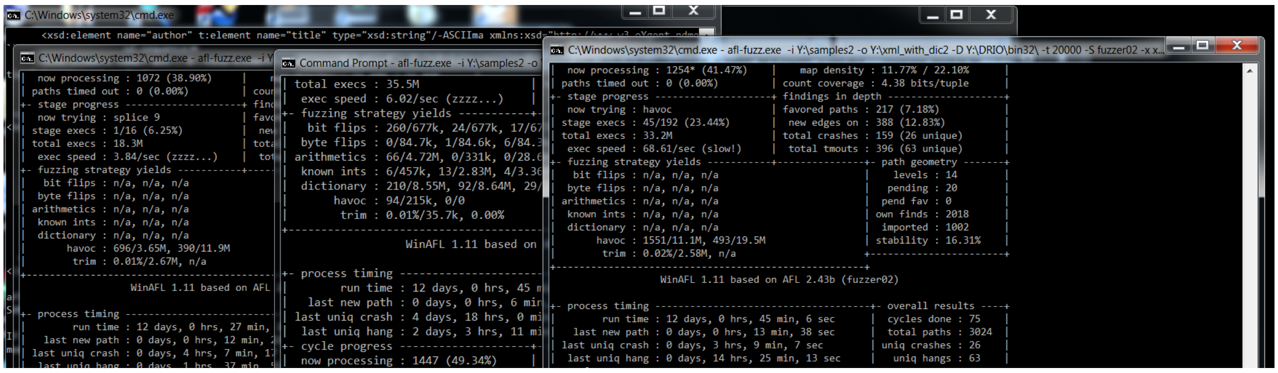
Fuzzing the MSXML6 library with WinAFL

symeonp.github.io/2017/09/17/fuzzing-winafl.html



C:\Windows\system32\cmd.exe

```
<xsd:element name="author" t:element name="title" type="xsd:string"/-ASCIIma xmlns:xsd="http://www.w3.oXgent.ndmo
```

C:\Windows\system32\cmd.exe - afl-fuzz.exe -i Y

```
| now processing : 1072 (38.90%)      |
| paths timed out : 0 (0.00%)         | cou
+- stage progress --------------------+ fin
| now trying : splice 9               | fav
| stage execs : 1/16 (6.25%)          |  new
| total execs : 18.3M                 | tota
| exec speed : 3.84/sec (zzzz...)     |  tot
+- fuzzing strategy yields -----------+----
|  bit flips : n/a, n/a, n/a
| byte flips : n/a, n/a, n/a
| arithmetics : n/a, n/a, n/a
| known ints : n/a, n/a, n/a
| dictionary : n/a, n/a, n/a
|      havoc : 696/3.65M, 390/11.9M
|       trim : 0.01%/2.67M, n/a
+-----------------------------------------
              WinAFL 1.11 based on AFL

+- process timing ------------------------
|       run time : 12 days, 0 hrs, 27 min,
|  last new path : 0 days, 0 hrs, 12 min,
| last uniq crash : 0 days, 4 hrs, 7 min, 1
```

Command Prompt - afl-fuzz.exe -i Y:\samples2 -o

```
| total execs : 35.5M                     |
|  exec speed : 6.02/sec (zzzz...)        |
+- fuzzing strategy yields -------------+-
|   bit flips : 260/677k, 24/677k, 17/67
|  byte flips : 0/84.7k, 1/84.6k, 6/84.3
| arithmetics : 66/4.72M, 0/331k, 0/28.6
|  known ints : 6/457k, 13/2.83M, 4/3.36
|  dictionary : 210/8.55M, 92/8.64M, 29/
|       havoc : 94/215k, 0/0
|        trim : 0.01%/35.7k, 0.00%
+-----------------------------------------
                   WinAFL 1.11 based on

+- process timing ------------------------
|       run time : 12 days, 0 hrs, 45 m
|  last new path : 0 days, 0 hrs, 6 min
| last uniq crash : 4 days, 18 hrs, 0 mi
| last uniq hang : 2 days, 3 hrs, 11 mi
+- cycle progress -------------------+-
| now processing : 1447 (49.34%)     |
```

C:\Windows\system32\cmd.exe - afl-fuzz.exe -i Y:\samples2 -o Y:\xml_with_dic2 -D Y:\DRIO\bin32\ -t 20000 -S fuzzer02 -x x...

```
| now processing : 1254* (41.47%)   |   map density : 11.77% / 22.10%   |
| paths timed out : 0 (0.00%)       | count coverage : 4.38 bits/tuple  |
+- stage progress ------------------+ findings in depth ------------------+
| now trying : havoc                | favored paths : 217 (7.18%)        |
| stage execs : 45/192 (23.44%)     |  new edges on : 388 (12.83%)       |
| total execs : 33.2M               | total crashes : 159 (26 unique)    |
| exec speed : 68.61/sec (slow!)    |  total tmouts : 396 (63 unique)    |
+- fuzzing strategy yields ----------+---------------+- path geometry -------+
|  bit flips : n/a, n/a, n/a                        |   levels : 14         |
| byte flips : n/a, n/a, n/a                        |  pending : 20         |
| arithmetics : n/a, n/a, n/a                       | pend fav : 0          |
| known ints : n/a, n/a, n/a                        | own finds : 2018      |
| dictionary : n/a, n/a, n/a                        |  imported : 1002      |
|      havoc : 1551/11.1M, 493/19.5M                | stability : 16.31%    |
|       trim : 0.02%/2.58M, n/a                     +-----------------------+
+---------------------------------------------------+
                WinAFL 1.11 based on AFL 2.43b (fuzzer02)

+- process timing -------------------------------------+- overall results ----+
|       run time : 12 days, 0 hrs, 45 min, 6 sec       |  cycles done : 75    |
|  last new path : 0 days, 0 hrs, 13 min, 38 sec      |  total paths : 3024  |
| last uniq crash : 0 days, 3 hrs, 9 min, 7 sec       | uniq crashes : 26    |
| last uniq hang : 0 days, 14 hrs, 25 min, 13 sec     |   uniq hangs : 63    |
```

Introduction

In this blog post, I'll write about how I tried to fuzz the MSXML library using the WinAFL fuzzer.

If you haven't played around with WinAFL, it's a massive fuzzer created by Ivan Fratric based on the lcumtuf's AFL which uses DynamoRIO to measure code coverage and the Windows API for memory and process creation. Axel Souchet has been actively contributing features such as corpus minimization, latest afl stable builds, persistent execution mode which will cover on the next blog post and the finally the afl-tmin tool.

We will start by creating a test harness which will allow us to fuzz some parsing functionality within the library, calculate the coverage, minimise the test cases and finish by kicking off the fuzzer and triage the findings. Lastly, thanks to Mitja Kolsek from 0patch for providing the patch which will see how one can use the 0patch to patch this issue!

Using the above steps, I've managed to find a NULL pointer dereference on the `msxml6!DTD::findEntityGeneral` function, which I reported to Microsoft but got rejected as this is not a security issue. Fair enough, indeed the crash is crap, yet hopefully somebody might find interesting the techniques I followed!

The Harness

While doing some research I ended up on this page which Microsoft has kindly provided a sample C++ code which allows us to feed some XML files and validate its structure. I am going to use Visual Studio 2015 to build the following program but before I do that, I am slightly going to modify it and use Ivan's charToWChar method so as to accept an argument as a file:

```
// xmlvalidate_fuzz.cpp : Defines the entry point for the console application.
//

#include "stdafx.h"
#include <stdio.h>
#include <tchar.h>
#include <windows.h>
#import <msxml6.dll>
extern "C" __declspec(dllexport)  int main(int argc, char** argv);

// Macro that calls a COM method returning HRESULT value.
#define CHK_HR(stmt)        do { hr=(stmt); if (FAILED(hr)) goto CleanUp; } while(0)

void dump_com_error(_com_error &e)
{
    _bstr_t bstrSource(e.Source());
    _bstr_t bstrDescription(e.Description());

    printf("Error\n");
    printf("\a\tCode = %08lx\n", e.Error());
    printf("\a\tCode meaning = %s", e.ErrorMessage());
    printf("\a\tSource = %s\n", (LPCSTR)bstrSource);
    printf("\a\tDescription = %s\n", (LPCSTR)bstrDescription);
}

_bstr_t validateFile(_bstr_t bstrFile)
{
    // Initialize objects and variables.
    MSXML2::IXMLDOMDocument2Ptr pXMLDoc;
    MSXML2::IXMLDOMParseErrorPtr pError;
    _bstr_t bstrResult = L"";
    HRESULT hr = S_OK;

    // Create a DOMDocument and set its properties.
    CHK_HR(pXMLDoc.CreateInstance(__uuidof(MSXML2::DOMDocument60), NULL,
CLSCTX_INPROC_SERVER));

    pXMLDoc->async = VARIANT_FALSE;
    pXMLDoc->validateOnParse = VARIANT_TRUE;
    pXMLDoc->resolveExternals = VARIANT_TRUE;

    // Load and validate the specified file into the DOM.
    // And return validation results in message to the user.
    if (pXMLDoc->load(bstrFile) != VARIANT_TRUE)
    {
        pError = pXMLDoc->parseError;

        bstrResult = _bstr_t(L"Validation failed on ") + bstrFile +
            _bstr_t(L"\n=====================") +
            _bstr_t(L"\nReason: ") + _bstr_t(pError->Getreason()) +
            _bstr_t(L"\nSource: ") + _bstr_t(pError->GetsrcText()) +
            _bstr_t(L"\nLine: ") + _bstr_t(pError->Getline()) +
            _bstr_t(L"\n");
    }
    else
    {
        bstrResult = _bstr_t(L"Validation succeeded for ") + bstrFile +
            _bstr_t(L"\n======================\n") +
            _bstr_t(pXMLDoc->xml) + _bstr_t(L"\n");
    }

CleanUp:
    return bstrResult;
}

wchar_t* charToWChar(const char* text)
{
    size_t size = strlen(text) + 1;
    wchar_t* wa = new wchar_t[size];
    mbstowcs(wa, text, size);
    return wa;
}

int main(int argc, char** argv)
{
    if (argc < 2) {
        printf("Usage: %s <xml file>\n", argv[0]);
        return 0;
    }

    HRESULT hr = CoInitialize(NULL);
    if (SUCCEEDED(hr))
    {
        try
        {
            _bstr_t bstrOutput = validateFile(charToWChar(argv[1]));
            MessageBoxW(NULL, bstrOutput, L"noNamespace", MB_OK);
        }
        catch (_com_error &e)
        {
            dump_com_error(e);
        }
        CoUninitialize();
    }

    return 0;

}
```

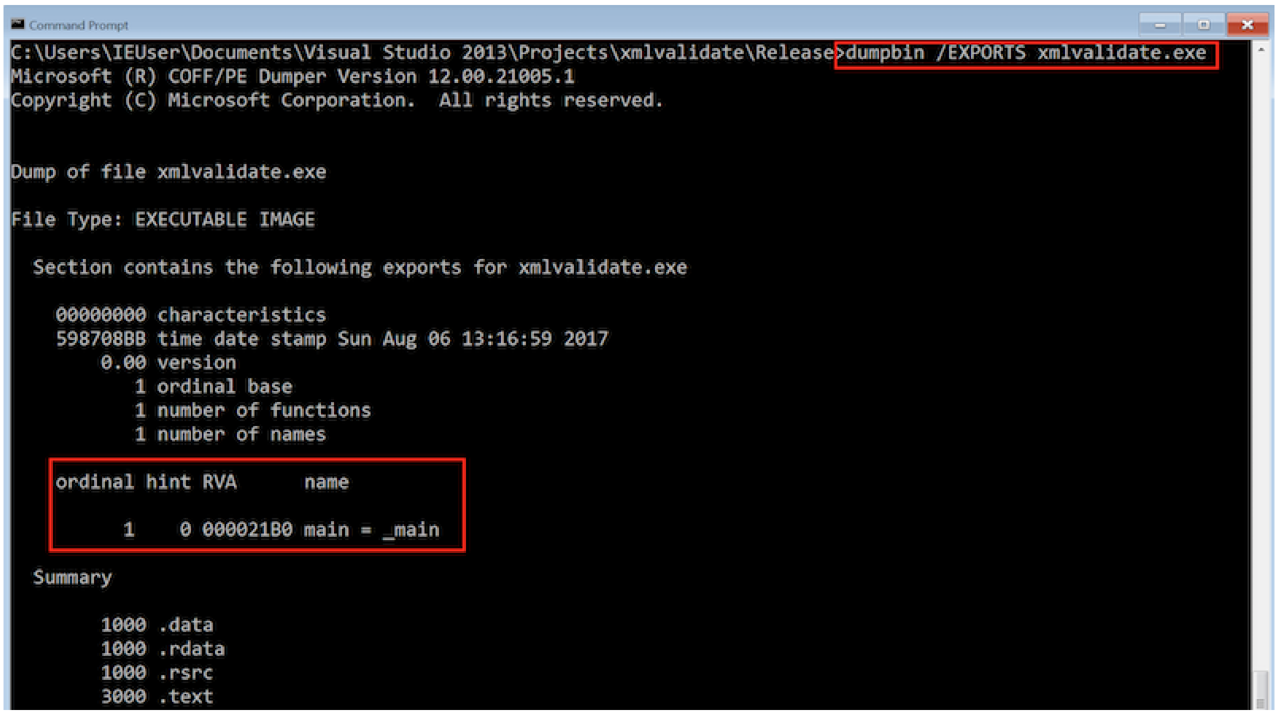
Notice also the following snippet: `extern "C" __declspec(dllexport) int main(int argc, char** argv);`

Essentially, this allows us to use `target_method` argument which DynamoRIO will try to retrieve the address for a given symbol name as seen here.

I could use the offsets method as per README, but due to ASLR and all that stuff, we want to scale a bit the fuzzing and spread the binary to many Virtual Machines and use the same commands to fuzz it. The `extern "C"` directive will unmangle the function name and will make it look prettier.

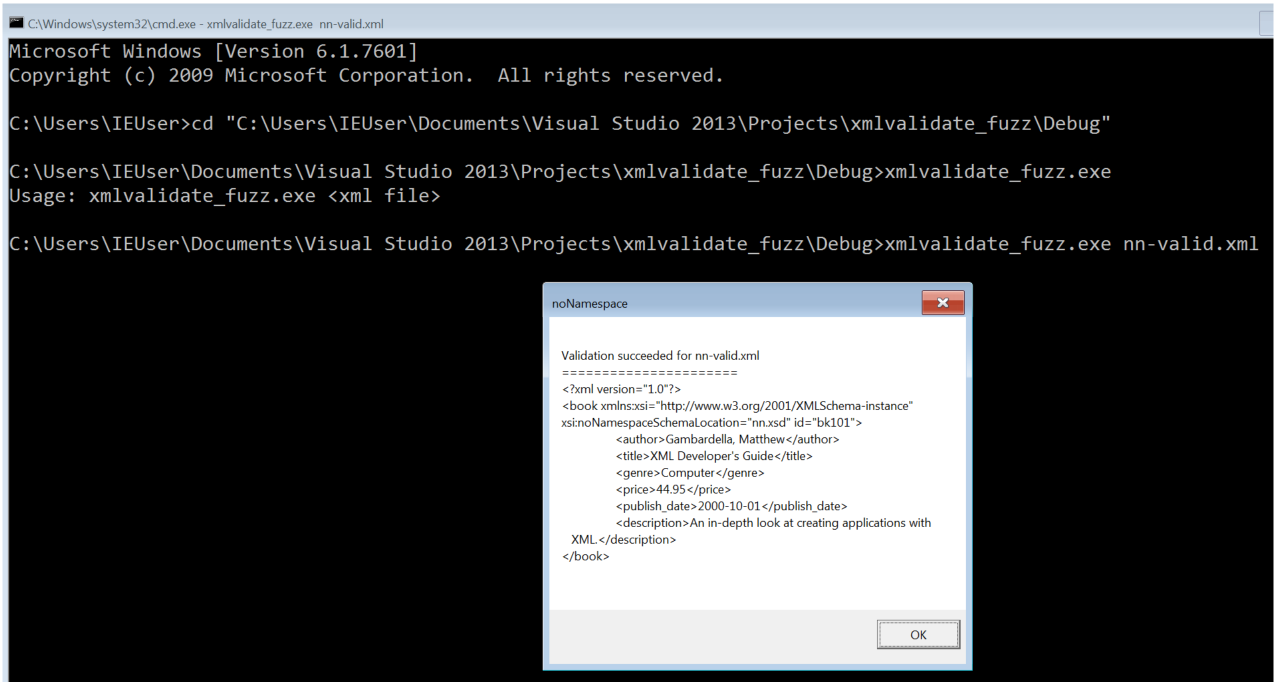
To confirm that indeed DynamoRIO can use this method the following command can be used:

```
dumpbin /EXPORTS xmlvalidate_fuzz.exe
```



```
Command Prompt
C:\Users\IEUser\Documents\Visual Studio 2013\Projects\xmlvalidate\Release>dumpbin /EXPORTS xmlvalidate.exe
Microsoft (R) COFF/PE Dumper Version 12.00.21005.1
Copyright (C) Microsoft Corporation.  All rights reserved.


Dump of file xmlvalidate.exe

File Type: EXECUTABLE IMAGE

  Section contains the following exports for xmlvalidate.exe

    00000000 characteristics
    598708BB time date stamp Sun Aug 06 13:16:59 2017
        0.00 version
           1 ordinal base
           1 number of functions
           1 number of names

    ordinal hint RVA      name

          1    0 000021B0 main = _main

  Summary

        1000 .data
        1000 .rdata
        1000 .rsrc
        3000 .text
```

Now let's quickly run the binary and observe the output. You should get the following output:



```
C:\Windows\system32\cmd.exe - xmlvalidate_fuzz.exe  nn-valid.xml
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\Users\IEUser>cd "C:\Users\IEUser\Documents\Visual Studio 2013\Projects\xmlvalidate_fuzz\Debug"

C:\Users\IEUser\Documents\Visual Studio 2013\Projects\xmlvalidate_fuzz\Debug>xmlvalidate_fuzz.exe
Usage: xmlvalidate_fuzz.exe <xml file>

C:\Users\IEUser\Documents\Visual Studio 2013\Projects\xmlvalidate_fuzz\Debug>xmlvalidate_fuzz.exe nn-valid.xml
```

```
noNamespace

Validation succeeded for nn-valid.xml
=====================
<?xml version="1.0"?>
<book xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="nn.xsd" id="bk101">
        <author>Gambardella, Matthew</author>
        <title>XML Developer's Guide</title>
        <genre>Computer</genre>
        <price>44.95</price>
        <publish_date>2000-10-01</publish_date>
        <description>An in-depth look at creating applications with
  XML.</description>
</book>

OK
```

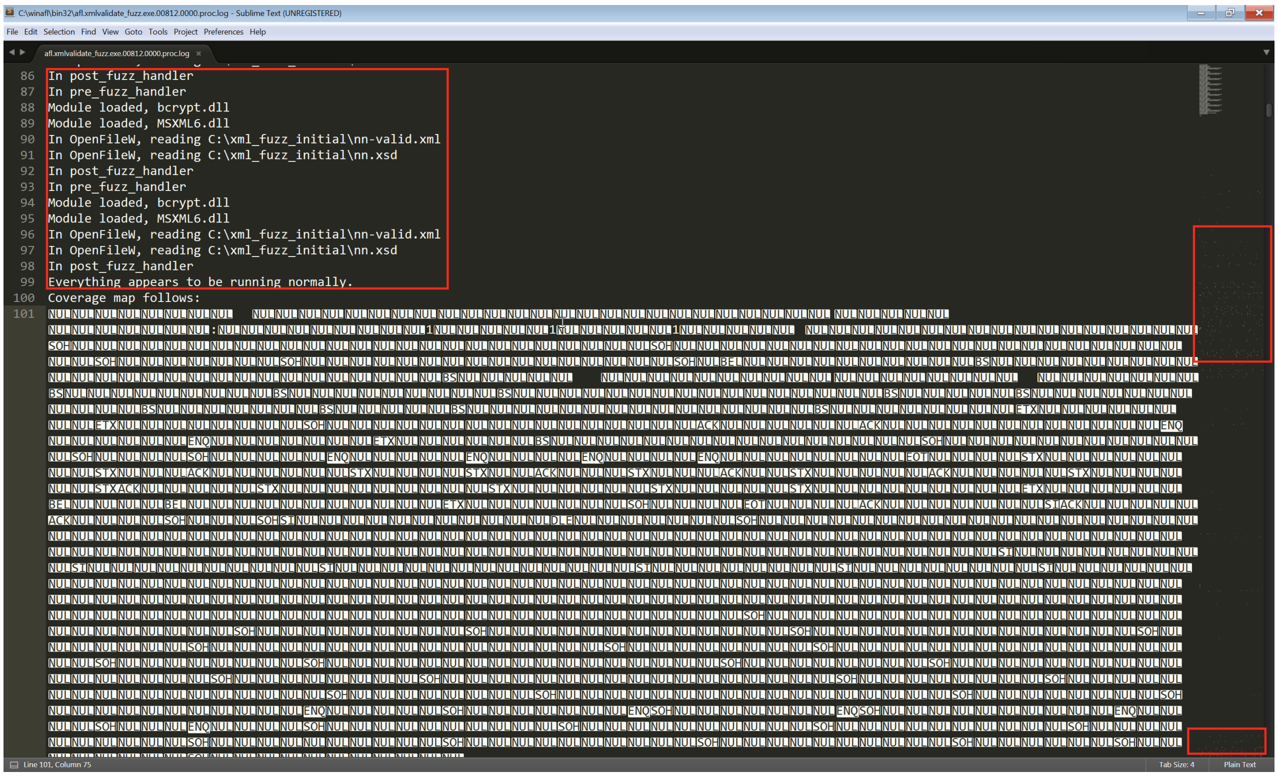
Code Coverage

WinAFL

Since the library is closed source, we will be using DynamoRIO's code coverage library feature via the WinAFL:

```
C:\DRIO\bin32\drrun.exe -c winafl.dll -debug -coverage_module msxml6.dll -target_module xmlvalidate.exe -target_method main -fuzz_iterations 10 -nargs 2 -- C:\xml_fuzz_initial\xmlvalidate.exe C:\xml_fuzz_initial\nn-valid.xml
```

WinAFL will start executing the binary ten times. Once this is done, navigate back to the winafl folder and check the log file:



```
In post_fuzz_handler
In pre_fuzz_handler
Module loaded, bcrypt.dll
Module loaded, MSXML6.dll
In OpenFileW, reading C:\xml_fuzz_initial\nn-valid.xml
In OpenFileW, reading C:\xml_fuzz_initial\nn.xsd
In post_fuzz_handler
In pre_fuzz_handler
Module loaded, bcrypt.dll
Module loaded, MSXML6.dll
In OpenFileW, reading C:\xml_fuzz_initial\nn-valid.xml
In OpenFileW, reading C:\xml_fuzz_initial\nn.xsd
In post_fuzz_handler
Everything appears to be running normally.
Coverage map follows:
```

C:\winafl\bin32\afl.xmlvalidate_fuzz.exe.00812.0000.proc.log - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help
afl.xmlvalidate_fuzz.exe.00812.0000.proc.log
Line 101, Column 75
Tab Size: 4
Plain Text

From the output we can see that everything appears to be running normally! On the right side of the file, the dots depict the coverage of the DLL, if you scroll down you'll see that we did hit many function as we are getting more dots throughout the whole file. That's a very good indication that we are hiting a lot of code and we properly targeting the **MSXML6** library.

Lighthouse - Code Coverage Explorer for IDA Pro

This plugin will help us understand better which function we are hitting and give a nice overview of the coverage using IDA. It's an excellent plugin with very good documentation and has been developed by Markus Gaasedelen (@gaasedelen) Make sure to download the latest DynamoRIO version 7, and install it as per instrcutions here. Luckily, we do have two sample test cases from the documentation, one valid and one invalid. Let's feed the valid one and observe the coverage. To do that, run the following command:

```
C:\DRIO7\bin64\drrun.exe -t drcov -- xmlvalidate.exe nn-valid.xml
```

Next step fire up IDA, drag the msxml6.dll and make sure to fetch the symbols! Now, check if a .log file has been created and open it on IDA from the **File -> Load File -> Code Coverage File(s)** menu. Once the coverage file is loaded it will highlight all the functions that your test case hit.

Case minimisation

Now it's time to grab some XML files (as small as possible). I've used a slightly hacked version of joxean's find_samples.py script. Once you get a few test cases let's minimise our initial seed files. This can be done using the following command:

```
python winafl-cmin.py --working-dir C:\winafl\bin32 -D C:\DRIO\bin32 -t 100000
-i C:\xml_fuzz\samples -o C:\minset_xml -coverage_module msxml6.dll -
target_module xmlvalidate.exe -target_method fuzzme -nargs 1 --
C:\xml_fuzz\xmlvalidate.exe @@
```

You might see the following output:

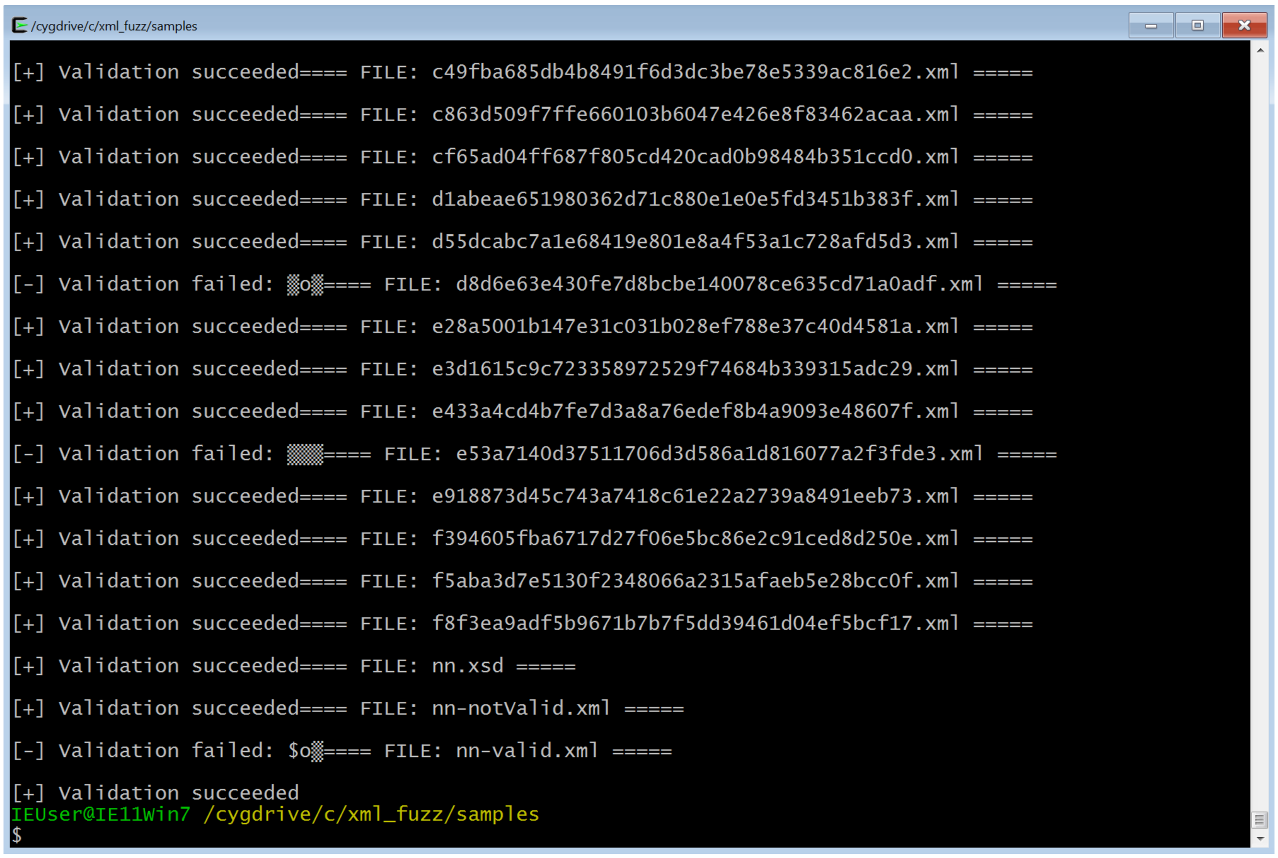
```
corpus minimization tool for WinAFL by <0vercl0k@tuxfamily.org>
Based on WinAFL by <ifratric@google.com>
Based on AFL by <lcamtuf@google.com>
[+] CWD changed to C:\winafl\bin32.
[*] Testing the target binary...
[!] Dry-run failed, 2 executions resulted differently:
Tuples matching? False
Return codes matching? True
```

I am not quite sure but I think that the **winafl-cmin.py** script expects that the initial seed files lead to the same code path, that is we have to run the script one time for the valid cases and one for the invalid ones. I might be wrong though and maybe there's a bug which in that case I need to ping Axel.

Let's identify the 'good' and the 'bad' XML test cases using this bash script:

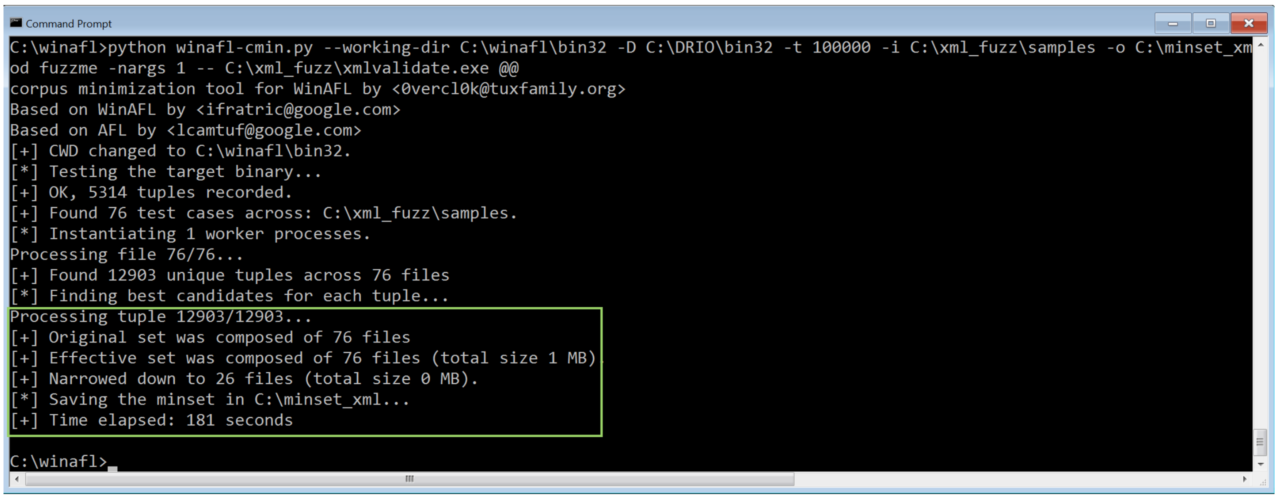
```
$ for file in *; do printf "==== FILE: $file =====\n";
/cygdrive/c/xml_fuzz/xmlvalidate.exe $file ;sleep 1; done
```

The following screenshot depicts my results:



```
[+] Validation succeeded==== FILE: c49fba685db4b8491f6d3dc3be78e5339ac816e2.xml =====
[+] Validation succeeded==== FILE: c863d509f7ffe660103b6047e426e8f83462acaa.xml =====
[+] Validation succeeded==== FILE: cf65ad04ff687f805cd420cad0b98484b351ccd0.xml =====
[+] Validation succeeded==== FILE: d1abeae651980362d71c880e1e0e5fd3451b383f.xml =====
[+] Validation succeeded==== FILE: d55dcabc7a1e68419e801e8a4f53a1c728afd5d3.xml =====
[-] Validation failed: ▒o▒==== FILE: d8d6e63e430fe7d8bcbe140078ce635cd71a0adf.xml =====
[+] Validation succeeded==== FILE: e28a5001b147e31c031b028ef788e37c40d4581a.xml =====
[+] Validation succeeded==== FILE: e3d1615c9c723358972529f74684b339315adc29.xml =====
[+] Validation succeeded==== FILE: e433a4cd4b7fe7d3a8a76edef8b4a9093e48607f.xml =====
[-] Validation failed: ▒▒▒==== FILE: e53a7140d37511706d3d586a1d816077a2f3fde3.xml =====
[+] Validation succeeded==== FILE: e918873d45c743a7418c61e22a2739a8491eeb73.xml =====
[+] Validation succeeded==== FILE: f394605fba6717d27f06e5bc86e2c91ced8d250e.xml =====
[+] Validation succeeded==== FILE: f5aba3d7e5130f2348066a2315afaeb5e28bcc0f.xml =====
[+] Validation succeeded==== FILE: f8f3ea9adf5b9671b7b7f5dd39461d04ef5bcf17.xml =====
[+] Validation succeeded==== FILE: nn.xsd =====
[+] Validation succeeded==== FILE: nn-notValid.xml =====
[-] Validation failed: $o▒==== FILE: nn-valid.xml =====
[+] Validation succeeded
IEUser@IE11Win7 /cygdrive/c/xml_fuzz/samples
$
```

Feel free to expirement a bit, and see which files are causing this issue - your mileage may vary. Once you are set, run again the above command and hopefully you'll get the following result:



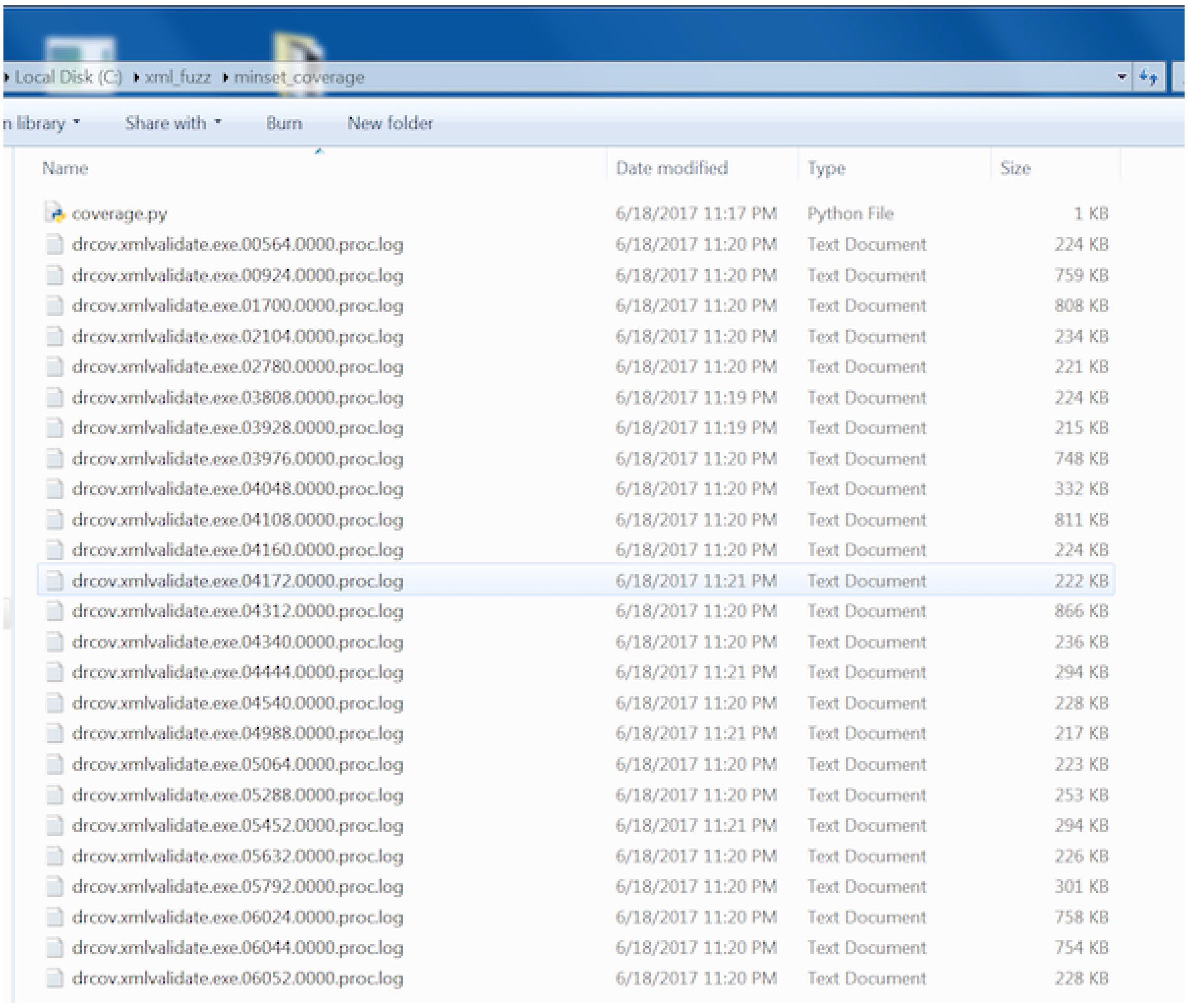
```
C:\winafl>python winafl-cmin.py --working-dir C:\winafl\bin32 -D C:\DRIO\bin32 -t 100000 -i C:\xml_fuzz\samples -o C:\minset_xm
od fuzzme -nargs 1 -- C:\xml_fuzz\xmlvalidate.exe @@
corpus minimization tool for WinAFL by <0vercl0k@tuxfamily.org>
Based on WinAFL by <ifratric@google.com>
Based on AFL by <lcamtuf@google.com>
[+] CWD changed to C:\winafl\bin32.
[*] Testing the target binary...
[+] OK, 5314 tuples recorded.
[+] Found 76 test cases across: C:\xml_fuzz\samples.
[*] Instantiating 1 worker processes.
Processing file 76/76...
[+] Found 12903 unique tuples across 76 files
[*] Finding best candidates for each tuple...
Processing tuple 12903/12903...
[+] Original set was composed of 76 files
[+] Effective set was composed of 76 files (total size 1 MB).
[+] Narrowed down to 26 files (total size 0 MB).
[*] Saving the minset in C:\minset_xml...
[+] Time elapsed: 181 seconds

C:\winafl>
```

So look at that! The initial campaign included 76 cases which after the minimisation it was narrowed down to 26.
Thank you Axel!

With the minimised test cases let's code a python script that will automate all the code coverage:

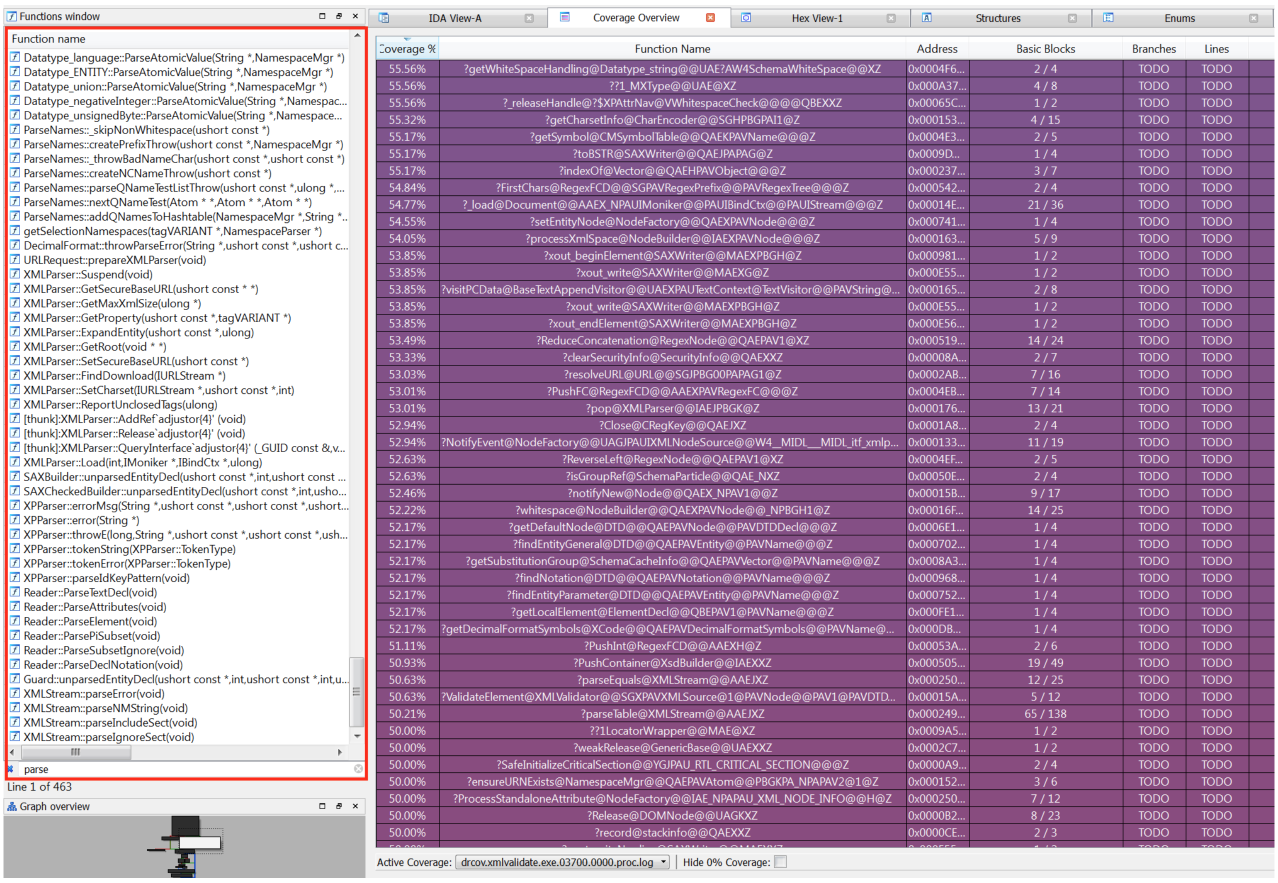
```
import sys
import os

testcases = []
for root, dirs, files in os.walk(".", topdown=False):
    for name in files:
        if name.endswith(".xml"):
            testcase =  os.path.abspath(os.path.join(root, name))
            testcases.append(testcase)

for testcase in testcases:
    print "[*] Running DynamoRIO for testcase: ", testcase
    os.system("C:\\DRIO7\\bin32\\drrun.exe -t drcov -- C:\\xml_fuzz\\xmlvalidate.exe %s"
% testcase)
```

The above script produced the following output for my case:



Name	Date modified	Type	Size
coverage.py	6/18/2017 11:17 PM	Python File	1 KB
drcov.xmlvalidate.exe.00564.0000.proc.log	6/18/2017 11:20 PM	Text Document	224 KB
drcov.xmlvalidate.exe.00924.0000.proc.log	6/18/2017 11:20 PM	Text Document	759 KB
drcov.xmlvalidate.exe.01700.0000.proc.log	6/18/2017 11:20 PM	Text Document	808 KB
drcov.xmlvalidate.exe.02104.0000.proc.log	6/18/2017 11:20 PM	Text Document	234 KB
drcov.xmlvalidate.exe.02780.0000.proc.log	6/18/2017 11:20 PM	Text Document	221 KB
drcov.xmlvalidate.exe.03808.0000.proc.log	6/18/2017 11:19 PM	Text Document	224 KB
drcov.xmlvalidate.exe.03928.0000.proc.log	6/18/2017 11:19 PM	Text Document	215 KB
drcov.xmlvalidate.exe.03976.0000.proc.log	6/18/2017 11:20 PM	Text Document	748 KB
drcov.xmlvalidate.exe.04048.0000.proc.log	6/18/2017 11:20 PM	Text Document	332 KB
drcov.xmlvalidate.exe.04108.0000.proc.log	6/18/2017 11:20 PM	Text Document	811 KB
drcov.xmlvalidate.exe.04160.0000.proc.log	6/18/2017 11:20 PM	Text Document	224 KB
drcov.xmlvalidate.exe.04172.0000.proc.log	6/18/2017 11:21 PM	Text Document	222 KB
drcov.xmlvalidate.exe.04312.0000.proc.log	6/18/2017 11:20 PM	Text Document	866 KB
drcov.xmlvalidate.exe.04340.0000.proc.log	6/18/2017 11:20 PM	Text Document	236 KB
drcov.xmlvalidate.exe.04444.0000.proc.log	6/18/2017 11:21 PM	Text Document	294 KB
drcov.xmlvalidate.exe.04540.0000.proc.log	6/18/2017 11:20 PM	Text Document	228 KB
drcov.xmlvalidate.exe.04988.0000.proc.log	6/18/2017 11:21 PM	Text Document	217 KB
drcov.xmlvalidate.exe.05064.0000.proc.log	6/18/2017 11:20 PM	Text Document	223 KB
drcov.xmlvalidate.exe.05288.0000.proc.log	6/18/2017 11:20 PM	Text Document	253 KB
drcov.xmlvalidate.exe.05452.0000.proc.log	6/18/2017 11:21 PM	Text Document	294 KB
drcov.xmlvalidate.exe.05632.0000.proc.log	6/18/2017 11:20 PM	Text Document	226 KB
drcov.xmlvalidate.exe.05792.0000.proc.log	6/18/2017 11:20 PM	Text Document	301 KB
drcov.xmlvalidate.exe.06024.0000.proc.log	6/18/2017 11:20 PM	Text Document	758 KB
drcov.xmlvalidate.exe.06044.0000.proc.log	6/18/2017 11:20 PM	Text Document	754 KB
drcov.xmlvalidate.exe.06052.0000.proc.log	6/18/2017 11:20 PM	Text Document	228 KB

As previously, using IDA open all those .log files under **File -> Load File -> Code Coverage File(s)** menu.



```
Function name
Datatype_language::ParseAtomicValue(String *,NamespaceMgr *)
Datatype_ENTITY::ParseAtomicValue(String *,NamespaceMgr *)
Datatype_union::ParseAtomicValue(String *,NamespaceMgr *)
Datatype_negativeInteger::ParseAtomicValue(String *,Namespac...
Datatype_unsignedByte::ParseAtomicValue(String *,Namespace...
ParseNames::_skipNonWhitespace(ushort const *)
ParseNames::createPrefixThrow(ushort const *,NamespaceMgr *)
ParseNames::_throwBadNameChar(ushort const *,ushort const *)
ParseNames::createNCNameThrow(ushort const *)
ParseNames::parseQNameTestListThrow(ushort const *,ulong *,...
ParseNames::nextQNameTest(Atom * *,Atom * *,Atom * *)
ParseNames::addQNamesToHashtable(NamespaceMgr *,String *..
getSelectionNamespaces(tagVARIANT *,NamespaceParser *)
DecimalFormat::throwParseError(String *,ushort const *,ushort c...
URLRequest::prepareXMLParser(void)
XMLParser::Suspend(void)
XMLParser::GetSecureBaseURL(ushort const * *)
XMLParser::GetMaxXmlSize(ulong *)
XMLParser::GetProperty(ushort const *,tagVARIANT *)
XMLParser::ExpandEntity(ushort const *,ulong)
XMLParser::GetRoot(void * *)
XMLParser::SetSecureBaseURL(ushort const *)
XMLParser::FindDownload(IURLStream *)
XMLParser::SetCharset(IURLStream *,ushort const *,int)
XMLParser::ReportUnclosedTags(ulong)
[thunk]:XMLParser::AddRef`adjustor{4}' (void)
[thunk]:XMLParser::Release`adjustor{4}' (void)
[thunk]:XMLParser::QueryInterface`adjustor{4}' (_GUID const &,v...
XMLParser::Load(int,IMoniker *,IBindCtx *,ulong)
SAXBuilder::unparsedEntityDecl(ushort const *,int,ushort const ...
SAXCheckedBuilder::unparsedEntityDecl(ushort const *,int,usho...
XPParser::errorMsg(String *,ushort const *,ushort const *,ushort...
XPParser::error(String *)
XPParser::throwE(long,String *,ushort const *,ushort const *,ush...
XPParser::tokenString(XPParser::TokenType)
XPParser::tokenError(XPParser::TokenType)
XPParser::parseIdKeyPattern(void)
Reader::ParseTextDecl(void)
Reader::ParseAttributes(void)
Reader::ParseElement(void)
Reader::ParsePiSubset(void)
Reader::ParseSubsetIgnore(void)
Reader::ParseDeclNotation(void)
Guard::unparsedEntityDecl(ushort const *,int,ushort const *,int,u...
XMLStream::parseError(void)
XMLStream::parseNMString(void)
XMLStream::parseIncludeSect(void)
XMLStream::parseIgnoreSect(void)
parse
Line 1 of 463
```

Coverage %	Function Name	Address	Basic Blocks	Branches	Lines
55.56%	?getWhiteSpaceHandling@Datatype_string@@UAE?AW4SchemaWhiteSpace@@XZ	0x0004F6...	2 / 4	TODO	TODO
55.56%	??1_MXType@@UAE@XZ	0x000A37...	4 / 8	TODO	TODO
55.56%	?_releaseHandle@?$XPAttrNav@VWhitespaceCheck@@@@QBEXXZ	0x00065C...	1 / 2	TODO	TODO
55.32%	?getCharsetInfo@CharEncoder@@SGHPBGPAI1@Z	0x000153...	4 / 15	TODO	TODO
55.17%	?getSymbol@CMSymbolTable@@QAEKPAVName@@@Z	0x0004E3...	2 / 5	TODO	TODO
55.17%	?toBSTR@SAXWriter@@QAEJPAPAG@Z	0x0009D...	1 / 4	TODO	TODO
55.17%	?indexOf@Vector@@QAEHPAVObject@@@Z	0x000237...	3 / 7	TODO	TODO
54.84%	?FirstChars@RegexFCD@@SGPAVRegexPrefix@@PAVRegexTree@@@Z	0x000542...	2 / 4	TODO	TODO
54.77%	?_load@Document@@AAEX_NPAUIMoniker@@PAUIBindCtx@@PAUIStream@@@Z	0x00014E...	21 / 36	TODO	TODO
54.55%	?setEntityNode@NodeFactory@@QAEXPAVNode@@@Z	0x000741...	1 / 4	TODO	TODO
54.05%	?processXmlSpace@NodeBuilder@@IAEXPAVNode@@@Z	0x000163...	5 / 9	TODO	TODO
53.85%	?xout_beginElement@SAXWriter@@MAEXPBGH@Z	0x000981...	1 / 2	TODO	TODO
53.85%	?xout_write@SAXWriter@@MAEXG@Z	0x000E55...	1 / 2	TODO	TODO
53.85%	?visitPCData@BaseTextAppendVisitor@@UAEXPAUTextContext@TextVisitor@@PAVString@...	0x000165...	2 / 8	TODO	TODO
53.85%	?xout_write@SAXWriter@@MAEXPBGH@Z	0x000E55...	1 / 2	TODO	TODO
53.85%	?xout_endElement@SAXWriter@@MAEXPBGH@Z	0x000E56...	1 / 2	TODO	TODO
53.49%	?ReduceConcatenation@RegexNode@@QAEPAV1@XZ	0x000519...	14 / 24	TODO	TODO
53.33%	?clearSecurityInfo@SecurityInfo@@QAEXXZ	0x00008A...	2 / 7	TODO	TODO
53.03%	?resolveURL@URL@@SGJPBG00PAPAG1@Z	0x0002AB...	7 / 16	TODO	TODO
53.01%	?PushFC@RegexFCD@@AAEXPAVRegexFC@@@Z	0x0004EB...	7 / 14	TODO	TODO
53.01%	?pop@XMLParser@@IAEJPBGK@Z	0x000176...	13 / 21	TODO	TODO
52.94%	?Close@CRegKey@@QAEJXZ	0x0001A8...	2 / 4	TODO	TODO
52.94%	?NotifyEvent@NodeFactory@@UAGJPAUIXMLNodeSource@@W4__MIDL___MIDL_itf_xmlp...	0x000133...	11 / 19	TODO	TODO
52.63%	?ReverseLeft@RegexNode@@QAEPAV1@XZ	0x0004EF...	2 / 5	TODO	TODO
52.63%	?isGroupRef@SchemaParticle@@QAE_NXZ	0x00050E...	2 / 4	TODO	TODO
52.46%	?notifyNew@Node@@QAEX_NPAV1@@Z	0x00015B...	9 / 17	TODO	TODO
52.22%	?whitespace@NodeBuilder@@QAEXPAVNode@@_NPBGH1@Z	0x00016F...	14 / 25	TODO	TODO
52.17%	?getDefaultNode@DTD@@QAEPAVNode@@PAVDTDDecl@@@Z	0x0006E1...	1 / 4	TODO	TODO
52.17%	?findEntityGeneral@DTD@@QAEPAVEntity@@PAVName@@@Z	0x000702...	1 / 4	TODO	TODO
52.17%	?getSubstitutionGroup@SchemaCacheInfo@@QAEPAVVector@@PAVName@@@Z	0x0008A3...	1 / 4	TODO	TODO
52.17%	?findNotation@DTD@@QAEPAVNotation@@PAVName@@@Z	0x000968...	1 / 4	TODO	TODO
52.17%	?findEntityParameter@DTD@@QAEPAVEntity@@PAVName@@@Z	0x000752...	1 / 4	TODO	TODO
52.17%	?getLocalElement@ElementDecl@@QBEPAV1@PAVName@@@Z	0x000FE1...	1 / 4	TODO	TODO
52.17%	?getDecimalFormatSymbols@XCode@@QAEPAVDecimalFormatSymbols@@PAVName@...	0x000DB...	1 / 4	TODO	TODO
51.11%	?PushInt@RegexFCD@@AAEXH@Z	0x00053A...	2 / 6	TODO	TODO
50.93%	?PushContainer@XsdBuilder@@IAEXXZ	0x000505...	19 / 49	TODO	TODO
50.63%	?parseEquals@XMLStream@@AAEJXZ	0x000250...	12 / 25	TODO	TODO
50.63%	?ValidateElement@XMLValidator@@SGXPAVXMLSource@1@PAVNode@@PAV1@PAVDTD...	0x00015A...	5 / 12	TODO	TODO
50.21%	?parseTable@XMLStream@@AAEJXZ	0x000249...	65 / 138	TODO	TODO
50.00%	??1LocatorWrapper@@MAE@XZ	0x0009A5...	1 / 2	TODO	TODO
50.00%	?weakRelease@GenericBase@@UAEXXZ	0x0002C7...	1 / 2	TODO	TODO
50.00%	?SafeInitializeCriticalSection@@YGJPAU_RTL_CRITICAL_SECTION@@@Z	0x0000A9...	2 / 4	TODO	TODO
50.00%	?ensureURNExists@NamespaceMgr@@QAEPAVAtom@@PBGKPA_NPAPAV2@1@Z	0x000152...	3 / 6	TODO	TODO
50.00%	?ProcessStandaloneAttribute@NodeFactory@@IAE_NPAPAU_XML_NODE_INFO@@H@Z	0x000250...	7 / 12	TODO	TODO
50.00%	?Release@DOMNode@@UAGKXZ	0x0000B2...	8 / 23	TODO	TODO
50.00%	?record@stackinfo@@QAEXXZ	0x0000CE...	2 / 3	TODO	TODO

Active Coverage: drcov.xmlvalidate.exe.03700.0000.proc.log | Hide 0% Coverage: ☐

Interestingly enough, notice how many **parse** functions do exist, and if you navigate around the coverage you'll see that we've managed to hit a decent amount of interesting code.

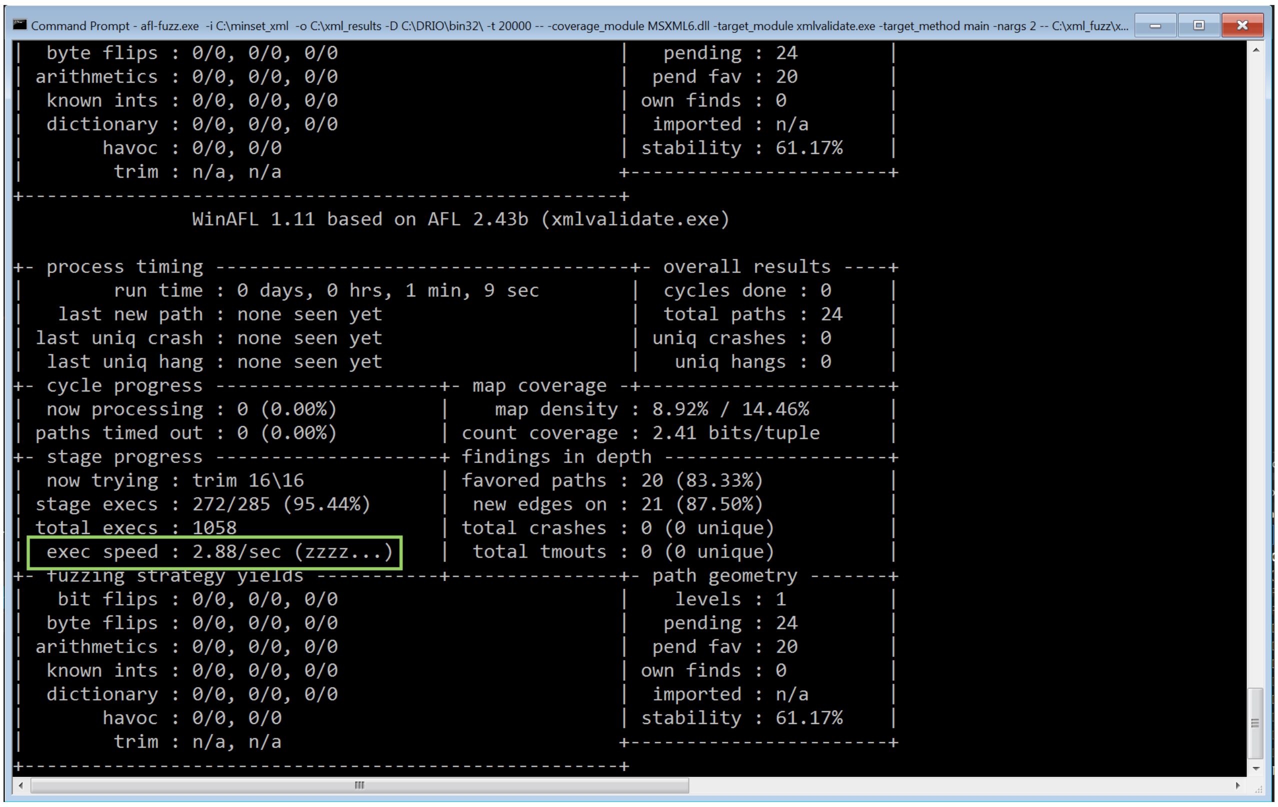
Since we do have some decent coverage, let's move on and finally fuzz it!

All I do is fuzz, fuzz, fuzz

Let's kick off the fuzzer:

```
afl-fuzz.exe -i C:\minset_xml -o C:\xml_results -D C:\DRIO\bin32\ -t 20000 -- -coverage_module MSXML6.dll -target_module xmlvalidate.exe -target_method main -nargs 2 -- C:\xml_fuzz\xmlvalidate.exe @@
```

Running the above yields the following output:



```
Command Prompt - afl-fuzz.exe  -i C:\minset_xml  -o C:\xml_results -D C:\DRIO\bin32\ -t 20000 -- -coverage_module MSXML6.dll -target_module xmlvalidate.exe -target_method main -nargs 2 -- C:\xml_fuzz\x...
|  byte flips : 0/0, 0/0, 0/0                        |   pending : 24         |
| arithmetics : 0/0, 0/0, 0/0                        |  pend fav : 20         |
|  known ints : 0/0, 0/0, 0/0                        | own finds : 0          |
|  dictionary : 0/0, 0/0, 0/0                        |  imported : n/a        |
|       havoc : 0/0, 0/0                             | stability : 61.17%     |
|        trim : n/a, n/a                             +-----------------------+
+-----------------------------------------------------+
                WinAFL 1.11 based on AFL 2.43b (xmlvalidate.exe)

+- process timing -------------------------------------+- overall results ----+
|        run time : 0 days, 0 hrs, 1 min, 9 sec        |  cycles done : 0     |
|   last new path : none seen yet                      |  total paths : 24    |
| last uniq crash : none seen yet                      | uniq crashes : 0     |
|  last uniq hang : none seen yet                      |   uniq hangs : 0     |
+- cycle progress --------------------+- map coverage -+----------------------+
|  now processing : 0 (0.00%)         |    map density : 8.92% / 14.46%       |
| paths timed out : 0 (0.00%)         | count coverage : 2.41 bits/tuple      |
+- stage progress --------------------+ findings in depth --------------------+
|  now trying : trim 16\16            | favored paths : 20 (83.33%)           |
| stage execs : 272/285 (95.44%)      |  new edges on : 21 (87.50%)           |
| total execs : 1058                  | total crashes : 0 (0 unique)          |
|  exec speed : 2.88/sec (zzzz...)    |  total tmouts : 0 (0 unique)          |
+- fuzzing strategy yields -----------+---------------+- path geometry -------+
|   bit flips : 0/0, 0/0, 0/0                        |    levels : 1          |
|  byte flips : 0/0, 0/0, 0/0                        |   pending : 24         |
| arithmetics : 0/0, 0/0, 0/0                        |  pend fav : 20         |
|  known ints : 0/0, 0/0, 0/0                        | own finds : 0          |
|  dictionary : 0/0, 0/0, 0/0                        |  imported : n/a        |
|       havoc : 0/0, 0/0                             | stability : 61.17%     |
|        trim : n/a, n/a                             +-----------------------+
+-----------------------------------------------------+
```

As you can see, the initial code does that job - however the speed is very slow. Three executions per second will take long to give some proper results. Interestingly enough, I've had luck in the past and with that speed (using python and radamsa prior the afl/winafl era) had success in finding bugs and within three days of fuzzing!

Let's try our best though and get rid of the part that slows down the fuzzing. If you've done some Windows programming you know that the following line initialises a COM object which could be the bottleneck of the slow speed:

```
HRESULT hr = CoInitialize(NULL);
```

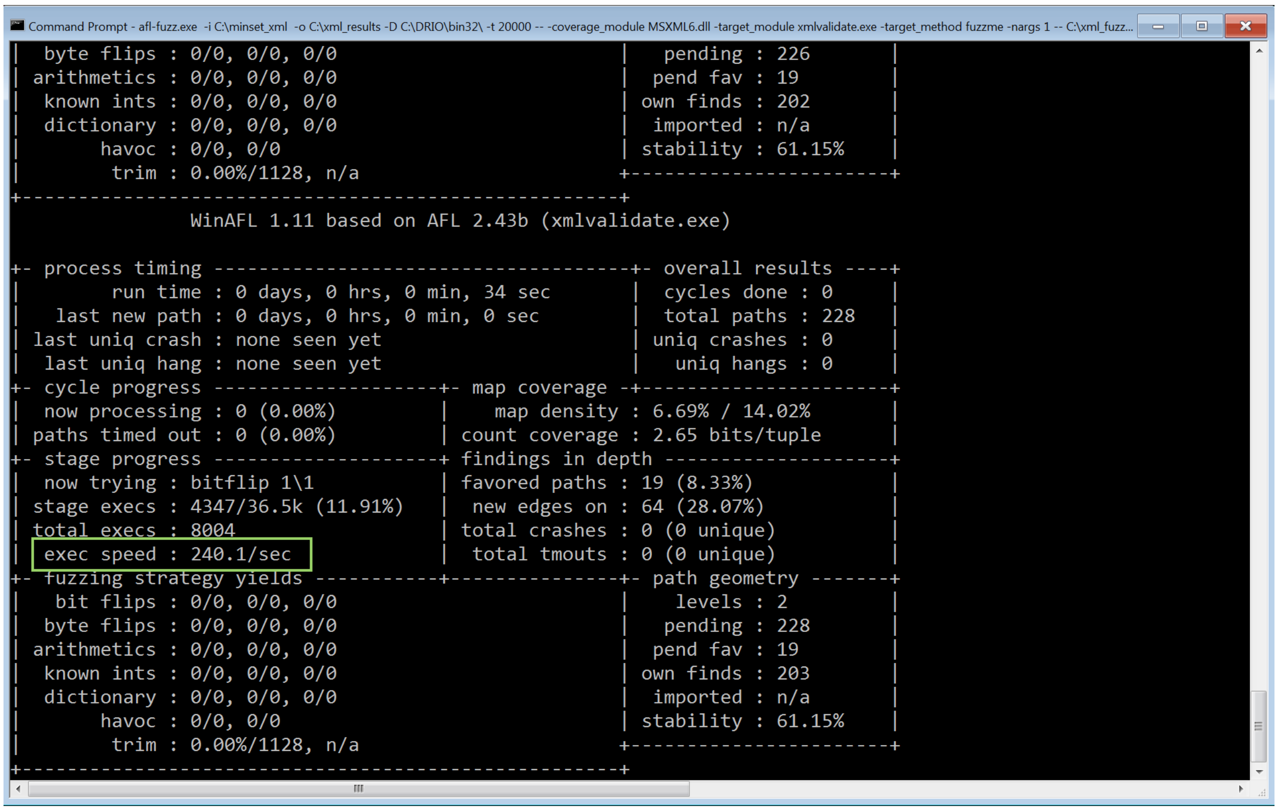
This line probably is a major issue so in fact, let's refactor the code, we are going to create a `fuzzme` method which is going to receive the filename as an argument outside the COM initialisation call. The refactored code should look like this:

```
--- cut ---

extern "C" __declspec(dllexport) _bstr_t fuzzme(wchar_t*
filename);

_bstr_t fuzzme(wchar_t* filename)
{
    _bstr_t bstrOutput = validateFile(filename);
    //bstrOutput += validateFile(L"nn-notValid.xml");
    //MessageBoxW(NULL, bstrOutput, L"noNamespace", MB_OK);
    return bstrOutput;

}
int main(int argc, char** argv)
{
    if (argc < 2) {
        printf("Usage: %s <xml file>\n", argv[0]);
        return 0;
    }

    HRESULT hr = CoInitialize(NULL);
    if (SUCCEEDED(hr))
    {
        try
        {
            _bstr_t bstrOutput = fuzzme(charToWChar(argv[1]));
        }
        catch (_com_error &e)
        {
            dump_com_error(e);
        }
        CoUninitialize();
    }
    return 0;
}
--- cut ---
```

You can grab the refactored version here. With the refactored binary let's run one more time the fuzzer and see if we were right. This time, we will pass the **fuzzme** target_method instead of main, and use only one argument which is the filename. While we are here, let's use the lcamtuf's xml.dic from here.

```
 afl-fuzz.exe -i C:\minset_xml -o C:\xml_results -D C:\DRIO\bin32\ -t 20000 -x
xml.dict -- -coverage_module MSXML6.dll -target_module xmlvalidate.exe -
target_method fuzzme -nargs 1 -- C:\xml_fuzz\xmlvalidate.exe @@
```

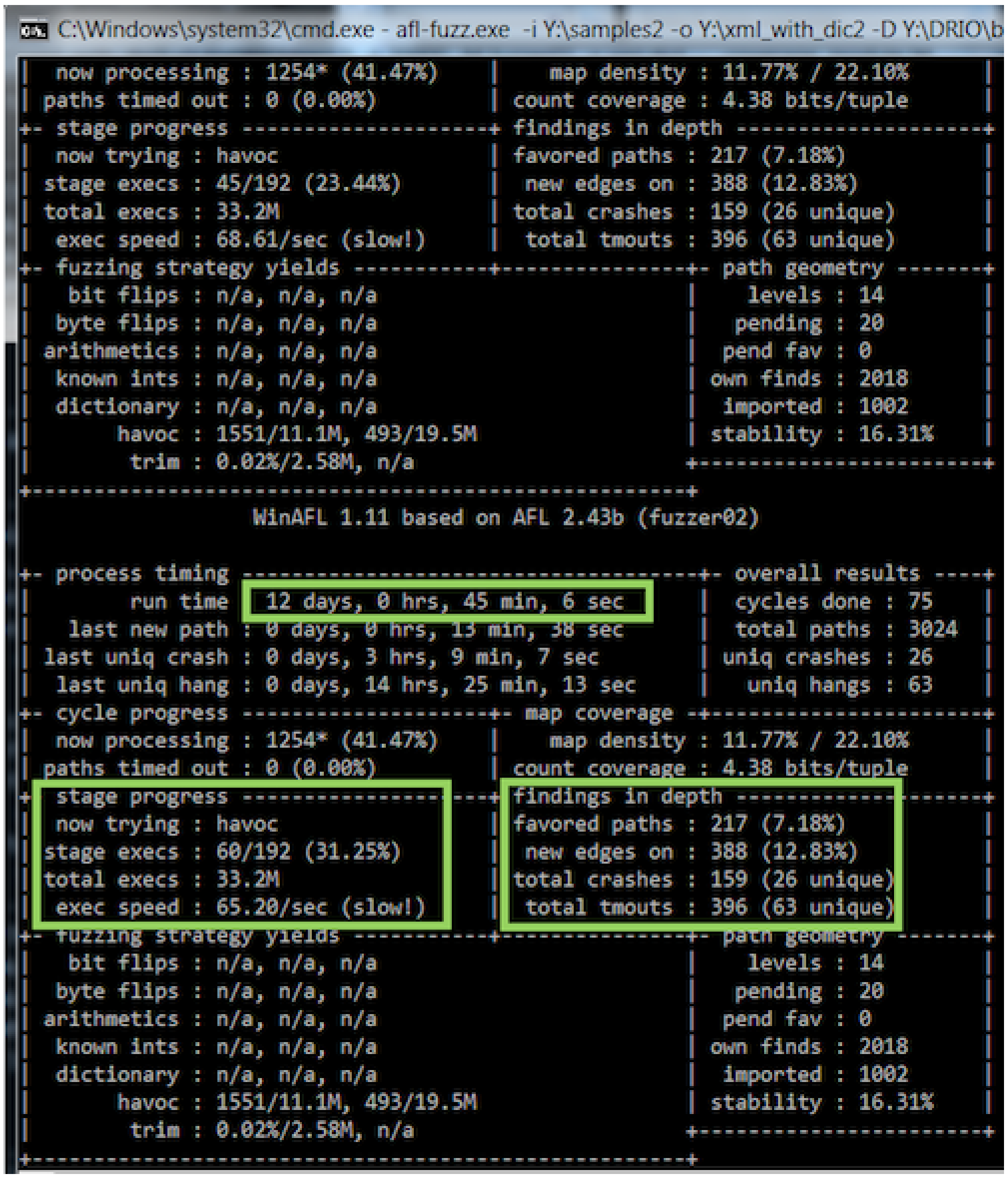
Once you've run that, here's the output within a few seconds of fuzzing on a VMWare instance:


```
Command Prompt - afl-fuzz.exe  -i C:\minset_xml  -o C:\xml_results -D C:\DRIO\bin32\ -t 20000 -- -coverage_module MSXML6.dll -target_module xmlvalidate.exe -target_method fuzzme -nargs 1 -- C:\xml_fuzz...
|  byte flips : 0/0, 0/0, 0/0                          |   pending : 226        |
| arithmetics : 0/0, 0/0, 0/0                          |  pend fav : 19         |
|  known ints : 0/0, 0/0, 0/0                          | own finds : 202        |
|  dictionary : 0/0, 0/0, 0/0                          |  imported : n/a        |
|       havoc : 0/0, 0/0                               | stability : 61.15%     |
|        trim : 0.00%/1128, n/a                        +-----------------------+
+-----------------------------------------------------+
                WinAFL 1.11 based on AFL 2.43b (xmlvalidate.exe)

+- process timing -------------------------------------+- overall results ----+
|        run time : 0 days, 0 hrs, 0 min, 34 sec       |  cycles done : 0     |
|   last new path : 0 days, 0 hrs, 0 min, 0 sec        |  total paths : 228   |
| last uniq crash : none seen yet                      | uniq crashes : 0     |
|  last uniq hang : none seen yet                      |   uniq hangs : 0     |
+- cycle progress --------------------+- map coverage -+----------------------+
|  now processing : 0 (0.00%)         |    map density : 6.69% / 14.02%       |
| paths timed out : 0 (0.00%)         | count coverage : 2.65 bits/tuple      |
+- stage progress --------------------+ findings in depth --------------------+
|  now trying : bitflip 1\1           | favored paths : 19 (8.33%)            |
| stage execs : 4347/36.5k (11.91%)   |  new edges on : 64 (28.07%)           |
| total execs : 8004                  | total crashes : 0 (0 unique)          |
|  exec speed : 240.1/sec             |  total tmouts : 0 (0 unique)          |
+- fuzzing strategy yields -----------+---------------+- path geometry -------+
|   bit flips : 0/0, 0/0, 0/0                          |    levels : 2         |
|  byte flips : 0/0, 0/0, 0/0                          |   pending : 228       |
| arithmetics : 0/0, 0/0, 0/0                          |  pend fav : 19        |
|  known ints : 0/0, 0/0, 0/0                          | own finds : 203       |
|  dictionary : 0/0, 0/0, 0/0                          |  imported : n/a       |
|       havoc : 0/0, 0/0                               | stability : 61.15%    |
|        trim : 0.00%/1128, n/a                        +-----------------------+
+-----------------------------------------------------+
```

Brilliant! That's much much better, now let it run and wait for crashes!

The findings - Crash triage/analysis

Generally, I've tried to fuzz this binary with different test cases, however unfortunately I kept getting the NULL pointer dereference bug. The following screenshot depicts the findings after a ~ 12 days fuzzing campaign:



```
C:\Windows\system32\cmd.exe - afl-fuzz.exe  -i Y:\samples2 -o Y:\xml_with_dic2 -D Y:\DRIO\b
|  now processing : 1254* (41.47%)     |    map density : 11.77% / 22.10%      |
| paths timed out : 0 (0.00%)          | count coverage : 4.38 bits/tuple      |
+- stage progress --------------------+ findings in depth --------------------+
|  now trying : havoc                  | favored paths : 217 (7.18%)           |
| stage execs : 45/192 (23.44%)        |  new edges on : 388 (12.83%)          |
| total execs : 33.2M                  | total crashes : 159 (26 unique)       |
|  exec speed : 68.61/sec (slow!)      |  total tmouts : 396 (63 unique)       |
+- fuzzing strategy yields -----------+---------------+- path geometry -------+
|   bit flips : n/a, n/a, n/a                          |    levels : 14        |
|  byte flips : n/a, n/a, n/a                          |   pending : 20        |
| arithmetics : n/a, n/a, n/a                          |  pend fav : 0         |
|  known ints : n/a, n/a, n/a                          | own finds : 2018      |
|  dictionary : n/a, n/a, n/a                          |  imported : 1002      |
|       havoc : 1551/11.1M, 493/19.5M                  | stability : 16.31%    |
|        trim : 0.02%/2.58M, n/a                       +-----------------------+
+-----------------------------------------------------+
                   WinAFL 1.11 based on AFL 2.43b (fuzzer02)

+- process timing -------------------------------------+- overall results ----+
|        run time : 12 days, 0 hrs, 45 min, 6 sec      |  cycles done : 75     |
|   last new path : 0 days, 0 hrs, 13 min, 38 sec      |  total paths : 3024   |
| last uniq crash : 0 days, 3 hrs, 9 min, 7 sec        | uniq crashes : 26     |
|  last uniq hang : 0 days, 14 hrs, 25 min, 13 sec     |   uniq hangs : 63     |
+- cycle progress --------------------+- map coverage -+----------------------+
|  now processing : 1254* (41.47%)     |    map density : 11.77% / 22.10%      |
| paths timed out : 0 (0.00%)          | count coverage : 4.38 bits/tuple      |
+- stage progress --------------------+ findings in depth --------------------+
|  now trying : havoc                  | favored paths : 217 (7.18%)           |
| stage execs : 60/192 (31.25%)        |  new edges on : 388 (12.83%)          |
| total execs : 33.2M                  | total crashes : 159 (26 unique)       |
|  exec speed : 65.20/sec (slow!)      |  total tmouts : 396 (63 unique)       |
+- fuzzing strategy yields -----------+---------------+- path geometry -------+
|   bit flips : n/a, n/a, n/a                          |    levels : 14        |
|  byte flips : n/a, n/a, n/a                          |   pending : 20        |
| arithmetics : n/a, n/a, n/a                          |  pend fav : 0         |
|  known ints : n/a, n/a, n/a                          | own finds : 2018      |
|  dictionary : n/a, n/a, n/a                          |  imported : 1002      |
|       havoc : 1551/11.1M, 493/19.5M                  | stability : 16.31%    |
|        trim : 0.02%/2.58M, n/a                       +-----------------------+
+-----------------------------------------------------+
```

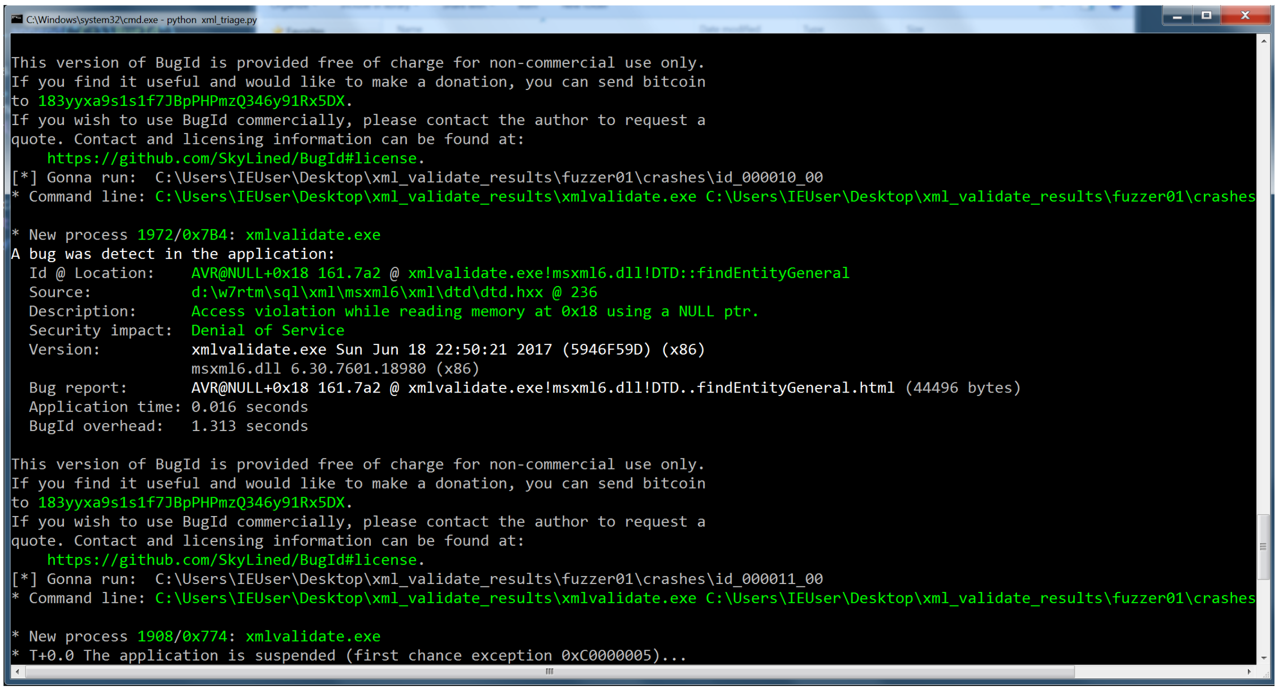
Notice that a total of 33 million executions were performed and 26 unique crashes were discovered!

In order to triage these findings, I’ve used the BugId tool from SkyLined, it’s an excellent tool which will give you a detailed report regarding the crash and the exploitability of the crash.

Here’s my python code for that:

```
import sys
import os


sys.path.append("C:\\BugId")

testcases = []
for root, dirs, files in os.walk(".\\fuzzer01\\crashes", topdown=False):
    for name in files:
        if name.endswith("00"):
            testcase =  os.path.abspath(os.path.join(root, name))
            testcases.append(testcase)

for testcase in testcases:
    print "[*] Gonna run: ", testcase
    os.system("C:\\python27\\python.exe C:\\BugId\\BugId.py
C:\\Users\\IEUser\\Desktop\\xml_validate_results\\xmlvalidate.exe -- %s" % testcase)
```

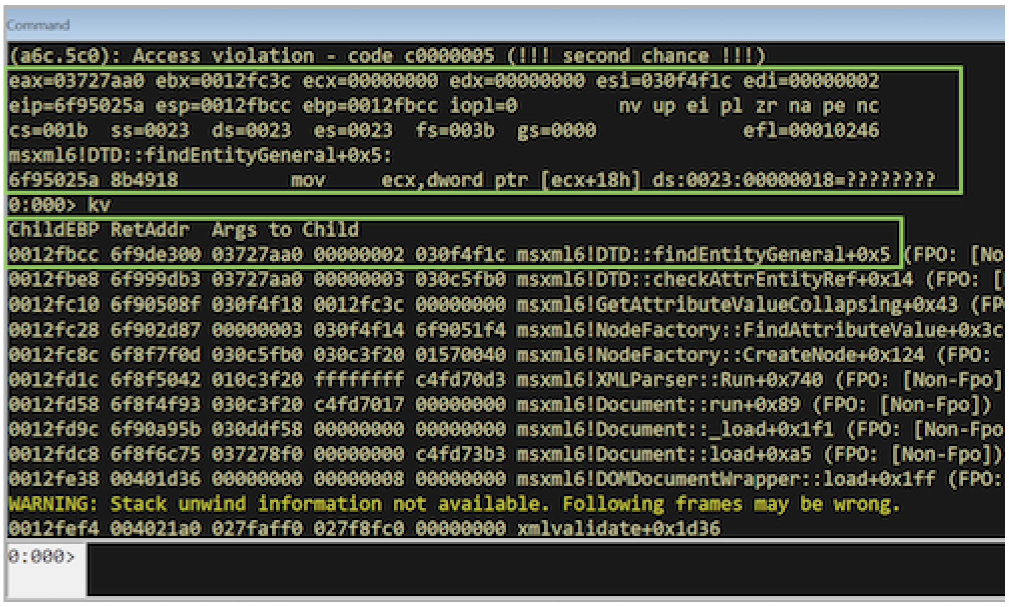
The above script gives the following output:



```
C:\Windows\system32\cmd.exe - python  xml_triage.py

This version of BugId is provided free of charge for non-commercial use only.
If you find it useful and would like to make a donation, you can send bitcoin
to 183yyxa9s1s1f7JBpPHPmzQ346y91Rx5DX.
If you wish to use BugId commercially, please contact the author to request a
quote. Contact and licensing information can be found at:
    https://github.com/SkyLined/BugId#license.
[*] Gonna run:  C:\Users\IEUser\Desktop\xml_validate_results\fuzzer01\crashes\id_000010_00
* Command line: C:\Users\IEUser\Desktop\xml_validate_results\xmlvalidate.exe C:\Users\IEUser\Desktop\xml_validate_results\fuzzer01\crashes

* New process 1972/0x7B4: xmlvalidate.exe
A bug was detect in the application:
  Id @ Location:    AVR@NULL+0x18 161.7a2 @ xmlvalidate.exe!msxml6.dll!DTD::findEntityGeneral
  Source:           d:\w7rtm\sql\xml\msxml6\xml\dtd\dtd.hxx @ 236
  Description:      Access violation while reading memory at 0x18 using a NULL ptr.
  Security impact:  Denial of Service
  Version:          xmlvalidate.exe Sun Jun 18 22:50:21 2017 (5946F59D) (x86)
                    msxml6.dll 6.30.7601.18980 (x86)
  Bug report:       AVR@NULL+0x18 161.7a2 @ xmlvalidate.exe!msxml6.dll!DTD..findEntityGeneral.html (44496 bytes)
  Application time: 0.016 seconds
  BugId overhead:   1.313 seconds

This version of BugId is provided free of charge for non-commercial use only.
If you find it useful and would like to make a donation, you can send bitcoin
to 183yyxa9s1s1f7JBpPHPmzQ346y91Rx5DX.
If you wish to use BugId commercially, please contact the author to request a
quote. Contact and licensing information can be found at:
    https://github.com/SkyLined/BugId#license.
[*] Gonna run:  C:\Users\IEUser\Desktop\xml_validate_results\fuzzer01\crashes\id_000011_00
* Command line: C:\Users\IEUser\Desktop\xml_validate_results\xmlvalidate.exe C:\Users\IEUser\Desktop\xml_validate_results\fuzzer01\crashes

* New process 1908/0x774: xmlvalidate.exe
* T+0.0 The application is suspended (first chance exception 0xC0000005)...
```

Once I ran that for all my crashes, it clearly showed that we’re hitting the same bug. To confirm, let’s fire up windbg:

```
0:000> g
(a6c.5c0): Access violation - code c0000005 (!!! second chance !!!)
eax=03727aa0 ebx=0012fc3c ecx=00000000 edx=00000000 esi=030f4f1c edi=00000002
eip=6f95025a esp=0012fbcc ebp=0012fbcc iopl=0         nv up ei pl zr na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00010246
msxml6!DTD::findEntityGeneral+0x5:
6f95025a 8b4918          mov     ecx,dword ptr [ecx+18h] ds:0023:00000018=????????
0:000> kv
ChildEBP RetAddr  Args to Child
0012fbcc 6f9de300 03727aa0 00000002 030f4f1c msxml6!DTD::findEntityGeneral+0x5 (FPO:
[Non-Fpo]) (CONV: thiscall) [d:\w7rtm\sql\xml\msxml6\xml\dtd\dtd.hxx @ 236]
0012fbe8 6f999db3 03727aa0 00000003 030c5fb0 msxml6!DTD::checkAttrEntityRef+0x14 (FPO:
[Non-Fpo]) (CONV: thiscall) [d:\w7rtm\sql\xml\msxml6\xml\dtd\dtd.cxx @ 1470]
0012fc10 6f90508f 030f4f18 0012fc3c 00000000 msxml6!GetAttributeValueCollapsing+0x43
(FPO: [Non-Fpo]) (CONV: stdcall) [d:\w7rtm\sql\xml\msxml6\xml\parse\nodefactory.cxx @
771]
0012fc28 6f902d87 00000003 030f4f14 6f9051f4 msxml6!NodeFactory::FindAttributeValue+0x3c
(FPO: [Non-Fpo]) (CONV: thiscall) [d:\w7rtm\sql\xml\msxml6\xml\parse\nodefactory.cxx @
743]
0012fc8c 6f8f7f0d 030c5fb0 030c3f20 01570040 msxml6!NodeFactory::CreateNode+0x124 (FPO:
[Non-Fpo]) (CONV: stdcall) [d:\w7rtm\sql\xml\msxml6\xml\parse\nodefactory.cxx @ 444]
0012fd1c 6f8f5042 010c3f20 ffffffff c4fd70d3 msxml6!XMLParser::Run+0x740 (FPO: [Non-
Fpo]) (CONV: stdcall) [d:\w7rtm\sql\xml\msxml6\xml\tokenizer\parser\xmlparser.cxx @
1165]
0012fd58 6f8f4f93 030c3f20 c4fd7017 00000000 msxml6!Document::run+0x89 (FPO: [Non-Fpo])
(CONV: thiscall) [d:\w7rtm\sql\xml\msxml6\xml\om\document.cxx @ 1494]
0012fd9c 6f90a95b 030ddf58 00000000 00000000 msxml6!Document::_load+0x1f1 (FPO: [Non-
Fpo]) (CONV: thiscall) [d:\w7rtm\sql\xml\msxml6\xml\om\document.cxx @ 1012]
0012fdc8 6f8f6c75 037278f0 00000000 c4fd73b3 msxml6!Document::load+0xa5 (FPO: [Non-Fpo])
(CONV: thiscall) [d:\w7rtm\sql\xml\msxml6\xml\om\document.cxx @ 754]
0012fe38 00401d36 00000000 00000008 00000000 msxml6!DOMDocumentWrapper::load+0x1ff (FPO:
[Non-Fpo]) (CONV: stdcall) [d:\w7rtm\sql\xml\msxml6\xml\om\xmldom.cxx @ 1111]
-- cut --
```



```
Command
(a6c.5c0): Access violation - code c0000005 (!!! second chance !!!)
eax=03727aa0 ebx=0012fc3c ecx=00000000 edx=00000000 esi=030f4f1c edi=00000002
eip=6f95025a esp=0012fbcc ebp=0012fbcc iopl=0         nv up ei pl zr na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00010246
msxml6!DTD::findEntityGeneral+0x5:
6f95025a 8b4918          mov     ecx,dword ptr [ecx+18h] ds:0023:00000018=????????
0:000> kv
ChildEBP RetAddr  Args to Child
0012fbcc 6f9de300 03727aa0 00000002 030f4f1c msxml6!DTD::findEntityGeneral+0x5 (FPO: [No
0012fbe8 6f999db3 03727aa0 00000003 030c5fb0 msxml6!DTD::checkAttrEntityRef+0x14 (FPO: [
0012fc10 6f90508f 030f4f18 0012fc3c 00000000 msxml6!GetAttributeValueCollapsing+0x43 (FP
0012fc28 6f902d87 00000003 030f4f14 6f9051f4 msxml6!NodeFactory::FindAttributeValue+0x3c
0012fc8c 6f8f7f0d 030c5fb0 030c3f20 01570040 msxml6!NodeFactory::CreateNode+0x124 (FPO:
0012fd1c 6f8f5042 010c3f20 ffffffff c4fd70d3 msxml6!XMLParser::Run+0x740 (FPO: [Non-Fpo]
0012fd58 6f8f4f93 030c3f20 c4fd7017 00000000 msxml6!Document::run+0x89 (FPO: [Non-Fpo])
0012fd9c 6f90a95b 030ddf58 00000000 00000000 msxml6!Document::_load+0x1f1 (FPO: [Non-Fpo
0012fdc8 6f8f6c75 037278f0 00000000 c4fd73b3 msxml6!Document::load+0xa5 (FPO: [Non-Fpo])
0012fe38 00401d36 00000000 00000008 00000000 msxml6!DOMDocumentWrapper::load+0x1ff (FPO:
WARNING: Stack unwind information not available. Following frames may be wrong.
0012fef4 004021a0 027faff0 027f8fc0 00000000 xmlvalidate+0x1d36
0:000>
```

Let’s take a look at one of the crasher:

```
C:\Users\IEUser\Desktop\xml_validate_results\fuzzer01\crashes>type id_000000_00
<?xml version="&a;1.0"?>
<book xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:noNamespaceSchemaLocation="nn.xsd"
      id="bk101">
   <author>Gambardella, Matthew</author>
   <title>XML Developer's Guide</title>
   <genre>Computer</genre>
   <price>44.95</price>
   <publish_date>2000-10-01</publish_date>
   <description>An in-depth look at creating applications with
   XML.</description>
```

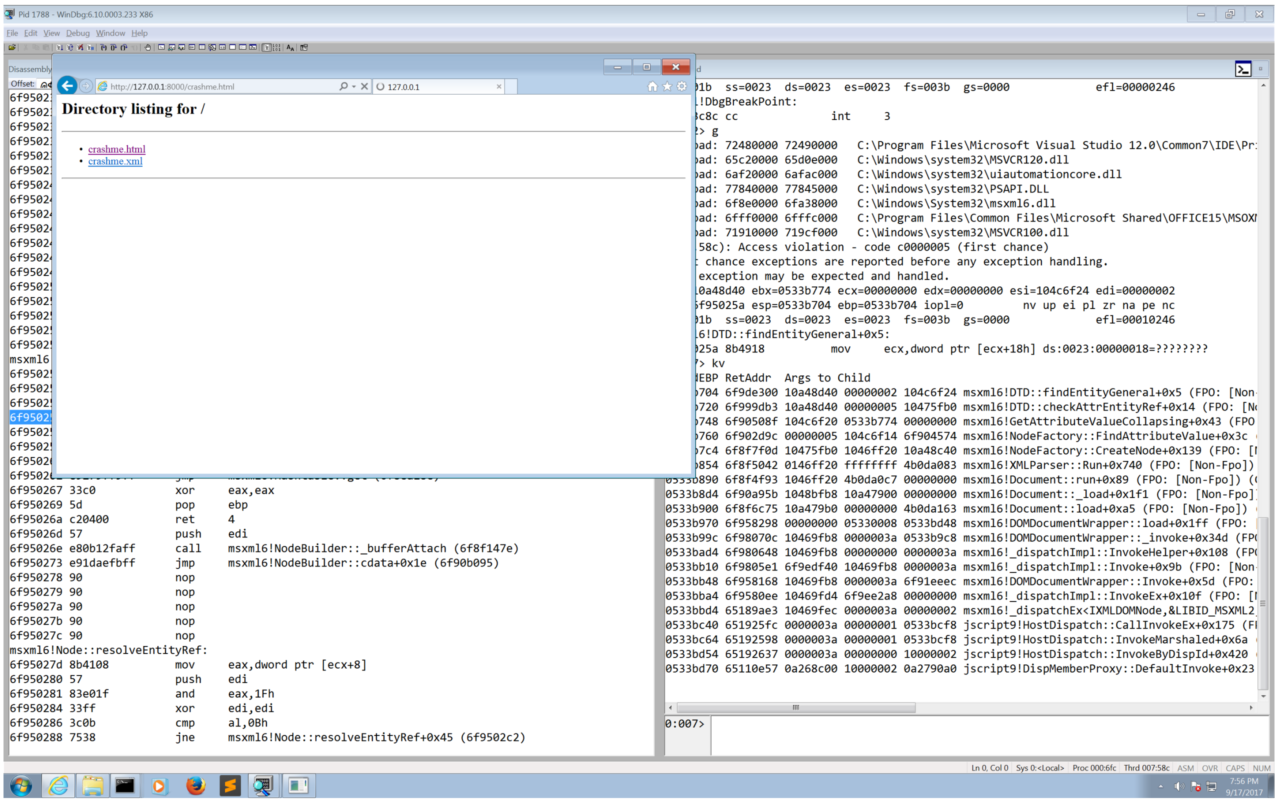
As you can see, if we provide some garbage either on the xml version or the encoding, we will get the above crash. Mitja also minimised the case as seen below:

```
<?xml version='1.0' encoding='&aaa;'?>
```

The whole idea of fuzzing this library was based on finding a vulnerability within Internet Explorer's context and somehow trigger it. After a bit of googling, let's use the following PoC (**crashme.html**) and see if it will crash IE11:

```
<!DOCTYPE html>
<html>
<head>
</head>
<body>
<script>

var xmlDoc = new ActiveXObject("Msxml2.DOMDocument.6.0");
xmlDoc.async = false;
xmlDoc.load("crashme.xml");
if (xmlDoc.parseError.errorCode != 0) {
   var myErr = xmlDoc.parseError;
   console.log("You have error " + myErr.reason);
} else {
   console.log(xmlDoc.xml);
}

</script>
</body>
</html>
```

Running that under Python's SimpleHTTPServer gives the following:


Pid 1788 - WinDbg:6.10.0003.233 X86
File Edit View Debug Window Help
Disassembly
Offset:
http://127.0.0.1:8000/crashme.html
127.0.0.1

Directory listing for /

- crashme.html
- crashme.xml

```
6f9502
6f9502
6f9502
6f9502
6f9502
6f9502
6f9502
6f9502
6f9502
6f9502
6f9502
6f9502
6f9502
6f9502
6f9502
6f9502
msxml6
6f9502
6f9502
6f9502
6f9502
6f9502
6f9502
6f9502
6f9502
6f950267 33c0            xor     eax,eax
6f950269 5d              pop     ebp
6f95026a c20400          ret     4
6f95026d 57              push    edi
6f95026e e80b12faff      call    msxml6!NodeBuilder::_bufferAttach (6f8f147e)
6f950273 e91daefbff      jmp     msxml6!NodeBuilder::cdata+0x1e (6f90b095)
6f950278 90              nop
6f950279 90              nop
6f95027a 90              nop
6f95027b 90              nop
6f95027c 90              nop
msxml6!Node::resolveEntityRef:
6f95027d 8b4108          mov     eax,dword ptr [ecx+8]
6f950280 57              push    edi
6f950281 83e01f          and     eax,1Fh
6f950284 33ff            xor     edi,edi
6f950286 3c0b            cmp     al,0Bh
6f950288 7538            jne     msxml6!Node::resolveEntityRef+0x45 (6f9502c2)
```

```
1b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000246
!DbgBreakPoint:
c8c cc              int     3
> g
ad: 72480000 72490000   C:\Program Files\Microsoft Visual Studio 12.0\Common7\IDE\Pr
ad: 65c20000 65d0e000   C:\Windows\system32\MSVCR120.dll
ad: 6af20000 6afac000   C:\Windows\system32\uiautomationcore.dll
ad: 77840000 77845000   C:\Windows\system32\PSAPI.DLL
ad: 6f8e0000 6fa38000   C:\Windows\System32\msxml6.dll
ad: 6fff0000 6fffc000   C:\Program Files\Common Files\Microsoft Shared\OFFICE15\MSOX
ad: 71910000 719cf000   C:\Windows\system32\MSVCR100.dll
58c): Access violation - code c0000005 (first chance)
 chance exceptions are reported before any exception handling.
 exception may be expected and handled.
0a48d40 ebx=0533b774 ecx=00000000 edx=00000000 esi=104c6f24 edi=00000002
f95025a esp=0533b704 ebp=0533b704 iopl=0         nv up ei pl zr na pe nc
1b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00010246
6!DTD::findEntityGeneral+0x5:
25a 8b4918          mov     ecx,dword ptr [ecx+18h] ds:0023:00000018=????????
> kv
EBP RetAddr  Args to Child
704 6f9de300 10a48d40 00000002 104c6f24 msxml6!DTD::findEntityGeneral+0x5 (FPO: [Non
720 6f999db3 10a48d40 00000005 10475fb0 msxml6!DTD::checkAttrEntityRef+0x14 (FPO: [N
748 6f90508f 104c6f20 0533b774 00000000 msxml6!GetAttributeValueCollapsing+0x43 (FPO
760 6f902d9c 00000005 104c6f14 6f904574 msxml6!NodeFactory::FindAttributeValue+0x3c
7c4 6f8f7f0d 10475fb0 1046ff20 10a48c40 msxml6!NodeFactory::CreateNode+0x139 (FPO: [
854 6f8f5042 0146ff20 ffffffff 4b0da083 msxml6!XMLParser::Run+0x740 (FPO: [Non-Fpo])
0533b890 6f8f4f93 1046ff20 4b0da0c7 00000000 msxml6!Document::run+0x89 (FPO: [Non-Fpo]) (
0533b8d4 6f90a95b 1048bfb8 10a47900 00000000 msxml6!Document::_load+0x1f1 (FPO: [Non-Fpo]
0533b900 6f8f6c75 10a479b0 00000000 4b0da163 msxml6!Document::load+0xa5 (FPO: [Non-Fpo])
0533b970 6f958298 00000000 05330008 0533bd48 msxml6!DOMDocumentWrapper::load+0x1ff (FPO:
0533b99c 6f98070c 10469fb8 0000003a 0533b9c8 msxml6!DOMDocumentWrapper::_invoke+0x34d (FP
0533bad4 6f980648 10469fb8 00000000 0000003a msxml6!_dispatchImpl::InvokeHelper+0x108 (FP
0533bb10 6f9805e1 6f9edf40 10469fb8 0000003a msxml6!_dispatchImpl::Invoke+0x9b (FPO: [Non
0533bb48 6f958168 10469fb8 0000003a 6f91eeec msxml6!DOMDocumentWrapper::Invoke+0x5d (FPO:
0533bba4 6f9580ee 10469fd4 6f9ee2a8 00000000 msxml6!_dispatchImpl::InvokeEx+0x10f (FPO: [
0533bbd4 65189ae3 10469fec 0000003a 00000002 msxml6!_dispatchEx<IXMLDOMNode,&LIBID_MSXML2
0533bc40 651925fc 0000003a 00000001 0533bcf8 jscript9!HostDispatch::CallInvokeEx+0x175 (F
0533bc64 65192598 0000003a 00000001 0533bcf8 jscript9!HostDispatch::InvokeMarshaled+0x6a
0533bd54 65192637 0000003a 00000000 10000002 jscript9!HostDispatch::InvokeByDispId+0x420
0533bd70 65110e57 0a268c00 10000002 0a2790a0 jscript9!DispMemberProxy::DefaultInvoke+0x23

0:007>
```

Ln 0, Col 0 Sys 0:<Local> Proc 000:6fc Thrd 007:58c ASM OVR CAPS NUM
7:56 PM
9/17/2017

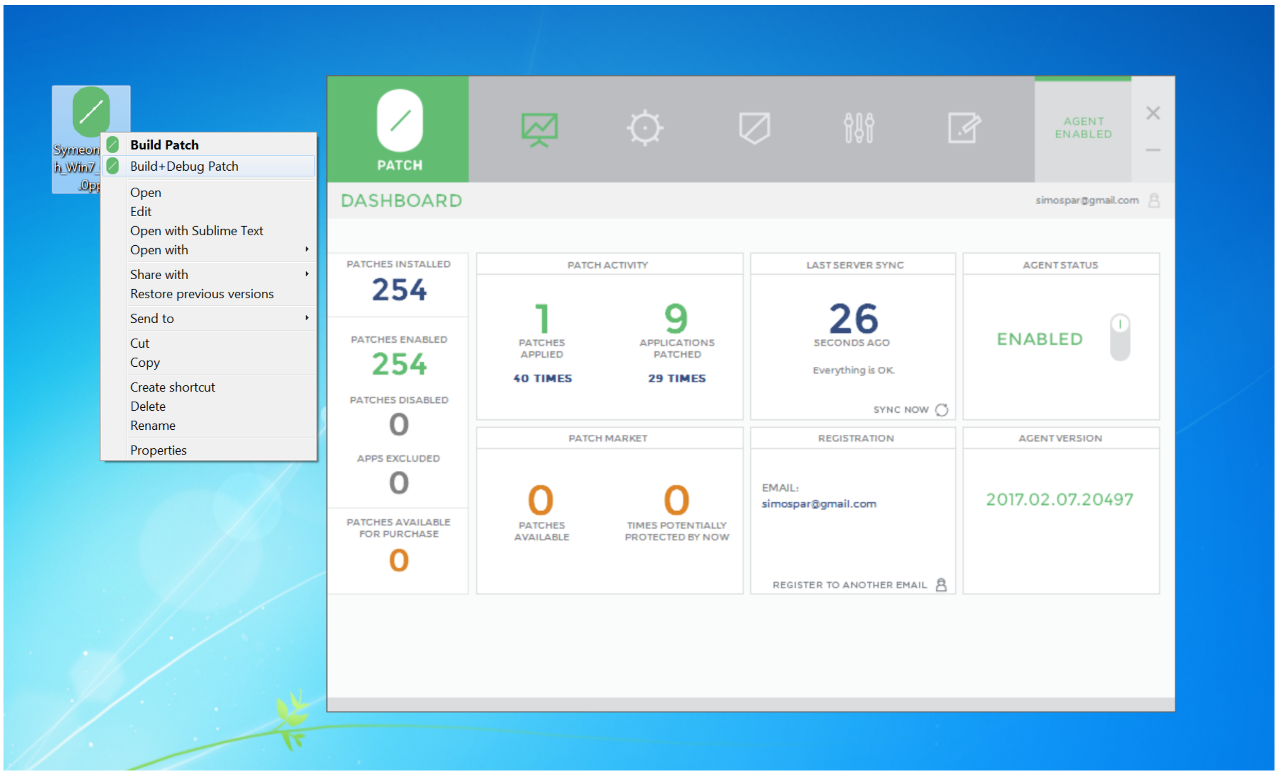
Bingo! As expected, at least with PageHeap enabled we are able to trigger exactly the same crash as with our harness. Be careful *not* to include that xml on Microsoft Outlook, because it will also crash it as well! Also, since it's on the library itself, had it been a more sexy crash would increase the attack surface!

Patching

After exchanging a few emails with Mitja, he kindly provided me the following patch which can be applied on a fully updated x64 system:

```
;target platform: Windows 7 x64
;
RUN_CMD C:\Users\symeon\Desktop\xmlvalidate_64bit\xmlvalidate.exe
C:\Users\symeon\Desktop\xmlvalidate_64bit\poc2.xml
MODULE_PATH "C:\Windows\System32\msxml6.dll"
PATCH_ID 200000
PATCH_FORMAT_VER 2
VULN_ID 9999999
PLATFORM win64


patchlet_start
 PATCHLET_ID 1
 PATCHLET_TYPE 2

 PATCHLET_OFFSET 0xD093D
 PIT msxml6.dll!0xD097D

 code_start

  test rbp, rbp ;is rbp (this) NULL?
  jnz continue
  jmp PIT_0xD097D
  continue:
 code_end
patchlet_end
```

Let's debug and test that patch, I've created an account and installed the 0patch agent for developers, and continued by right clicking on the above `.0pp` file:



DASHBOARD simospar@gmail.com

PATCHES INSTALLED 254
PATCHES ENABLED 254
PATCHES DISABLED 0
APPS EXCLUDED 0
PATCHES AVAILABLE FOR PURCHASE 0

PATCH ACTIVITY
1 PATCHES APPLIED 40 TIMES
9 APPLICATIONS PATCHED 29 TIMES

LAST SERVER SYNC
26 SECONDS AGO
Everything is OK.
SYNC NOW

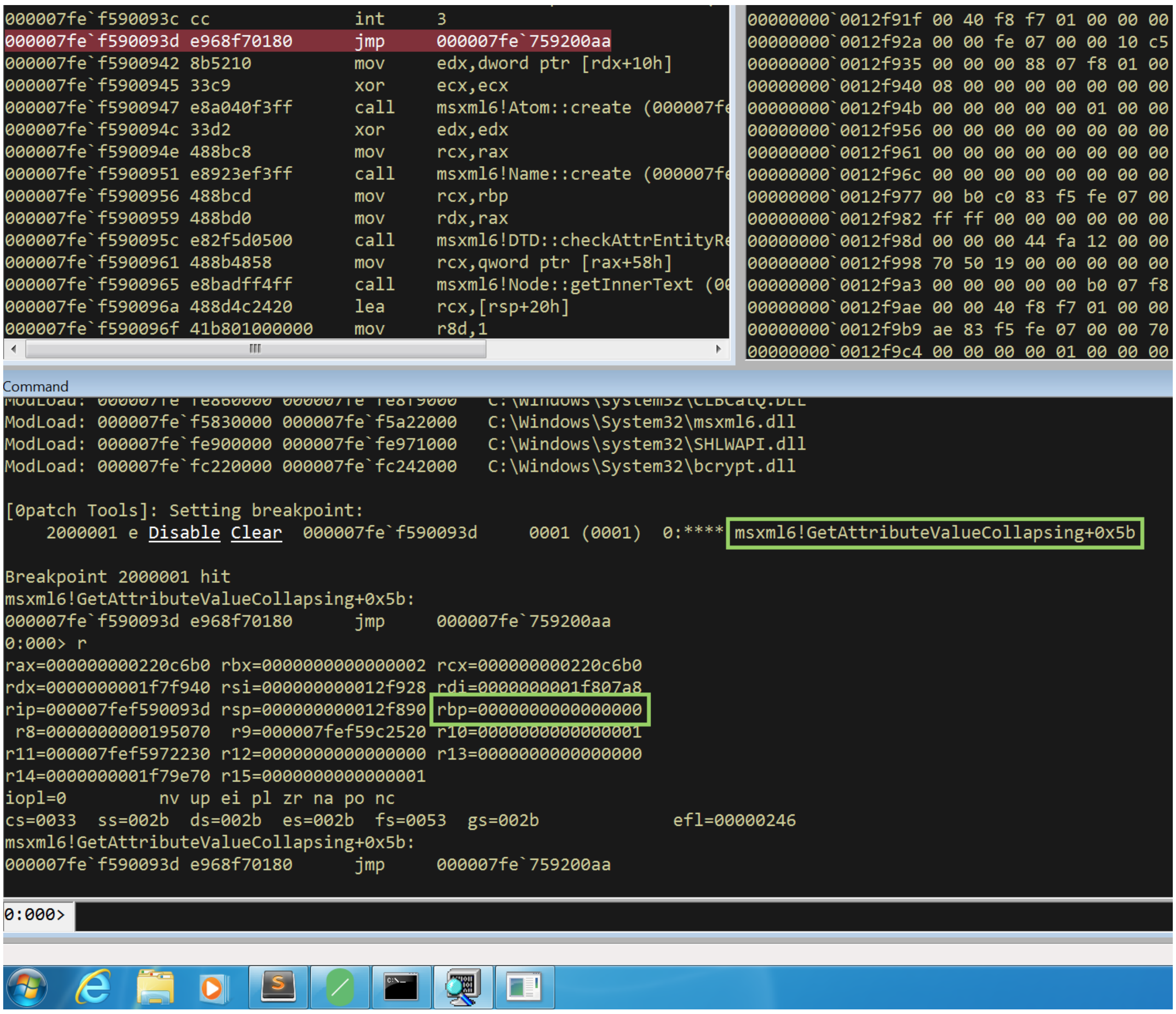
AGENT STATUS
ENABLED

PATCH MARKET
0 PATCHES AVAILABLE
0 TIMES POTENTIALLY PROTECTED BY NOW

REGISTRATION
EMAIL: simospar@gmail.com
REGISTER TO ANOTHER EMAIL

AGENT VERSION
2017.02.07.20497

Once I’ve executed my harness with the xml crasher, I immediately hit the breakpoint:



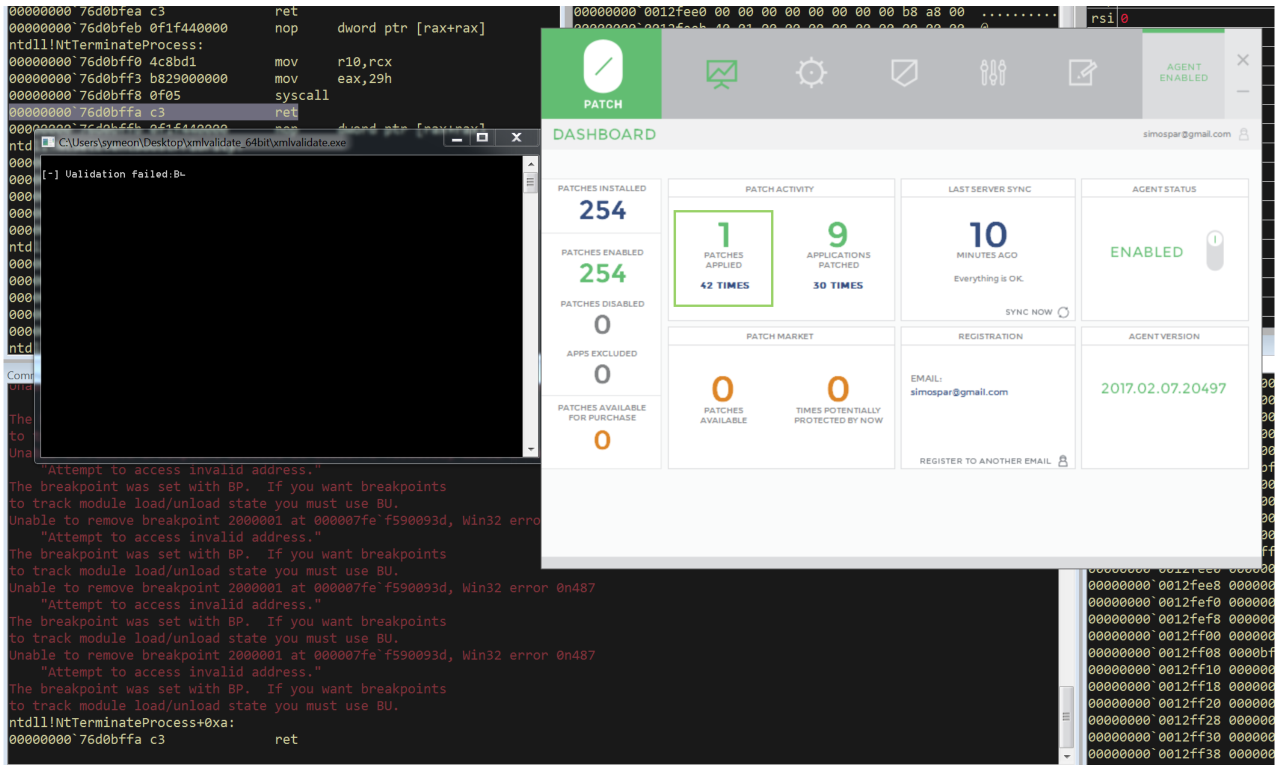
```
000007fe`f590093c cc              int     3
000007fe`f590093d e968f70180      jmp     000007fe`759200aa
000007fe`f5900942 8b5210          mov     edx,dword ptr [rdx+10h]
000007fe`f5900945 33c9            xor     ecx,ecx
000007fe`f5900947 e8a040f3ff      call    msxml6!Atom::create (000007fe
000007fe`f590094c 33d2            xor     edx,edx
000007fe`f590094e 488bc8          mov     rcx,rax
000007fe`f5900951 e8923ef3ff      call    msxml6!Name::create (000007fe
000007fe`f5900956 488bcd          mov     rcx,rbp
000007fe`f5900959 488bd0          mov     rdx,rax
000007fe`f590095c e82f5d0500      call    msxml6!DTD::checkAttrEntityRe
000007fe`f5900961 488b4858        mov     rcx,qword ptr [rax+58h]
000007fe`f5900965 e8badff4ff      call    msxml6!Node::getInnerText (00
000007fe`f590096a 488d4c2420      lea     rcx,[rsp+20h]
000007fe`f590096f 41b801000000    mov     r8d,1
```

```
00000000`0012f91f 00 40 f8 f7 01 00 00 00
00000000`0012f92a 00 00 fe 07 00 00 10 c5
00000000`0012f935 00 00 00 88 07 f8 01 00
00000000`0012f940 08 00 00 00 00 00 00 00
00000000`0012f94b 00 00 00 00 00 01 00 00
00000000`0012f956 00 00 00 00 00 00 00 00
00000000`0012f961 00 00 00 00 00 00 00 00
00000000`0012f96c 00 00 00 00 00 00 00 00
00000000`0012f977 00 b0 c0 83 f5 fe 07 00
00000000`0012f982 ff ff 00 00 00 00 00 00
00000000`0012f98d 00 00 00 44 fa 12 00 00
00000000`0012f998 70 50 19 00 00 00 00 00
00000000`0012f9a3 00 00 00 00 00 b0 07 f8
00000000`0012f9ae 00 00 40 f8 f7 01 00 00
00000000`0012f9b9 ae 83 f5 fe 07 00 00 70
00000000`0012f9c4 00 00 00 00 01 00 00 00
```

Command

```
ModLoad: 000007fe`fe860000 000007fe`fe879000   C:\Windows\system32\CLBCatQ.DLL
ModLoad: 000007fe`f5830000 000007fe`f5a22000   C:\Windows\System32\msxml6.dll
ModLoad: 000007fe`fe900000 000007fe`fe971000   C:\Windows\system32\SHLWAPI.dll
ModLoad: 000007fe`fc220000 000007fe`fc242000   C:\Windows\System32\bcrypt.dll

[0patch Tools]: Setting breakpoint:
    2000001 e Disable Clear  000007fe`f590093d     0001 (0001)  0:**** msxml6!GetAttributeValueCollapsing+0x5b

Breakpoint 2000001 hit
msxml6!GetAttributeValueCollapsing+0x5b:
000007fe`f590093d e968f70180      jmp     000007fe`759200aa
0:000> r
rax=000000000220c6b0 rbx=0000000000000002 rcx=000000000220c6b0
rdx=0000000001f7f940 rsi=000000000012f928 rdi=0000000001f807a8
rip=000007fef590093d rsp=000000000012f890 rbp=0000000000000000
 r8=0000000000195070  r9=000007fef59c2520 r10=0000000000000001
r11=000007fef5972230 r12=0000000000000000 r13=0000000000000000
r14=0000000001f79e70 r15=0000000000000001
iopl=0         nv up ei pl zr na po nc
cs=0033  ss=002b  ds=002b  es=002b  fs=0053  gs=002b             efl=00000246
msxml6!GetAttributeValueCollapsing+0x5b:
000007fe`f590093d e968f70180      jmp     000007fe`759200aa
0:000>
```

From the code above, indeed **rbp** is `null` which would lead to the null pointer dereference. Since we have deployed the 0patch agent though, in fact it’s going to jump to `msxml6.dll!0xD097D` and avoid the crash:



```
00000000`76d0bfea c3              ret
00000000`76d0bfeb 0f1f440000      nop     dword ptr [rax+rax]
ntdll!NtTerminateProcess:
00000000`76d0bff0 4c8bd1          mov     r10,rcx
00000000`76d0bff3 b829000000      mov     eax,29h
00000000`76d0bff8 0f05            syscall
00000000`76d0bffa c3              ret
```

C:\Users\symeon\Desktop\xmlvalidate_64bit\xmlvalidate.exe

```
[-] Validation failed:B
```

```
    "Attempt to access invalid address."
The breakpoint was set with BP.  If you want breakpoints
to track module load/unload state you must use BU.
Unable to remove breakpoint 2000001 at 000007fe`f590093d, Win32 erro
    "Attempt to access invalid address."
The breakpoint was set with BP.  If you want breakpoints
to track module load/unload state you must use BU.
Unable to remove breakpoint 2000001 at 000007fe`f590093d, Win32 error 0n487
    "Attempt to access invalid address."
The breakpoint was set with BP.  If you want breakpoints
to track module load/unload state you must use BU.
Unable to remove breakpoint 2000001 at 000007fe`f590093d, Win32 error 0n487
    "Attempt to access invalid address."
The breakpoint was set with BP.  If you want breakpoints
to track module load/unload state you must use BU.
ntdll!NtTerminateProcess+0xa:
00000000`76d0bffa c3              ret
```

```
00000000`0012fee0 00 00 00 00 00 00 00 00 b8 a8 00  ..........
```

```
rsi 0
```

```
00000000`0012fee8 000000
00000000`0012fef0 000000
00000000`0012fef8 000000
00000000`0012ff00 000000
00000000`0012ff08 0000bf
00000000`0012ff10 000000
00000000`0012ff18 000000
00000000`0012ff20 000000
00000000`0012ff28 000000
00000000`0012ff30 000000
00000000`0012ff38 000000
```

PATCHES INSTALLED	254
PATCHES ENABLED	254
PATCHES DISABLED	0
APPS EXCLUDED	0
PATCHES AVAILABLE FOR PURCHASE	0

PATCH ACTIVITY: 1 PATCHES APPLIED 42 TIMES; 9 APPLICATIONS PATCHED 30 TIMES

LAST SERVER SYNC: 10 MINUTES AGO. Everything is OK. SYNC NOW

AGENT STATUS: ENABLED

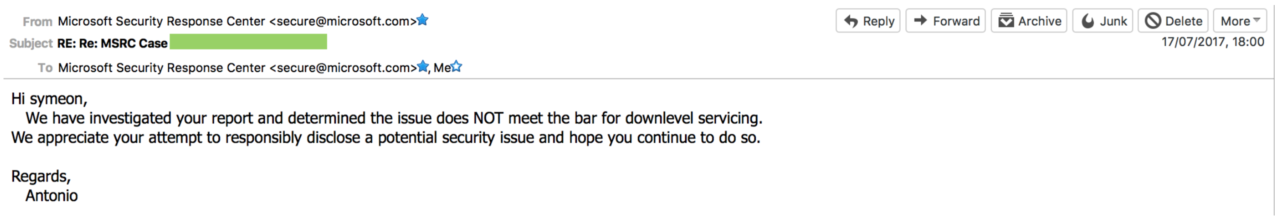
PATCH MARKET: 0 PATCHES AVAILABLE; 0 TIMES POTENTIALLY PROTECTED BY NOW

REGISTRATION: EMAIL: simospar@gmail.com. REGISTER TO ANOTHER EMAIL

AGENT VERSION: 2017.02.07.20497

Fantastic! My next step was to fire up winafl again with the patched version which unfortunately failed. Due to the nature of 0patch (function hooking?) it does not play nice with WinAFL and it crashes it.

Nevertheless, this is a sort of "DoS 0day" and as I mentioned earlier I reported it to Microsoft back in June 2017 and after twenty days I got the following email:



Reply
Forward
Archive
Junk
Delete
More

From Microsoft Security Response Center <secure@microsoft.com>
Subject **RE: Re: MSRC Case** [redacted]
To Microsoft Security Response Center <secure@microsoft.com>, Me
17/07/2017, 18:00

Hi symeon,
We have investigated your report and determined the issue does NOT meet the bar for downlevel servicing.
We appreciate your attempt to responsibly disclose a potential security issue and hope you continue to do so.

Regards,
Antonio

I totally agree with that decision, however I was mostly interested in patching the annoying bug so I can move on with my fuzzing :o)
After spending a few hours on the debugger, the only "controllable" user input would be the length of the encoding string:

```
eax=03052660 ebx=0012fc3c ecx=00000011 edx=00000020 esi=03054f24 edi=00000002
eip=6f80e616 esp=0012fbd4 ebp=0012fbe4 iopl=0         nv up ei pl zr na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000246
msxml6!Name::create+0xf:
6f80e616 e8e7e6f9ff      call    msxml6!Name::create (6f7acd02)
0:000> dds esp L3
0012fbd4  00000000
0012fbd8  03064ff8
0012fbdc  00000003

0:000> dc 03064ff8 L4
03064ff8  00610061 00000061 ???????? ????????  a.a.a...????????
```

The above unicode string is in fact our entity from the test case, where the number 3 is the length aparently (and the signature of the function: `Name *__stdcall Name::create(String *pS, const wchar_t *pch, int iLen, Atom *pAtomURN))`

Conclusion

As you can see, spending some time on Microsoft's APIs/documentation can be gold! Moreover, refactoring some basic functions and pinpointing the issues that affect the performance can also lead to massive improvements!

On that note I can't thank enough Ivan for porting the afl to Windows and creating this amazing project. Moreover thanks to Axel as well who's been actively contributing and adding amazing features.

Shouts to my colleague Javier (we all have one of those heap junkie friends, right?) for motivating me to write this blog, Richard who's been answering my silly questions and helping me all this time, Mitja from the 0patch team for building this patch and finally Patroklo for teaching me a few tricks about fuzzing a few years ago!

References

Evolutionary Kernel Fuzzing-BH2017-rjohnson-FINAL.pdf
Super Awesome Fuzzing, Part One