Passthru - A Protocol Omni-multiplexer

(De-)multiplexing TCP connections based on Protocol-specific Application Data sniffing.

Product Definition

Milestone Report

Slides

GitHub Repository

Original Problem

It is difficult to proxy multiple target hosts (running similar or different services) through a single TCP/UDP port. For example, proxying TLS services WITHOUT obtaining a certificate for each target would usually cost one tcp port per target. Under certain circumstances, this may cause extra cost for a reverse proxy server when trying to relay multiple targets using one premium port.

Solution: Deep Packet Inspection

Passthru is the first protocol omni-multiplexer that can de-multiplex TCP connections based on protocol-specific application data sniffing with interface allowing users to define their own protocol handlers.

Each `Protocol` can detect a certain protocol by matching the first few bytes (usually the first incoming packets) in the connection. If a match is found, the `Protocol` will return the corresponding `Action` to the `Handler`. The `Handler` will then forward the connection to the corresponding `Action` handler.

Demo: Usage

Build

```
$ go build ./cmd/passthru
```

Run

```
$ ./passthru -c=<configfile> -w=<worker_num> -t=<timeout>
```

Note that in the current release, the `worker_num` and `timeout` are not used for simplicity in demo. You may manually enable it in the `main()` function.

Config

```
{
    "version": "v0.2.0",
    "servers": {
        "127.0.0.1:443": {
            "TLS": {
                "SNI gaukas.wang": {
                    "action": "FORWARD",
                    "to_addr": "185.199.111.153:443"
                },
                "SNI google.com": {
                    "action": "FORWARD",
                    "to_addr": "142.250.72.46:443"
                },
                "SNI www.google.com": {
                    "action": "FORWARD",
                    "to_addr": "142.250.72.46:443"
                },
                "CATCHALL": {
                    "action": "REJECT"
                }
            },
            "CATCHALL": {
                "CATCHALL": {
                    "action": "FORWARD",
                    "to_addr": "neverssl.com:80"
                }
            }
        }
    }
}
```

Packages

Config

The config package is used to parse the config file. The config file follows the structure below:

- Version
- Servers
    - ServerAddr1
        - Protocol1: defined in `protocol` package
            - Rule1
                - Action: either `FORWARD` or `REJECT`
                - ToAddr (`FORWARD` only): the address to forward to
            - Rule2
                - ...
            - CATCHALL (as a rule)
                - Action
                - ToAddr (optional)
        - Protocol2: defined in `protocol` package

      - Rule1
        - ...
      - ...
    - CATCHALL (as a protocol)
      - CATCHELL (as the only rule of this protocol)
        - ...
  - ServerAddr2
    - ...

Handler

Handler defines the handler of all incoming connections to a certain address as a `Server`.

When `Server` receives a connection, it streams the first few bytes of the connection to `ProtocolManager` to identify the protocol and search for the corresponding action(REJECT/FORWARD) to apply to the connection. Once a match is found, the `Server` will handle the connection according to the action, including forwarding the connection to the corresponding address or rejecting the connection immediately.

Protocol

Protocol defines the identifier/detector of a certain `Protocol`. It also defines `ProtocolManager` to manage all known protocols. Upon receiving a buffer of bytes, `ProtocolManager` will try to match the buffer with all known protocols. If a match is found, the matching `Action` will be returned.

`Protocol` Interface

A `Protocol` interface defines a detector module which is capable of identifying a certain protocol according to the unencrypted application data at the beginning of a TCP connection.

```go
// Protocol is the interface for protocol identification.
type Protocol interface {
    // Name prints the name of the protocol, like "TLS", which is going to be used
as a key in the ProtocolGroup
    Name() config.Protocol

    // Clone creates a new Protocol instance with the same rules (as a deep copy)
    Clone() Protocol

    // ApplyRules save the rules for later Identify calls.
    // Note the rules are out-of-order intentionally to prevent conflicting rules.
    // Protocol implementations should make sure the CATCHEALL rule is always the
last rule to be applied.
    ApplyRules(rules []config.Rule) error

    // Identify identifies the rule that matches the request.
    Identify(ctx context.Context, cBuf *ConnBuf) (config.Rule, error) // Identify
will keep checking cBuf until it can make a deterministic decision or the context
is cancelled.
}
```

For an example, please see our TLS `Protocol` implementation in `protocol/tls/protocol.go`.

Related Work

Reverse Proxy

For example, *HAProxy, Nginx, Squid*. These reverse proxies are capable of proxying multiple targets through multiple ports only. However, they are not capable of proxying multiple targets through a single port. Thesefore they incur extra cost for a reverse proxy server especially on premium infrastructure with limited ports.

CDN

For example, *Cloudflare, Cloudfront, Akamai*. These CDNs are capable of proxying multiple targets (HTTP-over-TLS websites only) through a single port. However, they are not capable of proxying multiple targets through a single port without obtaining a certificate for each target, which imposes a threat to the privacy.