime)</td>
<td>Forbidden IP</td>
<td></td>
</tr>
<tr>
<td>Microsoft Teams / Skype</td>
<td>V</td>
<td>X</td>
</tr>
<tr>
<td>Facebook Messenger</td>
<td>V</td>
<td>X</td>
</tr>
<tr>
<td>Jitsi Meet</td>
<td>V</td>
<td>X</td>
</tr>
<tr>
<td>Riot.im (Matrix)</td>
<td>V</td>
<td>X</td>
</tr>
<tr>
<td>BlueJeans</td>
<td>V</td>
<td>X</td>
</tr>
</tbody>
</table>
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

- Restricting 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?