Lab: Stack-Based Buffer Overflows

1. **Submit all the exploit code with comments for every line to demonstrate you understand how the exploits work.**
2. **Submit a diagram of the stack for `uppercase` which includes the stack frames for `main()` and `strcpy()` and all local variables for `main()`. Also, include the location of `main()`'s return address and how it gets overwritten by an overflown `buf`. Finally, include `argv[1]` in the diagram as well, relative to the stack frames. Exact offsets aren't that important here, but relative locations are.**

Stack Frames:

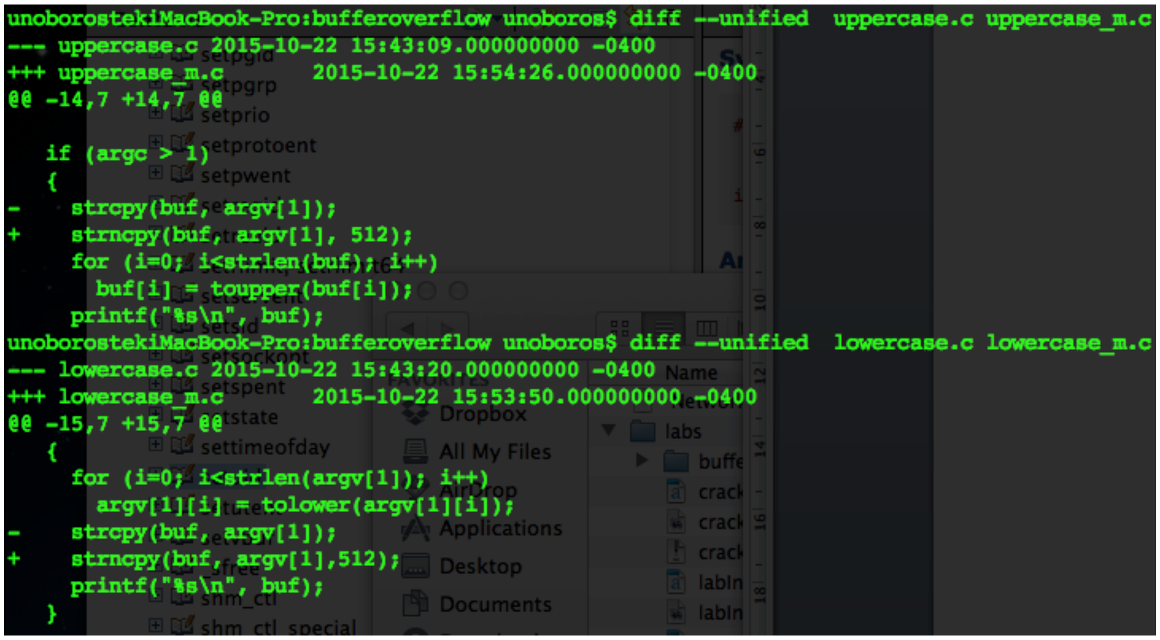
strcpy():

[ebp for strcpy]   [return address for strcpy] [pointer to buf]   [pointer to argv[1]]

main():

[buf]   [i]   [ebp for main]   [return address for main]   [argc]   [pointer to argv[0]]
[pointer to argv[1]]   [argv[0]]   [argv[1]]

3. **How would you modify `uppercase.c` and `lowercase.c` to make them safe from buffer overflow attacks? Submit an answer to this question in the form of a code diff of each program in the *unified* format. See the man page for `diff(1)`. Verify that your changes fixed the problem by running your exploits on the patched programs.**



```
unoborostekiMacBook-Pro:bufferoverflow unoboros$ diff --unified  uppercase.c uppercase_m.c
--- uppercase.c 2015-10-22 15:43:09.000000000 -0400
+++ uppercase_m.c     2015-10-22 15:54:26.000000000 -0400
@@ -14,7 +14,7 @@

   if (argc > 1)
   {
-    strcpy(buf, argv[1]);
+    strncpy(buf, argv[1], 512);
     for (i=0; i<strlen(buf); i++)
       buf[i] = toupper(buf[i]);
     printf("%s\n", buf);
unoborostekiMacBook-Pro:bufferoverflow unoboros$ diff --unified  lowercase.c lowercase_m.c
--- lowercase.c 2015-10-22 15:43:20.000000000 -0400
+++ lowercase_m.c     2015-10-22 15:53:50.000000000 -0400
@@ -15,7 +15,7 @@
   {
     for (i=0; i<strlen(argv[1]); i++)
       argv[1][i] = tolower(argv[1][i]);
-    strcpy(buf, argv[1]);
+    strncpy(buf, argv[1],512);
     printf("%s\n", buf);
   }
```

4. **If `buf` were instead allocated via `malloc(3)` in these vulnerable programs, would the same techniques for exploitation work? Why or why not?**

No it will not work anymore. Because our buffer overflow exploitation is targeting on the stack, more specifically, the return address on the stack. However, malloc() allocate memory on the heap. Although there is also a vulnerability for the heap overflow, it works totally differently.

5. **What is different about executing a setuid program versus a non-setuid program? What are the dangers of setuid programs? Why are setuid programs sometimes necessary?**

setuid() gives the program a privilege escalation during the runtime so that it can perform special tasks. Say, setuid(0) gives the program root privilege and makes it basically can do anything. The dangerous of this kind of program is that once it's been compromised (say, a buffer overflow attacking), the attacker will abuse it to get a privilege escalation and hence tack over the whole system! setuid() is necessary because sometimes users do need to perform certain system level operations during the program running, it's reasonable to elevate the program's privilege temporarily.

6. **Find an advisory for a remote overflow vulnerability published within the last year on the web which has an associated, published exploit. The exploit must be in the form of source code, and should be designed to execute code on the remote system (you do not need to test this). In what scenarios can the vulnerability you found be exploited? Include links and references to an advisory on the vulnerability, the exploit itself, and any related information you find.**

Vulnerability: the GHOST vulnerability is a serious weakness in the Linux glibc library. (https://community.qualys.com/blogs/laws-of-vulnerabilities/2015/01/27/the-ghost-vulnerability)

Advisory: http://tools.cisco.com/security/center/content/CiscoSecurityAdvisory/cisco-sa-20150128-ghost

Source: https://community.qualys.com/blogs/laws-of-vulnerabilities/2015/03/17/ghost-remote-code-execution-exploit

Scenario: there is a buffer overflow position in the __nss_hostname_digits_dots() function of glibc. And this buffer can be overflowed via a remote call of function gethostbyname().*