MAYHEM Sword Binary Introduction

Alex Eubanks <alex.eubanks@forallsecure.com>

August 1, 2018

Agenda

MAYHEM Sword Binary Introduction

This tutorial is designed to be the first in a series of tutorials on how to use MAYHEM to work with binary applications. In this tutorial, we will:

- Package a simple binary executable for analysis with MAYHEM.
- Upload the package to MAYHEM for analysis.

Requirements

- A working copy of MAYHEM Sword.
- MAYHEM client installed on the command line in a modern linux distribution, such as Ubuntu or Debian.

What can MAYHEM Analyze?

In order for our binary to be analyzed with MAYHEM, the following properties must hold:

- The binary application must accept input through a means supported by MAYHEM.
- The binary application must run on an architecture and operating system supported by MAYHEM.
- You are looking for errors in the binary you are testing. For example, if you ran the python interpreter over python scripts, MAYHEM will not find errors in the python scripts, MAYHEM will find errors in the python interpreter.

Packaging objdump

`objdump` is an application distributed with GNU binutils to dump and display information about programs stored in common binary executable file formats. We will package `objdump` for MAYHEM, and upload `objdump` to our instance of MAYHEM Sword.

To begin, we will install `objdump`. To do this on a debian-based distro (Debian, Ubuntu, etc.), we run:

```
apt-get update && \
apt-get upgrade -y && \
apt-get install -y binutils
```

Running objdump

We can now run `objdump`, and see what options are available.

```
root@d5ec5618a4ed:~# objdump /bin/ls
Usage: objdump <option(s)> <file(s)>
 Display information from object <file(s)>.
 At least one of the following switches must be given:
  -a, --archive-headers    Display archive header information
  -f, --file-headers       Display the contents of the overall file
       header
  -p, --private-headers    Display object format specific file header
       contents
  -P, --private=OPT,OPT... Display object format specific contents
  -h, --[section-]headers  Display the contents of the section headers
  -x, --all-headers        Display the contents of all headers
...
  -r, --reloc              Display the relocation entries in the file
  -R, --dynamic-reloc      Display the dynamic relocation entries in the
        file
  @<file>                  Read options from <file>
  -v, --version            Display this program's version number
  -i, --info               List object formats and architectures
       supported
  -H, --help               Display this information
root@d5ec5618a4ed:~#
```

objdump with flags

Running with no flags isn't very helpful. If we take a look at the help menu that's printed out, we see there's a variety of flags printed out. The flags `-x` and `-s` look useful. Let's try running again with those flags.

```
root@d5ec5618a4ed:~# objdump -xD /bin/ls | head

/bin/ls:     file format elf64-x86-64
/bin/ls
architecture: i386:x86-64, flags 0x00000150:
HAS_SYMS, DYNAMIC, D_PAGED
start address 0x0000000000005430

Program Header:
    PHDR off    0x0000000000000040 vaddr 0x0000000000000040 paddr 0
         x0000000000000040 align 2**3
         filesz 0x00000000000001f8 memsz 0x00000000000001f8 flags r-x

...

  1d:   35 62 64 31 30          xor    $0x30316462,%eax
  22:   65 63 63 64             movslq %gs:0x64(%rbx),%esp
  26:   2e 64 65 62             cs fs gs (bad) {%k6}
  2a:   75 67                   jne    93 <_init@@Base-0x3435>
  2c:   00 00                   add    %al,(%rax)
  2e:   00 00                   add    %al,(%rax)
  30:   c2 01 46                retq   $0x4601
  33:   f8                      clc
root@d5ec5618a4ed:~#
```

FOR ALL SECURE

mayhem package objdump

Terrific. Now we know exactly how we want to invoke `objdump`, with the `-xD` flags.

To package this binary, we run `mayhem package /usr/bin/objdump`. Let's try that out now.

```
root@d5ec5618a4ed:~# mayhem package /usr/bin/objdump
INFO:root:Packaging application: /usr/bin/objdump
INFO:root:Packaging dependency: /usr/bin/objdump -> /tmp/objdump-
    wng7lsdf/root/usr/bin/objdump
INFO:root:Packaging dependency: /lib/x86_64-linux-gnu/libz.so.1 -> /tmp/
    objdump-wng7lsdf/root/lib/x86_64-linux-gnu/libz.so.1
INFO:root:Packaging dependency: /usr/lib/x86_64-linux-gnu/libopcodes
    -2.28-system.so -> /tmp/objdump-wng7lsdf/root/usr/lib/x86_64-linux-
    gnu/libopcodes-2.28-system.so
INFO:root:Packaging dependency: /usr/lib/x86_64-linux-gnu/libbfd-2.28-
    system.so -> /tmp/objdump-wng7lsdf/root/usr/lib/x86_64-linux-gnu/
    libbfd-2.28-system.so
INFO:root:Generating default configuration under: /tmp/objdump-wng7lsdf/
    config.json
INFO:root:Packaged /usr/bin/objdump under: /tmp/objdump-wng7lsdf
root@d5ec5618a4ed:~#
```

Package directory contents

There are a few files and folders inside the package we have created at /tmp/objdump-wng7lsdf.

```
root@d5ec5618a4ed:~# ls -lh /tmp/objdump-wng7lsdf
total 12K
-rw-r--r-- 1 root root  277 Jul  9 20:45 config.json
drwxr-xr-x 2 root root 4.0K Jul  9 20:45 corpus
drwxr-xr-x 4 root root 4.0K Jul  9 20:45 root
```

The corpus directory

The `corpus` directory contains seed inputs. Earlier, when we talked about mutation-based input generation, these are the inputs that MAYHEM would mutate. Fortunate for us, objdump parses linux executables, and we have plenty of those right in our example VM. Let's copy a couple over to the corpus directory.

```
~# cp /bin/pwd /tmp/objdump-wng7lsdf/corpus/
~# cp /bin/true /tmp/objdump-wng7lsdf/corpus/
~# cp /bin/cat /tmp/objdump-wng7lsdf/corpus/
```

The root directory contains all of the files and dependencies required for MAYHEM to invoke and run the binary we wish to test. This directory has been auto-populated with the dynamically-linked runtime dependencies for us.

FOR ALL SECURE

config.json

The `config.json` file contains auto-populated configuration information for MAYHEM, informing MAYHEM how to invoke and run our binary under test.

We do need to modify the `config.json` file. Let's take a look at what's inside first.

```
root@d5ec5618a4ed:~# cat /tmp/objdump-wng7lsdf/config.json
{
    "fuzzers": [
        {
            "target_args": [
                "@@"
            ],
            "target_input": "@@",
            "library_path": "root/usr/lib/x86_64-linux-gnu:root/lib/
                x86_64-linux-gnu",
            "target": "root/usr/bin/objdump"
        }
    ]
}
```

Changing config.json

For the purposes of this tutorial, we are only concerned with the "target_args" field. This is a json array of command-line arguments to be passed to the target binary. "@@" is a special argument that tells MAYHEM, "Create an input filename for me, and place it here." We want to pass the command line argument -xD before @@, so we will change "target_args" to look like this:

```
"target_args": [
    "-xD",
    ""@@"
]
```

mayhem upload

We're now going to upload this package to MAYHEM. We need the URL for the running instance of MAYHEM we wish to upload this package to. For me, this URL is `http://192.168.99.101:32434`.

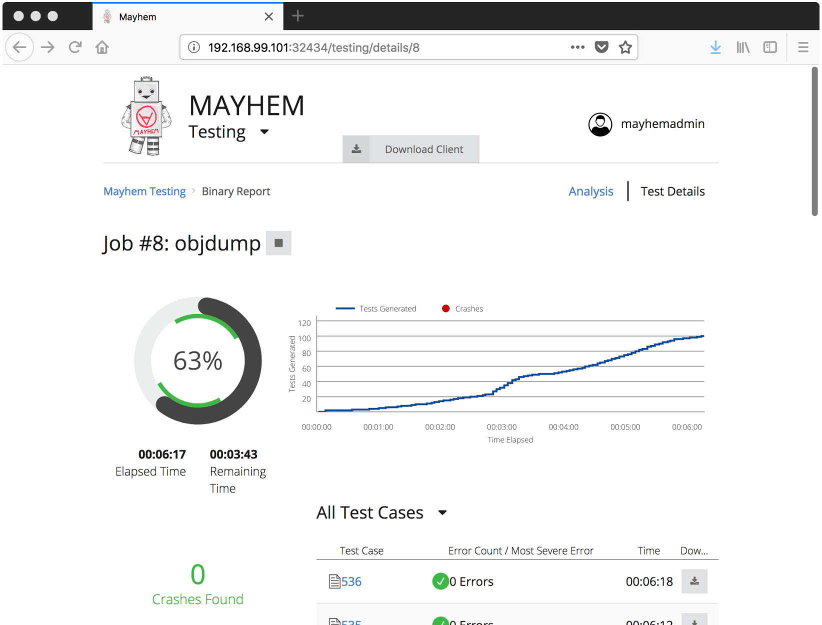
To upload, I will type:

```
mayhem upload --start-sword -u http://192.168.99.101:32434/ /tmp/
    objdump-wng7lsdf
```

`-start-sword` tells MAYHEM to start fuzzing as soon as the binary is uploaded, and `-u` allows us to specify a URL to our specific instance of MAYHEM.

FOR ALL SECURE

MAYHEM Sword Running



Mayhem

192.168.99.101:32434/testing/details/8

MAYHEM Testing

Download Client

mayhemadmin

Mayhem Testing › Binary Report

Analysis | Test Details

Job #8: objdump

63%

00:06:17 Elapsed Time

00:03:43 Remaining Time

0 Crashes Found

All Test Cases

Test Case	Error Count / Most Severe Error	Time	Dow...
536	✓ 0 Errors	00:06:18	
535	✓ 0 Errors	00:06:12	