Strengthening Capsicum Capabilities with Libpreopen

by

© *Stanley Uche Godfrey*

A thesis submitted to the

School of Graduate Studies

in partial fulfilment of the

requirements for the degree of

Master of *Science*

Department of *Scientific Computing*

Memorial University of Newfoundland

December 2017

St. John's Newfoundland

Abstract

"The aim of the project is to develop a library called libpreopen to make it possible for application authors to sandbox their applications with capsicum without rigorous modification to their application. At the moment, some applications that call lib c functions which capsicum friendly variant of these lib c functions are implemented in libpreopen wrapper functions can be successfully sandboxed with capsicum using libpreopen without any rigorous modification by the authors of these application."

— MUN School of Graduate Studies

Acknowledgements

Put your acknowledgements here...

"Intellectual and practical assistance, advice, encouragement and sources of monetary support should be acknowledged. It is appropriate to acknowledge the prior publication of any material included in the thesis either in this section or in the introductory chapter of the thesis."

— MUN School of Graduate Studies

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Gargets to be Secured

As more networks and gadgets are connected to the internet, the web becomes indispensable, it hosts productivity software suites for creating documents, spreadsheets and emails.Applications suites for scientific calculations, live television streaming and weather details hosted on the web. Some of the services web applications provide are online banking services, services for storing pictures and documents in the cloud, personal computers and mobile devices. Web applications also provide services that connect home devices such IP cameras to mobile phones for remote monitoring and e-commerce services. Sensitive data like passwords and credit card details are usually required to access these web services. These web services can have vulnerabilities which attackers can exploit, despite network defenses like firewall and intrusion prevention systems [2].

One such vulnerability in an application could be as a result of a Buffer Overflow [3]. A Buffer Overflow is a programming bug usually in a non-memory safe programs that results when a programmer fails to check if an input data is within the bounds of that input buffer. If the input data is more than the buffer can accommodate, the overflowing data will overwrite the contents of adjacent memory which may push instructions to be executed into the stack. An attacker who is able to add more data in a buffer than the buffer can accommodate could change execution path of applications intentionally and may acquire the root user right of the system which will allow the attacker to take total control of the system.

If a web application is vulnerable, attackers can use a technique known as Heap Spraying [1] to send malicious code to the heap memory of the web application in a computer. The Heap Spraying technique is used to duplicate the malicious code in different locations of the running application's heap memory to increase the chances of execution of the malicious code. Heap spraying is created with scripting languages like JavaScript. Different Malware exploited vulnerabilities found in internet explorer 6 and 7 around 2004 when the internet explorer web browser was believed to be the most popular web browser. Some of the vulnerabilities exploited in internet explorer include ANI(CVE2007-0038),VML(CVE-2006-4868) and the Operation Aurora exploit (CVE-2010-0248).

The first known buffer overflow exploitation that gained mainstream media attention was accomplished by Robert Morris, a graduate student of Cornell University. [3] Morris wrote an experimental program that duplicates itself in a computer and disseminates itself to other computers through a computer network. Morris was able to put this program on the internet which was fast replicating,infecting and refecting computer at a fast rate. Morris' program, known as a worm exploited a buffer overflow bug in the UNIX Sendmail program, a program which runs on a computer and waits for connections from other computers which it receives emails from. Morris' program also exploited a buffer overflow in UNIX finger. UNIX finger is a program that prints out the login and other details of a logged in user in a UNIX system.

These sorts of unauthorized access to computer resources by attackers are what capsicum mitigates, and libpreopen will make capsicum easier to use in limiting the damage intrusive malicious code from such cyber attack could cause.

1.2 Background

Capsicum is a system that boosts UNIX security with sandboxed capability mode and capabilities. Capability mode is the ability of capsicum to prohibit application fragments to interact with each other except in a regulated manner using capsicum capabilities rights. Fragments of applications in capability mode are totally isolated and these fragments are not allowed access to global namespaces. The reason for these restrictions is to contain vulnerabilities to a fragment and not allow the corruption to spread to other fragments of the application or the entire system.

Processes in total isolation cannot perform any task, this is where capsicum capabilities are required. capsicum capabilities are used to grant isolated processes in capsicum capability mode limited rights to perform specific actions in the capability token on a shared resource. For instance, a process may inherit file descriptors from a parent process or it may request access to a file from another process that has the right to send the file descriptor of the requested system file through IPC and before each of the processes enters capsicum capability mode. Regardless of how capability rights are acquired, processes in capability mode can only perform actions allowed in the capabilities granted on the file descriptors they acquire. A file descriptor acquired with capability right of cap_read cannot be used for fchmod(2) or have cap_write operation perform on it. [4]

For an application to be compartmentalized by capsicum, the application developer would make some modification to make the application conform to capsicum features. The modifications can be rigorous and time-consuming. These difficult application modifications are what libpreopen relieves application developers who want to make use of capsicum compartmentalization features of.

An example of an application developed without capsicum capability sandboxing in mind is tcpdump. tcpdump is a command line application for printing protocols and packets transmitted or received over a connected network and for printing the communication of another user or computer. In a network through which unencrypted traffic such as telnet or HTTP passes, tcpdump can be used to view login details, url and the content of visited websites by a superuser. Packet filter such BPF can be used to limit the number of packets captured by tcpdump.[4] For tcpdump to be sandboxed with capsicum and has its privileges reduced, it must be modified with the code in listing (1.1) and (1.2) and analysed with procstat tool to ensure that the capabilities exposed are the ones intended by the program author.

Listing 1.1: code to add capability mode to tcpdump

```
if (cap_enter() < 0)
 error("cap_enter: %s", pcap_strerror(errno));
status = pcap_loop(pd, cnt, callback, pcap_userdata);
```

Listing 1.2: code to narrow rights delegation in tcpdump

```
if (lc_limitfd(STDIN_FILENO, CAP_FSTAT) < 0)
 error("lc_limitfd: unable to limit STDIN_FILENO");
if (lc_limitfd(STDOUT_FILENO, CAP_FSTAT | CAP_SEEK | CAP_WRITE) < 0)
error("lc_limitfd: unable to limit STDOUT_FILENO");
 if (lc_limitfd(STDERR_FILENO, CAP_FSTAT | CAP_SEEK | CAP_WRITE) < 0)
 error("lc_limitfd: unable to limit STDERR_FILENO");
```

Chapter 2

Design and Implementation of Libpreopen

2.1 Design

The design of libpreopen was made to strengthen capsicum from these two viewpoints.

(1) libpreopen fortifies capsicum by making it possible for an application running in capsicum capability mode to run some commands that require System calls and access global namespaces without compromising the system security.

(2) libpreopen eradicates tedious application modifications, developers have to make in order to incorporate capsicum compartmentalization and sandboxing capabilities in their application.

2.2 Implementation

Libpreopen makes it possible for applications that requires global file namespace access to be sandboxed in capsicum without any modification by the author of these applications. Libpreopen is able to workaround the global file namespace access request of some applications at the moment being sandboxed with capsicum and have these applciations executed successfully in caspsicum capability mode. Listing (2.1) is the header file of libpreopen.

Listing 2.1: The header file of Libpreopen, libpreopen.h

```
struct po_map;

struct po_relpath {
int dirfd;
const char *relative_path;
};

struct po_map* po_map_create(int capacity);

void po_map_free(struct po_map *);


struct po_map* po_map_get(void);

void po_map_set(struct po_map*);
```

```

struct po_map* po_add(struct po_map *map,
 const char *path, int fd);

int po_preopen(struct po_map *, const char *path);

struct po_relpath po_find(struct po_map *map, const char *path,
cap_rights_t *rights);

const char* po_last_error(void);

int po_pack(struct po_map *map);

int po_map_length(struct po_map *map);

const char* po_map_name(struct po_map *map, int i);

int po_map_fd(struct po_map *map, int i);
```

Listing (2.1) is the header of file of libpreopen, listing the structures and functions which libpreopen was implemented with.

Libpreopen creates an extendable storage, pre-opens directories of file system that may be required by an untrusted application and stores the directory descriptors and the paths to the directories in the extendable storage.

Libpreopen has its implementation of lib c functions open(2) , access(2) and stat(2) at the moment. The functions are not allowed in capsicum capability mode because access to global file namespace is required. Therefore libpreopen implements these lib c functions using the fstatat(2) ,openat(2) and faccessat(2) variants which are capsicum capability friendly.When libpreopen is loaded with runtime linker ld_preload as environment variable, libpreopen is loaded before any other library and call to any of libc functions open(2) , access(2) and stat(2) made in the environment will execute libpreopen's version of the function because the search for the function will first be made in libpreopen.

2.3 Capsh

Capsh is a shell program for compartmentalization and running of untrusted application in capsicum capability sandbox. A program to be run in capsh is given as commands to capsh in one line. The first command is the application name and the others that follow are arguments needed to run the application. When a command is given to capsh to execute a program, capsh puts the name of the program to be executed and the program's arguments into a storage. capsh forks itself into a new process, enters capsicum sandbox capability mode and set file descriptors of shared libraries' directories as environment variables using ld_library_path_fds . Setting file descriptors of paths to library directories as enviroment variables conform to capsicum compartmentalization and sandboxing rule. After setting the environment variables, capsh starts the execution of the program with libc function fexecv.

Without libpreopen, capsh cannot do much exciting things. Only applications that do not require access to global file namespace can be executed on capsh for example, echo, a command that prints strings on the terminals of UNIX shell programs. If an attempt to execute a command that requires access to global file system namespaces is made the "action not permitted in capability mode" exception of capsicum is thrown and the program execution terminated.

Capsh with libpreopen at the moment can execute some UNIX commands that require access to global file namespace for execution. Figure 2.3.1 shows the how incorporation of libpreopen into capsh can make safe execution of an application that requires global file namespace in capsicum possible.

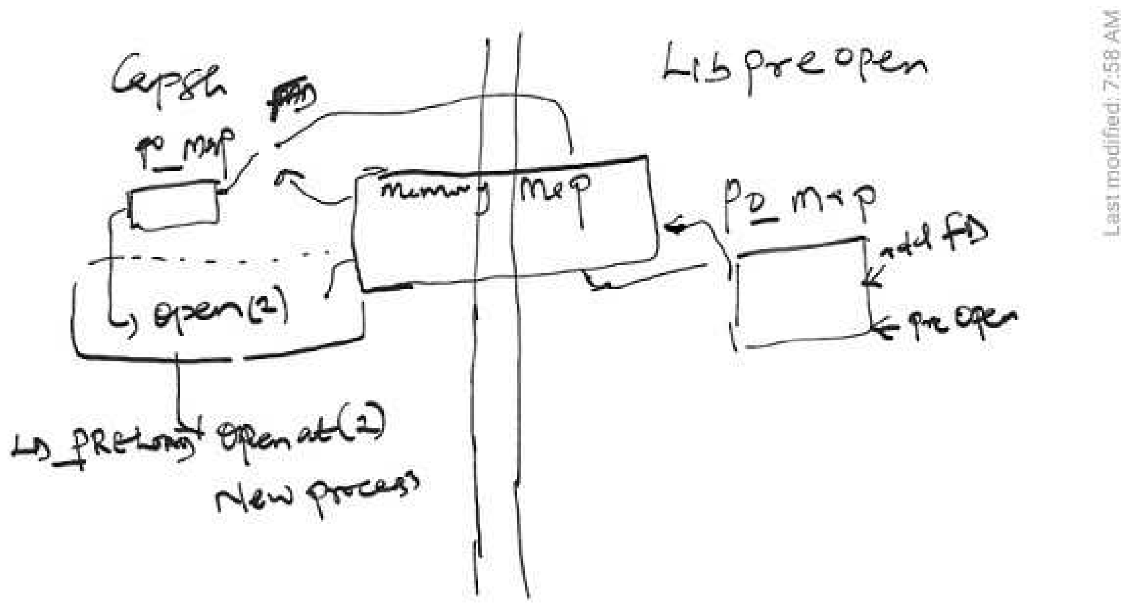

Figure 2.1: Shared memory mapping between libpreopen and a process started by capsh

From figure 2.1, it can be observed that when libpreopen is used with capsh, capsh delegates pre-opening of directories an application to be executed in capsh may need access to their contents. The file descriptors and paths to these directories are put in an extendable storage, mapped into shared memory segment and the file descriptor of the shared memory segment returned to capsh by libpreopen. Capsh sets the returned shared memory segment's file descriptor as environment variable, pre-load libpreopen with ld_preload in the environment for the application capsh is about to start executing and fexecves to start the execution of the new application. During the execution of the application, if call to any of lib c functions open(2) , access(2) and stat(2) is encountered, libpreopen's version of these functions which are capsicum sandbox capability friendly will be called instead.

Chapter 3

Evalution

3.1 Functional

Libpreopen was vetted by running some UNIX commands on FreeBSD shell, capsh without libpreopen and capsh with libpreopen. The UNIX shell commands used for vetting libpreopen are:

Echo displays are given string in the terminal window. A look into the source code of echo , shows that no access to global file namespace is required to execute the command.

The ls command lists all files matching the name provided if no name is provided, ls list all files and directories in a directory. The source code of ls between lines 263 and 266 shows that a system call that requires access to a global file namespace is called as shown in the listing 3.1, libpreopen is yet to implement a capsicum friendly variant of this system call.

Listing 3.1: The code snippet for UNIX command ls.c

```
if ((ftsp =fts_open(argv, options,
    f_nosort ? NULL : mastercmp)) == NULL)
        err(1, NULL);
```

The command Head accesses global file namespace during its execution with fopen function which has not been implemented in libpreopen's wrapper functions. Listing 3.2 is head.c line 67-79 code.

Listing 3.2: The code snippet for file of UNIX command head.c

```
for (first = 1; *argv; ++argv) {
    if ((fp = fopen(*argv, "r")) == NULL) {
err(0, "%s: %s", *argv, strerror(errno));
            continue;
    }
}
```

The current libpreopen wrapper functions are able to make wc execute on file in capsicum sandbox capability mode.

however command tail does not execute successfully because a function fopen and fstat will attemp access to global file namespace which is not allowed in capsicum capability mode as shown in list 3.3, a code snippet of tail.c} between lines 138 and 142. Implementations of safe variants of these functions are yet to made in libpreopen

Listing 3.3: The code snippet for file of UNIX command tail.c

```
if ((fp = fopen(fname, "r")) == NULL
    || fstat(fileno(fp), &sb)) {
            ierr();
            continue;
}
```

The command grep executes successfully in capsh which incorporates libpreopn while strings command fails. Listing 3.4, a code snippet of strings.c between lines 114 and 119 of shows that strings command makes a system call that accesses global file namespace. The capsicum safe variant of this system call is yet to be implemented in libpreopen

Listing 3.4: The code snippet for file of UNIX command strings.c

```
        if (!freopen(file, "r", stdin)) {
            (void)fprintf(stderr,"strings: %s: %s\n",
                file, strerror(errno));
             exitcode = 1;
             goto nextfile;
        }
```

3.2 Performance

capsh and libpreopen is expected to add to the performance cost of executing the application. Time is required for capsh to call libpreopen's functions and delegates pre-opening of resources needed by the application to libpreopen. Time is required for libpreopen to open and store resources to the shared memory map and time is required for capsh to set environment variables and fork itself into a new process. To compare the performance oflibpreopen, The UNIX commands wc, grep and cat were executed on text files of sizes between 10MB and 1000MB, in a machine with the following specification, Intel (R) Xeon(R) CPU E3-1240 v5 @ 3.50GHz, 32 GiB RAM and UFS filesystem on a 7200 RPM disk.

Figure 3.1, is the graph of the time of execution of UNIX command wc in FreeBSD shell against the size of data the command wc is called on. And the execution time of wc with libpreopen in capsh against the size of data wc is called on.

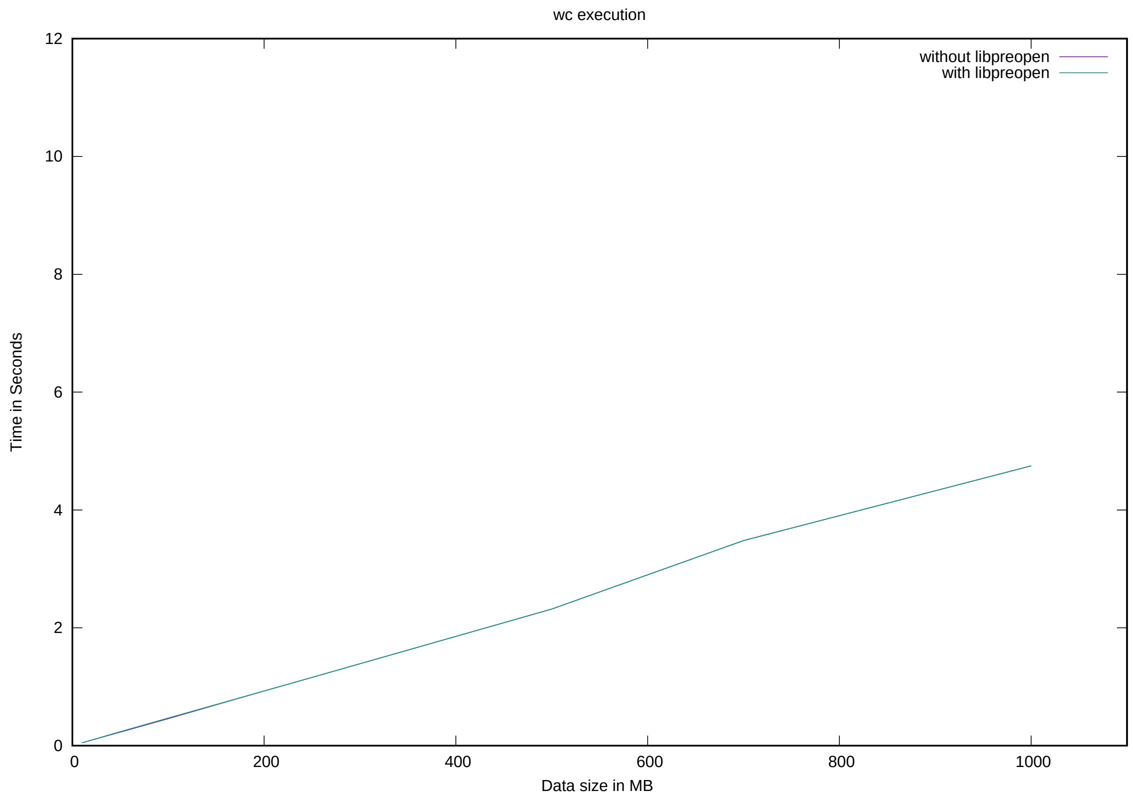

Figure 3.1: Data size vs time graph of wc command in FreeBSD shell and capsh

From figure 3.1, it can be seen that cost of running wc command in both FreeBSD shell, and capsh is the same on the same data size.

grep is a UNIX command that executes successfully in capsh with libpreopen. Figure 3.2 compares the cost of running grep command on text files of size between 10 MB and 1000 MB in FreeBSD shell and capsh.

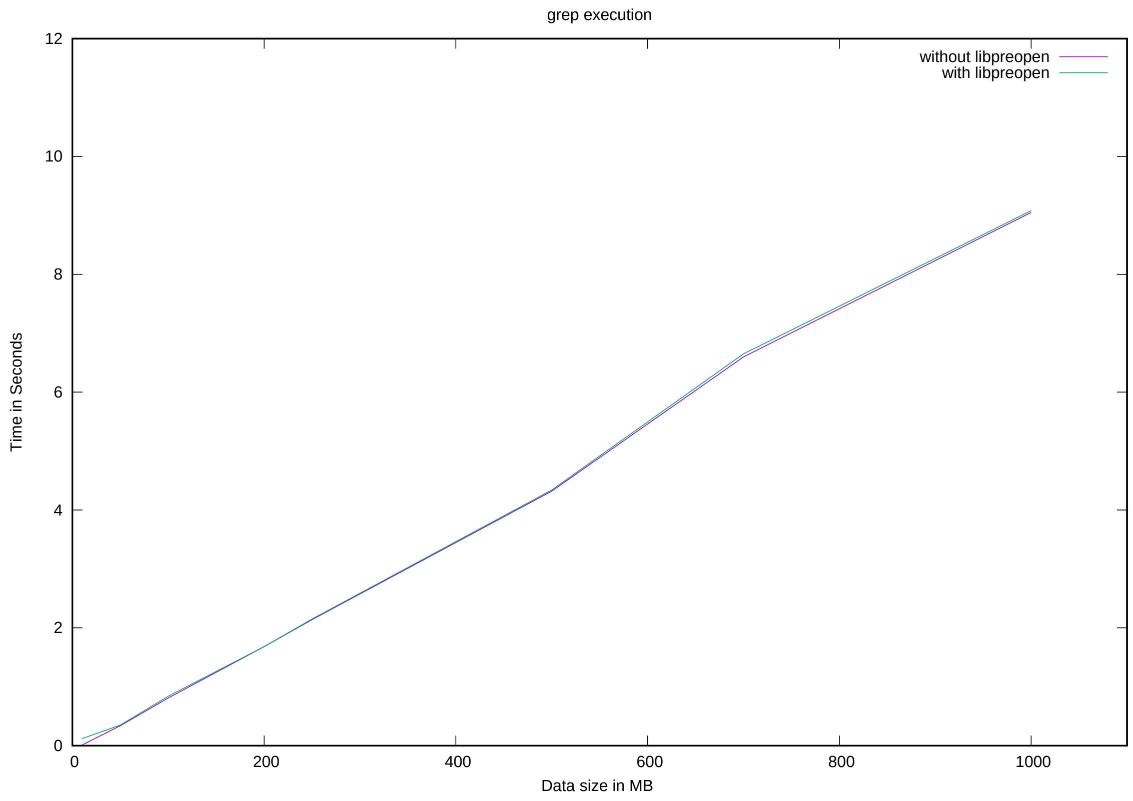

Figure 3.2: Data size vs time graph of grep command in FreeBSD shell and capsh

From Figure 3.2 it can be seen that the performance of grep command in FreeBSD shell, and capsh are almost the same.

The third FreeBSD command that execute successfully in capsh at the moment, is cat. cat displays the content of file and the larger the file the longer the execution duration. figure 3.3 shows the performance of cat in FreeBSD shell and casph.

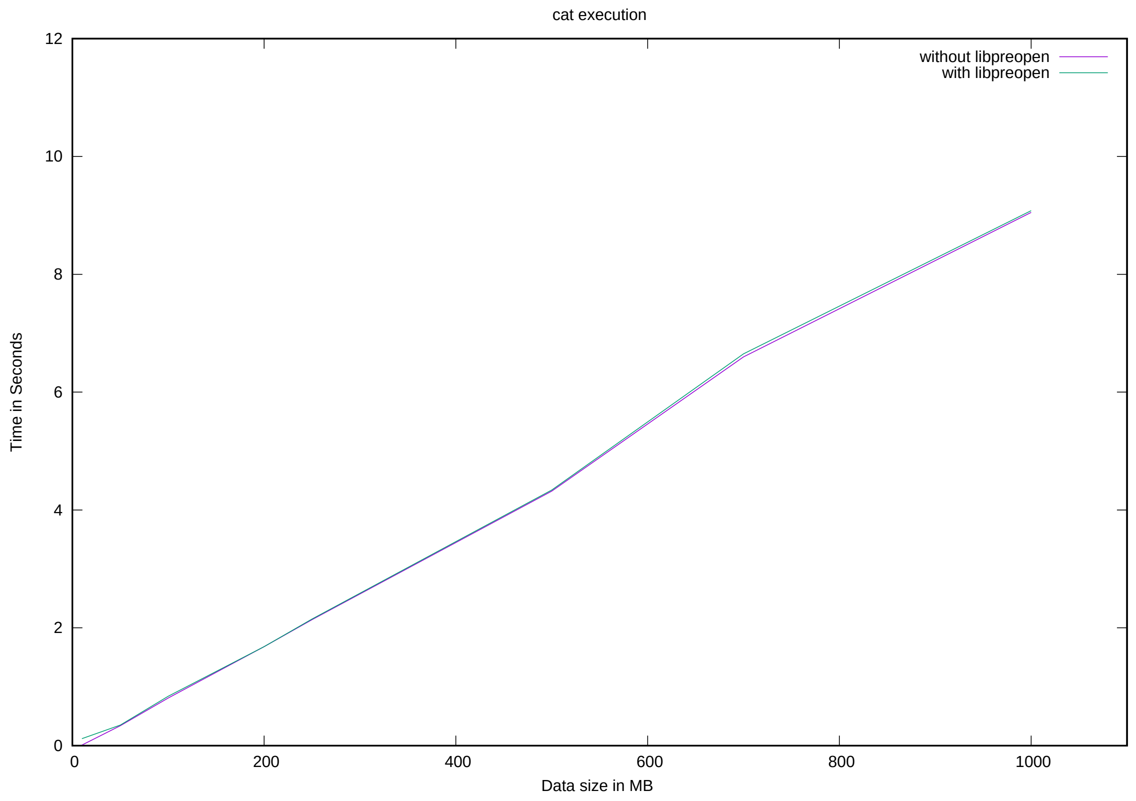

Figure 3.3: Data size vs time graph of cat command in FreeBSD shell and capsh

figure 3.3 shows that the cost of executing cat on a file in FreeBSD shell is almost the same as the cost executing cat on a file of same size in capsh with libpreopen

3.3 Future work

Not all UNIX commands are able to execute successfully in capsh with libpreopen. Such UNIX commands include head, tail, strings, and ls. These UNIX commands call variants of lib c functions which are not allowed in capsicum sandbox capability mode and which capsicum friendly variants are yet to be included in libpreopen wrapper functions. Such lib c functions include fopen, freopen , fstat and others yet to be identified. Including capsicum friendly variants of these lib c functions will make it possible for more applications to be sandboxed with capsicum using libpreopen without the authors of the applications making rigorous modifcations to make their applications conform to capsicum rules.

Chapter 4

Conclusion

If a vulnerability in an application is exploited and malicious data injected into the system, capsicum mitigates the spread of the malicious data by con

fining them to the affected process, since an application running in capsicum sandboxed mode is compartmentalized into processes and each process sandboxed.

However, an application running in capsicum sandboxed capability mode is forbidden to access global OS namespaces such as file system, process IDs, IPC namespaces. The application also has a restricted access to system calls while access to system calls that involves global namespace access is forbidden. For capsicum to contain the damage exploit of a vulnerability in an application can cause, the application has to give up its right to perform certain operations.

Applications running in capsicum capability mode can acquire capsicum capability rights, and libpreopen makes it possible for such applications to request system call operations which the applications have the capsicum capability rights for and have libpreopen performs these system call operations with libpreopen's version of lib c functions.

The cost of sandboxing an application with capsicum using libpreopen is insignificant compare to hours of rigorous application modification by applicaitons authors in order to make their application conform to capsicum rules. Application authors wishing to run unmodified binaries in a sandboxed capsicum capability mode should use libpreopen libary in order to avoid tedious and time consuming application modification.

Bibliography

[1] A. Ansari. Heap spraying. https://www.exploit-db.com/docs/31019.pdf. Accessed December 16, 2017.

[2] P. Lonescu. The 10 most common application attacks in action. https://securityintelligence.com/the-10-most-common-application-attacks-in-act Accessed December 15, 2017.

[3] Veracode. What is a buffer overflow learn about buffer overrun vulnerabilities exploits and attacks. http://https://www.veracode.com/security/buffer-overflow/. Accessed December 15, 2017.

[4] R. Watson, J. Anderson, B. Laurie, and K. Kennaway. Capsicum:practical capabilities for unix., 2011.

Appendix A

Appendix title

This is Appendix A.

You can have additional appendices too (*e.g.*, `apdxb.tex`, `apdxc.tex`, *etc.*). If you don't need any appendices, delete the appendix related lines from `thesis.tex` and the file names from `Makefile`.