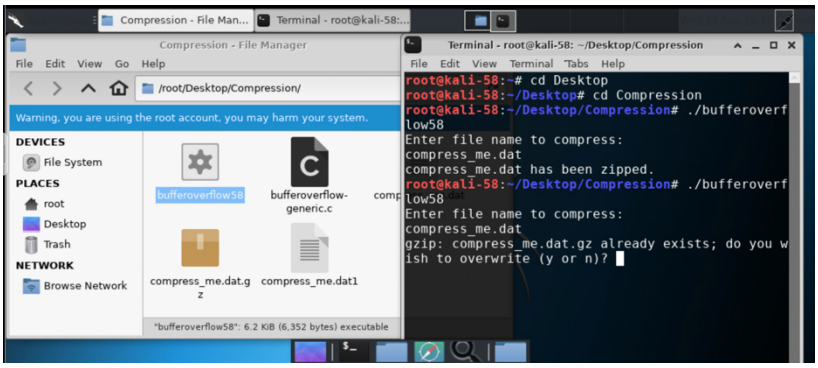
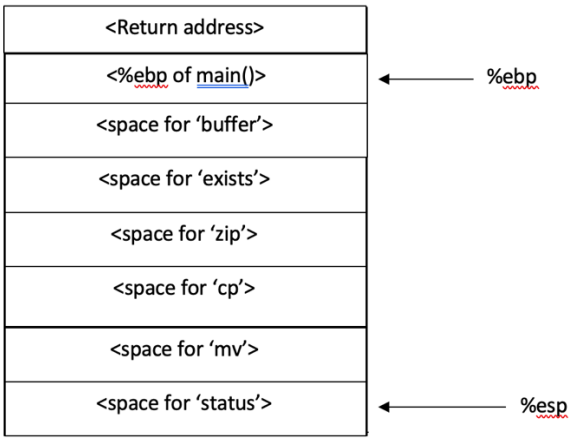
1. When you try to compress it again, it says it already exists and if you want to overwrite it





2.

3. The address for the secret function is 080484bb

4. The system runs on little Endian. I checked with

   lscpu | grep Endian

   which told me my byte order was little endian

5. *$ python -c 'print "a"*NN + "\xDD\xCC\xBB\xAA"' | ./bufferoverflowX*

   *print "a"*NN* takes the character a and prints it a certain number of times

+ \xDD\xCC\xBB\xAA takes the address and adds the 32 bits of char to it to overflow the buffer at that address

| ./bufferoverflowX pipes the payload into the program binary

For mine, NN is 36, DD is bb, CC is 84, BB is 04, and AA is 08

6. The secret message is “The password is violet58”

Step 9)

a) 10840 Buffer Overflow Attacks
    a. CVE-2020-5735 Cameras are vulnerable to a stack-based buffer overflow. An attacker can remotely take control of the device as well as execute arbitrary code.
    b. CVE-2020-3172: Due to a vulnerability in insufficiently validated packet headers, an attacker could cause a buffer overflow that could allow the attacker to execute arbitrary code as root or cause a DoS condition on the affected device

b) A Dynamic Buffer Overflow Detector by Olatunji Ruwase and Monica S. Lam

This article discusses a new buffer overflow detector called CRED (C Range Error Detector). It is different from other detectors because avoids this issues that other buffer detectors have like not being able to guard against all buffer overrun attacks, breaking existing code, or incurring too high an overhead. It doesn’t break code because it uses a novel solution and bound checks are

restricted to reduce overhead. It has been tested on more than 20 open source programs and more than a million lines of C code.

Buffer overflows have been around for decades. A buffer overflow is what allowed Robert T Morris to infect 10% of the internet with a worm in the 1980s(1). Decades later, we are still seeing buffer overflow issues like the OpenSSL's Heartbleed and the recent discovery of a bug in Apple's iOS mail app (2). However, prevention and detection are not always as simple as they seem.

One way to protect against buffer overflow attacks is to use a language that does not allow them like Java, Python, and .NET. The languages, do not "require special checks or changes to mitigate overflow vulnerabilities" (1). However, changing a language is almost never realistic so other options need to be considered. We can look at what type of functions we are using, like strcopy, strcat and strn-, which copy strings. However, strn- provides more protection than the others. Strcopy and strcat copy a string into a buffer and append it to another, without worrying about size limits. However, strn- will only write to the maximum target buffer size (1). However, this is not a flawless solution. It still requires a terminating character and without that character, you will have issues.

Another prevention would be to make sure the "data segment of the victims program's address space" is non-executable (3). This would make it so attackers could not inject code into the victim's program input buffers. However, as the author of this idea states, not all of a program's data segments can be non-executable without sacrificing program compatibility. However, program compatibility can mostly be preserved if only the stack segment is made non-executable.

There are more ways to help prevent buffer overflow. However, this paper only needs to be one page and I have finals next week, so these will be the only methods I'll be mentioning.

Works Cited

1. “How to Detect, Prevent, and Mitigate Buffer Overflow Attacks: Synopsys.” *Software Integrity Blog*, 28 Aug. 2019, www.synopsys.com/blogs/software-security/detect-prevent-and-mitigate-buffer-overflow-attacks/.

2. Tracy, Phillip. “Apple Confirms IOS Mail App Flaw in Millions of Devices - Promises Fix 'Soon'.” *LaptopMag*, Laptop Mag, 24 Apr. 2020, www.laptopmag.com/news/apple-confirms-ios-mail-app-flaw-in-millions-of-devices-promises-fix-soon.

3. Cowan, C., et al. “Buffer Overflows: Attacks and Defenses for the Vulnerability of the Decade.” *Proceedings DARPA Information Survivability Conference and Exposition. DISCEX00*, doi:10.1109/discex.2000.821514.

Step 10) I used -fno-stack-protector. When turned on, it allows more space to be allocated on the stack and checks whether the stack has been overwritten while in a function. You could also disassemble the the input file, convert it into an x86 file, add instructions/functions to make it invulnerable, and recompile using gcc. The pros of the first one is that it is quick and easy. However, it can cause linking problems when it compiles. The pro of the second one is that it permanently changes the code and makes it more secure. Its downside is that it takes much longer and requires converting back and forth.

Step 11) See attached file: bufferoverflow-generic.c

Step 13)

Question 6) We don't stop using the language because they are useful. They allow for memory modifications and are good for low level, high performance.

Question 7) One tool is FlawFinder, which analyzes the code and warns of any problems.

Question 8)

```
//reads in a line and stores it in the buffer
gets(buffer);
//joins buffer to the end of exists
strcat(exists, buffer);
//executes the system check for exists
status = system(exists);
//joins the buffer to the end of cp
```

```
strcat(cp, buffer);
//joins a space to the end of cp which will go in after buffer join
strcat(cp, " ");
//joins the buffer to the end of cp. Now its cp buffer " " buffer
strcat(cp, buffer);
//Again, it joins 1 to cp. Added to the end of what I wrote above
strcat(cp, "1");
//makes a system call
system(cp);
strcat(zip, buffer);
status = system(zip);
strcat(mv, buffer);
strcat(mv, "1");
strcat(mv, " ");
strcat(mv, buffer);
system(mv);
```

Step 14) get() strcpy() strcat().

Step 15)

Return-to-libc: Return address is changed. Gets around non-executable stack

Return oriented programming: advanced stack smashing that uses control flow to execute instructions in a specific way. Can fix with binary code randomization

Integer overflow: Integer arithmetic results in number too large to store. Use built in functions to check for overflow.

Format string vulnerability: Allows writing outside of the bounds of allocated memory.