Security Advisory

Goproxy & Smokescreen Denial Of Service (DoS)

Created by Lorenzo Stella, Doyensec LLC
08/16/2022

Overview

This document summarizes the results of a vulnerability discovered during a separate engagement activity in the popular elazarl/goproxy library (4.9k stargazers, 925 forks as of August 2022). The issue was also found in a Stripe-maintained fork (stripe/goproxy), consequently affecting another widespread library that was using it as a dependency, stripe/smokescreen, used to mitigate Server-Side Request Forgery (SSRF) attacks. While security testing was not meant to be comprehensive in terms of attack and code coverage, we have identified several Denial Of Service (DoS) vulnerabilities that could severely affect the availability of any software using the aforementioned libraries.

About Us

Doyensec is an independent security research and development company focused on vulnerability discovery and remediation. We work at the intersection of software development and offensive engineering to help companies craft secure code.

Research is one of our founding principles and we invest heavily in it. By discovering new vulnerabilities and attack techniques, we constantly improve our capabilities and contribute to secure the applications we all use.

Goproxy & Smokescreen Prone To DoS

Vendor	Stripe
Severity	**Low**
Vulnerability Class	Denial of Service (DoS)
Components	• github.com/stripe/goproxy • github.com/elazarl/goproxy • github.com/stripe/smokescreen
Status	Fixed for: • github.com/stripe/smokescreen • github.com/stripe/goproxy Still unresolved for: • github.com/elazarl/goproxy
CVE	N/A
Credits	Lorenzo Stella

Summary

From the *stripe/smokescreen*'s project README:

> *"Smokescreen[1] is a HTTP CONNECT proxy. It proxies most traffic from Stripe to the external world (e.g., webhooks). Smokescreen restricts which URLs it connects to: it resolves each domain name that is requested and ensures that it is a publicly routable IP and not a Stripe-internal IP. This prevents a class of attacks where, for instance, our own webhooks infrastructure is used to scan Stripe's internal network. Smokescreen also allows us to centralize egress from Stripe, allowing us to give financial partners stable egress IP addresses and abstracting away the details of which Stripe service is making the request."*

Stripe's *Smokescreen* proxy is internally created using a fork of the *elazarl/goproxy* package (*stripe/goproxy*), a high-performance network proxy. Its README describes it as:

> *"A customizable HTTP proxy library for Go (golang), It supports regular HTTP proxy, HTTPS through CONNECT, and "hijacking" HTTPS connection using "Man in the Middle" style attack."*

Both *elazarl/goproxy* and *stripe/goproxy* codebases are using the package `ioutil` to leverage common or recurring utility functions for reading/writing plain HTTP responses. One of these utilities is the `ReadAll` function. When attempting to read the plain HTTP request body `resp.Body`, the function will read in the incoming payload and attempt to assign it to the `resp` variable. This results in the entire response body

[1] https://github.com/stripe/smokescreen, the `smokescreen_proxy_url` is hardcoded to `http://localhost:4750`.

being loaded into memory from a remote network request. This is particularly dangerous since some of the requested hosts can be selected and controlled by untrusted users. An attacker could abuse this implementation to load large chunks of content into the server's memory, causing an Out-Of-Memory (OOM) error condition (and potentially the consequent forceful restart of the container/service).

Technical Description

The following function appears to read the response insecurely:

- `NewConnectDialToProxyWithHandler` (`github.com/stripe/goproxy/https.go:388`)
  - Referenced by `NewConnectDialToProxy` (`github.com/stripe/goproxy/https.go:384`)
    - Referenced by `dialerFromEnv` (`github.com/stripe/goproxy/https.go:373`)
      - Referenced by `NewProxyHttpServer` (`github.com/stripe/goproxy/proxy.go:180`). This creates and returns a proxy server, which is used by *Smokescreen* (`smokescreen/smokescreen.go:399`) to provide the proxy functionality.

```
func (proxy *ProxyHttpServer) NewConnectDialToProxyWithHandler(https_proxy
string, connectReqHandler func(req *http.Request)) func(network, addr string)
(net.Conn, error) {
 u, err := url.Parse(https_proxy)
 if err != nil {
        return nil
 }
 if u.Scheme == "" || u.Scheme == "http" {
        if strings.IndexRune(u.Host, ':') == -1 {
               u.Host += ":80"
        }
        return func(network, addr string) (net.Conn, error) {
               connectReq := &http.Request{
                      Method: "CONNECT",
                      URL:    &url.URL{Opaque: addr},
                      Host:   addr,
                      Header: make(http.Header),
               }
               if connectReqHandler != nil {
                      connectReqHandler(connectReq)
               }
               c, err := proxy.dial(network, u.Host)
               if err != nil {
                      return nil, err
               }
               connectReq.Write(c)
               // Read response.
               // Okay to use and discard buffered reader here, because
               // TLS server will not speak until spoken to.
               br := bufio.NewReader(c)
               resp, err := http.ReadResponse(br, connectReq)
               if err != nil {
                      c.Close()
                      return nil, err
               }
               defer resp.Body.Close()
               if resp.StatusCode != 200 {
                      resp, err := ioutil.ReadAll(resp.Body)
                      if err != nil {
                             return nil, err
                      }
                      c.Close()
```

```
                        return nil, errors.New("proxy refused connection" +
string(resp))
                }
                return c, nil
        }
    }

     ...

   return nil
}
```

Reproduction Steps

It should be possible to trigger multiple plain HTTP requests from different webhooks and return very large response bodies at the same time to crash the proxy.

Impact

Depending on the supervisor's restart policy set up on the machine, the process using Goproxy directly or as a sub-dependency (Smokescreen) could crash or be killed, leading to a considerable downtime of the service in case of a prolonged attack.

Complexity

Low. An attacker only needs to fire multiple HTTP webhooks to a controlled endpoint, returning very large request bodies.

Remediation

While all the Stripe-owned libraries have released an update to address the issue, some sinks in elazarl/goproxy remain unfixed. The mitigation was achieved by avoid loading arbitrary data into memory regardless of the size and limiting the size of a valid response while returning an error closing the connection when it consumes a substantial amount of memory.

The `io.LimitReader()`[2] was used, through which it is possible to specify the requested maximum amount of bytes to read, e.g.:

```
limited := io.LimitReader(fz, 40*1024*1024)
s, err := ioutil.ReadAll(limited)
```

The `io.Reader` returned by `io.LimitReader()` will report `io.EOF` when the returned data is more than 40 MB[3]. More information on the vulnerability can be found at:

- "Be careful with ioutil.ReadAll in Golang", Haisum Bhatti
  https://haisum.github.io/2017/09/11/golang-ioutil-readall/

[2] https://golang.org/pkg/io/#LimitedReader

[3] https://stackoverflow.com/questions/56629115/how-to-protect-service-from-gzip-bomb

Disclosure Timeline

03/14/2022	Disclosure to elazar/goproxy maintainer (Elazar Leibovich, <elazarl@gmail.com>), stripe/goproxy & stripe/smokescreen maintainers (Stripe Security Team <security@stripe.com>)
05/8/2022	Fix for stripe/goproxy (39ea38205)
05/8/2022	Dependency bump in stripe/smokescreen (dbb816b6) of stripe/goproxy
08/16/2022	Advisory released by Doyensec