Lab: Smashing the Stack For Fun and Learning

A lab on arbitrary code execution in a locked down system

Daniel Bosk

Department of Information and Communication Systems,
Mid Sweden University, SE-851 70 Sundsvall

1 Introduction

Buffer overruns of different kinds are common vulnerabilities in software. Examples vary from being able to break the DRM of a Nintendo Wii console [Wii] to reading the memory of a running OpenSSL implementation [Ope14].

How this really works is that the software mistakenly do not check the boundaries of buffers. This might be due to incorrect assumptions by developers, the system being comlpex, or simply human error.

1.1 Aim

The aims of this assignment is to look into buffer overrun vulnerabilities. The intended learning outcomes are as followin, after completion of this assignment you should be able to:

- Understand the consequences of vulnerabilities in software.
- Protect software from the easiest vulnerabilities.
- Evaluate strengths and weaknesses in software design.

The next section covers what you must read before you understand this assignment and how to do the work. Section 3 covers the work to be done, i.e. how you should learn this. Section 4 covers how it will be examined, i.e. how you show that you have fulfilled the intended learning outcomes given above.

2 Theory

To grasp this assignment you must first read Chap. 4, 8, 9, 18 in *Security Engineering* [And08] and then you must read Chap. 5–7 (and optionally 8), 10–12, 20, in *Computer Security* [Gol11].

After reading the material given above you need to know some assembly programming, specifically x86-64 assembler and some tools. For this you should read *x86-64 Machine-Level Programming* by Bryant [Bry05]. You also need to be acquainted with some tools, for that reason, study the manual pages for objdump(1), as(1), and gdb(1).

Finally, you should read the main paper of this assignment: the classic paper on stack smashing, the first paper on the matter to be precise, “Smashing the stack for fun and profit” [Ale96].

3 Assignment

This assignment will use the scenario of a buffer overrun bug found in Sun's Solaris 8 and 9 [McA05]. The scenario is a vulnerability in the passwd(1) program which allows for arbitrary code execution. We will use a much simplified version of the passwd(1) utility, the source code can be found in Sect. A in the appendix.

The first part of the lab concerns exploiting a buffer overrun vulnerability in this program to execute a new shell. This shell will, due to the nature of the passwd(1) utility, have root priviledges, since it can execute the setuid(2) system call to change the effective user ID to that of root.

This first part of the assignment will be solved together during a full-class hackathon in the computer lab. There will be a projector with the code for all to see, then we will rotate who will be by the keyboard writing what the rest of the class is saying. This way we will discuss together and write the code together, everyone will thus participate in the process.

The second part of the assignment is to dicsuss the consequences of this, among other things we will discuss the following questions:

- How can we possibly detect if this attack has occured somewhere?
- How can we protect ourselves from vulnerabilities such as these?

4 Examination

To pass this assignment you must first actively participate in the hackathon lab session. You must also actively contribute to the post-coding discussions.

If you cannot participate in the lab session you have to solve the lab yourself, then orally present your solution during one of the lab sessions after the course-end.

Acknowledgements This work is licensed under the Creative Commons Attribution-ShareAlike 3.0 Unported license. To view a copy of this license, visit `http://creativecommons.org/licenses/by-sa/3.0/`. Its original source code can be found in URL `https://github.com/OpenSecEd/stacksmashlab/`.

References

[Ale96] Aleph One. "Smashing the stack for fun and profit". In: *Phrack magazine* 7.49 (1996), pp. 14–16. URL: `http://www-inst.eecs.berkeley.edu/~cs161/fa08/papers/stack_smashing.pdf`.

[And08] Ross J. Anderson. *Security Engineering. A guide to building dependable distributed systems*. 2nd ed. Indianapolis, IN: Wiley, 2008. ISBN: 978-0-470-06852-6 (hbk.) URL: `http://www.cl.cam.ac.uk/~rja14/book.html`.

[Bry05] David R. Bryant Randal E. and O'Hallaron. *x86-64 Machine-Level Programming*. Sept. 2005. URL: https://www.cs.cmu.edu/~fp/courses/15213-s07/misc/asm64-handout.pdf.

[Gol11] Dieter Gollmann. *Computer Security*. 3rd ed. Chichester, West Sussex, U.K.: Wiley, 2011. ISBN: 9780470741153 (pbk.)

[McA05] Shaun McAdams. "Local Privilege Escalation in Solaris 8 and Solaris 9 via Buffer Overflow in passwd(1)". In: *SANS Institute Reading Room* (Feb. 2005). URL: https://www.sans.org/reading-room/whitepapers/solaris/local-privilege-escalation-solaris-8-solaris-9-buffer-overflow-passwd1-1600.

[Ope14] OpenSSL. *OpenSSL Security Advisory: TLS heartbeat read overrun (CVE-2014-0160)*. Apr. 7, 2014. URL: https://www.openssl.org/news/secadv_20140407.txt.

[Wii] Wiibrew. *Twilight Hack*. Accessed 9 April 2014. URL: http://www.wiibrew.org/wiki/Twilight_Hack.

A The simple passwd.c program

The following program will be used for experimenting with buffer overruns. The source code is available in a file with the source code of this document, that source also contains a Makefile to build it properly (with disabled stack protection etc.). See the repository on URL

https://github.com/OpenSecEd/stacksmashlab/.

```
#include <err.h>
#include <errno.h>
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#ifdef _GNU_SOURCE
#include <crypt.h>
#endif

int
test_passwd( const char *passwd )
{
        char pwd[256];
        size_t len = 0;
        FILE *file;

        /* use setuid(2) here to acquire priviledges */
        file = fopen( "master.passwd", "r" );

        if ( file == NULL )
```

```
                err( -1, "Could not check password" );

        len = fread( pwd, 1, 256, file );
        if ( len < 1 )
                return 0;
        pwd[len] = 0;

        if ( ! strcmp( pwd, crypt( passwd, "aa" ) ) )
                return 0;

        fclose( file );
        /* use setuid(2) to restore priviledges */

        return -1;
}

int
write_passwd( const char *passwd )
{
        char *pwdstr = crypt( passwd, "aa" );
        size_t len;
        FILE *file;

        /* use setuid(2) here to acquire priviledges */
        file = fopen( "master.passwd", "w" );

        if ( file == NULL )
                err( -1, "Could not write password" );

        len = fwrite( pwdstr, strlen( pwdstr ), 1, file );
        if ( len < 1 )
                err( -1, "Could not write password" );

        fclose( file );
        /* use setuid(2) to restore priviledges */

        return 0;
}

int
main( int argc, char **argv )
{
        char passwd[256];
        char newpass[256];
```

```
        printf( "Enter old password: " );
        scanf( "%s", passwd );

        if ( test_passwd( passwd ) < 0 ) {
                errx( -1, "Old password is wrong" );
        }

        printf( "Enter new password: " );
        scanf( "%s", passwd );

        printf( "Reenter new password: " );
        scanf( "%s", newpass );

        if ( strcmp( passwd, newpass ) ) {
                errx( -1, "Passwords do not match" );
        }
        else if ( write_passwd( passwd ) < 0 ) {
                err( -1, "Could not write password" );
        }

        return 0;
}
```