Smashing lab

This assignment investigates an old-fashioned way of breaking into systems executing x86 machine code, along with a couple of machine-level defenses against this attack. It's chosen not to give you a toolkit to break in to other sites – the method is well-known and ought to be commonly defended against nowadays – but instead, to give a general idea of how an attacker can break into a system by exploiting behavior that is undefined at the C level but defined at the machine level, and what we can do about it at the machine level.

Useful pointers

- Jake Edge, "Strong" stack protection for GCC, LWN.net (2014-02-05).
- Konstantin Serebryany, Derek Bruening, Alexander Potapenko, and Dmitry Vyukov, AddressSanitizer: a fast address sanity checker, *Proc USENIX ATC '12* (2012).
- Ben Lynn, 64-bit Linux Return-Oriented Programming (2012).
- Harold Rodriguez, 64-bit Linux stack smashing tutorial: Part 1 (2015-04-10) and Part 2 (2015-04-21). *Techorganic*.

Keep a log

Keep a log in the file `smashinglab.txt` of what you do in the lab so that you can reproduce the results later. This should not merely be a transcript of what you typed: it should be more like a true lab notebook, in which you briefly note down what you did and what happened. The log should help us verify that you've done the steps listed below, so that we can in principle reproduce them. For example, if one of the steps says "get a backtrace", make sure the backtrace goes into the log.

That's a nice little program you got there. Shame if anything were to happen to it

Consider the following patch to sthttpd. This patch applies to sthttpd 2.27.0, and deliberately introduces a bug into it. The idea is to suppose the the original programmer wrote this version by mistake.

```
--- sthttpd-2.27.0/src/thttpd.c 2014-10-02 15:02:36.000000000 -0700
+++ sthttpd-2.27.0-delta/src/thttpd.c   2015-04-30 19:15:24.820042000 -0700
@@ -999,7 +999,7 @@ static void
 read_config( char* filename )
     {
     FILE* fp;
-    char line[10000];
+    char line[100];
     char* cp;
     char* cp2;
     char* name;
@@ -1012,7 +1012,7 @@ read_config( char* filename )
        exit( 1 );
        }

-    while ( fgets( line, sizeof(line), fp ) != (char*) 0 )
+    while ( fgets( line, 1000, fp ) != (char*) 0 )
        {
        /* Trim comments. */
        if ( ( cp = strchr( line, '#' ) ) != (char*) 0 )
--- sthttpd-2.27.0/src/libhttpd.c       2014-10-03 11:43:00.000000000 -0700
+++ sthttpd-2.27.0-delta/src/libhttpd.c 2017-05-22 11:22:11.235627000 -0700
@@ -4078,7 +4078,7 @@ httpd_ntoa( httpd_sockaddr* saP )
        }
     else if ( IN6_IS_ADDR_V4MAPPED( &saP->sa_in6.sin6_addr ) && strncmp( str, "::ffff:", 7 ) == 0 )
        /* Elide IPv6ish prefix for IPv4 addresses. */
```

```
-        (void) strcpy( str, &str[7] );
+        (void) memmove( str, &str[7], strlen (&str[7]) + 1 );

     return str;
```

Steps to illustrate and defend against the bug

1. Make sure that `/usr/local/cs/bin` is at the start of your PATH; the command `which gcc` should output "`/usr/local/cs/bin/gcc`".
2. Build sthttpd with this patch applied. Assuming you are building with `/usr/local/cs/bin/gcc`, configure it with the shell command:

   ```
   ./configure \
      LDFLAGS="-Xlinker --rpath=/usr/local/cs/gcc-$(gcc -dumpversion)/lib"
   ```

   Compile it three times, with the following sets of compiler options:

   (SP) for strong stack protection:
   `-g3 -O2 -fno-inline -fstack-protector-strong`
   (AS) for address sanitization:
   `-g3 -O2 -fno-inline -fsanitize=address -static-libasan`
   (NO) for neither:
   `-g3 -O2 -fno-inline -fno-stack-protector -zexecstack`

   Call the resulting executables `src/thttpd-sp`, `src/thttpd-as`, and `src/thttpd-no`. You can do this with, for example, the command `make clean` followed by `make CFLAGS='-g3 -O2 -fno-inline -fstack-protector-strong'` and then by `mv src/thttpd src/thttpd-sp` and similarly for the other two variants.
3. Run each of the modified sthttpd daemons under GDB on port (12330 + 3 * (X % 293) + Y) on one of the SEASnet GNU/Linux servers, where X is your 9-digit student ID and Y is either 1, 2, or 3 depending on which variant of the daemon you're running (1=SP, 2=AS, 3=NO). For example, if your student ID is 123-456-789, (12330 + 3 * (123456789 % 293) + 1) equals 12532, so you can run the command `src/thttpd-sp -p 12532 -D`; the `-p` option specifies the port number and the `-D` is for debugging. You may find the thttpd man page useful.
4. Verify that your web servers work in the normal case. You can use the `curl` command to do this, e.g., the shell command `curl http://localhost:12532/foo.txt` where `foo.txt` is a text file in the working directory of your HTTPD server. The shell command `man curl` will give you more information about `curl`.
5. Make variant SP crash by invoking it in a suitable way. Run it under GDB, and get a backtrace immediately after the crash. Identify which machine instruction caused the crash, and why.
6. Make variant AS crash by invoking it in a similar way. Similarly, get a backtrace for it, and identify the machine instruction that crashed it and wy.
7. Likewise for variant NO.
8. Generate the assembly language code for `thttpd.c` three times, one for each variant, by using `gcc -S` rather than `gcc -c -g3` when compiling the file. (Use the same `-O` and `-f` flags as before.) Call the resulting files `src/thttpd-sp.s` and `src/thttpd-as.s` and `src/thttpd-no.s`. Compare the three assembly-language files' implementations of the `handle_read` function. Describe the techniques used by `-fstack-protector-strong` and `-fsanitize=address` to prevent buffer-overrun exploits in `handle_read`.
9. Build an exploit for the bug in variant NO that relies on the attacker tricking the victim into invoking `thttpd` with a particular value for the `-C` option. (Admittedly this is not much of an exploit, but we don't want you to have to put more easily exploitable HTTP servers on our network.) Your exploit should cause the victim web server that is invoked via GDB with `-C` to remove the file `victim.txt` in the working directory of the web server. Or, if such an exploit is impossible, explain why not, and investigate simple ways to alter the compiler flags (or, in the worst case, to insert plausible bugs into the source code) to make the exploit possible.

Submit

Submit the files `smashinglab.txt`, `thttpd-sp.s`, `thttpd-as.s`, and `thttpd-no.s` as described above, along with any other files needed to explain your exploit.

All text files should be ASCII files, with no carriage returns, and with no more than 200 columns per line. For example, the shell command

```
expand smashinglab.txt thttpd-sp.s thttpd-as.s thttpd-no.s |
  awk '/\r/ || 200 < length'
```

should output nothing.

$Id: smashinglab.html,v 1.11 2020/02/12 00:11:18 eggert Exp $