New issue

Jump to bottom

Bypass mTLS by mixing SNI and Host headers #3340

Closed ropnop opened this issue on Apr 13, 2021 · 6 comments

Assignees

Labels t:risk

Milestone 2021 Cycle 5

ropnop commented on Apr 13, 2021

Describe the bug
If Ambassador is deployed with multiple hosts/tlscontexts with some using mTLS (`cert_required: true`) and some not, it is possible to bypass the mTLS requirements for the backends that have it configured by sending an SNI for a TLSContext that does not.

I believe this stems from the fact that Ambassador matches TLSContexts based on SNI, but Mappings based on Host header. If `service1.upstream` does not require mTLS but `service2.upstream` does, it looks like its possible to talk directly to `service1.upstream` without a valid client certificate by doing something like:

```
curl -H "Host: service2.upstream" https://service1.upstream
```

In some configurations it is also possible to do the same by just talking directly to the Ambassador LoadBalancer IP address (not sending SNI at all). This works if the "default" TLSContext does not require mTLS (I'm not exactly clear what TLSContext Ambassador will fall back on if no SNI is given)

To Reproduce
I've put together a working minimal example with 2 services and the YAML configurations:

https://gist.github.com/ropnop/f306a8b46957dbf3a5199899b72be0b5

```
$ /usr/bin/curl --cacert certs/ServerTestCA.crt https://echo.test
curl: (35) error:1401E410:SSL routines:CONNECT_CR_FINISHED:sslv3 alert handshake failure

$ /usr/bin/curl --cacert certs/ServerTestCA.crt -H "Host: echo.test" https://quote.test
CLIENT VALUES:
client_address=('10.1.0.8', 49418) (10.1.0.8)
command=GET
path=/
real path=/
query=
request_version=HTTP/1.1

SERVER VALUES:
server_version=BaseHTTP/0.6
sys_version=Python/3.5.0
protocol_version=HTTP/1.0

HEADERS RECEIVED:
accept=*/*
content-length=0
host=echo.test
user-agent=curl/7.64.1
x-envoy-expected-rq-timeout-ms=3000
x-envoy-internal=true
x-envoy-original-path=/
x-forwarded-for=192.168.65.3
x-forwarded-proto=https
x-request-id=846c224a-c63f-444b-b68d-68aee96122ae
```

Expected behavior
Ambassador should not apply a Host mapping if the SNI does not match. If this is expected behavior and you can't securely mix mTLS and non-mTLS upstreams this should definitely be called out in the docs for those who need to rely on mTLS authentication.

Versions (please complete the following information):

- Ambassador: 1.7.2
- Kubernetes environment: EKS and DockerDesktop
- Version: 1.16 and 1.19

Additional context
It is possible my configuration is bad/wrong, so if I have missed something in the docs please let me know

ropnop commented on Apr 30, 2021 Author

FWIW i've recreated this behavior on the latest version of the Helm chart, running AES 1.13.2 as well

ropnop commented on Apr 30, 2021 Author

Looking at the generated Envoy config, i'm guessing the cause is the first entry in the filter_chain, which matches any TLS traffic:

```
$ cat envoy.json| jq '.static_resources.listeners[0].filter_chains[0].filter_chain_match'
{
  "transport_protocol": "tls"
}
```

It looks like Ambassador applies every mapping to this "catch-all" TLS filter (`filter_chains[0]`). It adds a route to my mTLS backend to the default vhost (`"domains: ["*"]`) based on host header:

```
$ cat envoy.json| jq '.static_resources.listeners[0].filter_chains[0].filters[0].typed_config.route_config.virtual_hosts[0].routes[23].match'
{
  "case_sensitive": true,
  "headers": [
    {
      "exact_match": "echo.test",
      "name": ":authority"
    },
    {
      "exact_match": "https",
      "name": "x-forwarded-proto"
    }
  ],
  "prefix": "/",
  "runtime_fraction": {
    "default_value": {
      "denominator": "HUNDRED",
      "numerator": 100
    },
    "runtime_key": "routing.traffic_shift.cluster_echoserver_echoserver_echoserver"
  }
}
```

Later in the chain, there is a correct filter for my mTLS backend which accurately matches based on server name:

```
$ cat envoy.json|jq '.static_resources.listeners[0].filter_chains[2].filter_chain_match'
{
  "server_names": [
    "echo.test"
  ],
  "transport_protocol": "tls"
}
```

I think this explains the behavior I'm seeing but why does Ambassador do this? It appears to add every route to the default wildcard vhost and filter, which means you can bypass any client TLS requirement by either specifying a bogus SNI (or not including one) and manually supplying a Host header since the upstream route will still be there and match the vhost routes?

kflynn commented on Jun 11, 2021 — Contributor

Thanks for the report on mTLS, and thanks as well for the excellent reproducer!

I have confirmed your research: it is indeed possible to bypass mTLS with your configuration, in 1.7 and subsequent 1.X releases. The catchall behavior you describe is definitely part of it -- I won't bore you to death with the whole story, but the short version is that we sometimes have to infer things in 1.X that require the catchall. Your situation appears to be one where we take that too far.

Although there's no real workaround in 1.X, there are some mitigations. A malicious client doesn't actually get to bypass all of TLS: what they really get to do is switch from the TLS configuration dictated by SNI to the TLS configuration dictated by the `Host` header. That means that you can mitigate this by making sure that none of your `Host`s completely disable mTLS. Likewise, the rest of the authentication system is intact, so enabling external auth can also be a helpful mitigation.

Obviously, though, confusing mTLS in this way shouldn't be possible at all. We're investigating a fix for the 1.X family, and we have confirmed that our upcoming 2.0.0 release is not vulnerable to this issue at all. We'll be sure to keep you in the loop as we have more news. Thanks again for the report!

khussey assigned **kflynn** on Jun 15, 2021

khussey added this to the **2021 Cycle 4 Cool-down** milestone on Jun 15, 2021

khussey added the t:risk label on Jun 16, 2021

ropnop commented on Jun 21, 2021 — Author

Hi **@kflynn**

Thank you for the response. I have one clarifying technical question about the proposed mitigation: even if we ensure that none of our Hosts behind Ambassador disable mTLS, it would still be possible to confuse the client ca_secret for each Host's TLSContext right? So if Service A and Service B are both using mTLS but with different CAs, a client cert for Service A could be used to establish the connection to Ambassador via sending Service A's SNI but then talk to Service B via Host header mappings?

efunk modified the milestones: **2021 Cycle 4 Cool-down**, **2021 Cycle 5** on Jun 29, 2021

kflynn commented on Jul 7, 2021 — Contributor

@ropnop So first up, the developer preview of 2.0.0 did indeed ship, and it does fix this issue. Investigating a fix for 1.13 mostly turned up roadblocks, so we're not planning to backport to 1.13 at this point – the way it's addressed in 2.0 is really much better than how 1.13 handles things.

In 1.13, I think that you can confuse things as you describe only if you don't explicitly define a wildcard `Host` (where the `hostname` is `*`). 1.13 really wants there to always be a wildcard host, to the extent that if you don't define one, it will promote a `Host` to be the wildcard `Host` for you. If you explicitly define one, though, you can make its TLS config very restrictive, and set `prune_unreachable_hosts: true` in the `ambassador` module. Then:

- If SNI matches one of your `Host`s, that's the one whose TLS config will be used;
- If not, the wildcard `Host`'s TLS config will be used;
- `prune_unreachable_hosts: true` in the `ambassador` module should prevent non-wildcard `Host` from including routes for any other `Host`; and
- the restrictive TLS setup for the wildcard `Host` should prevent skipping in through a permissive setting and using `:authority` to ask for a more-restrictive `Host`.

kflynn commented on Jul 7, 2021 — Contributor

I'm going to go ahead and close this, but feel free to reopen if needed!

kflynn closed this as completed on Jul 7, 2021

Assignees

kflynn

Labels

t:risk

Projects

None yet

Milestone

2021 Cycle 5

Development

No branches or pull requests

4 participants