Ex070

Joshua Main-Smith

2020-10-14

Contents

Technical Report

Risk Assessment

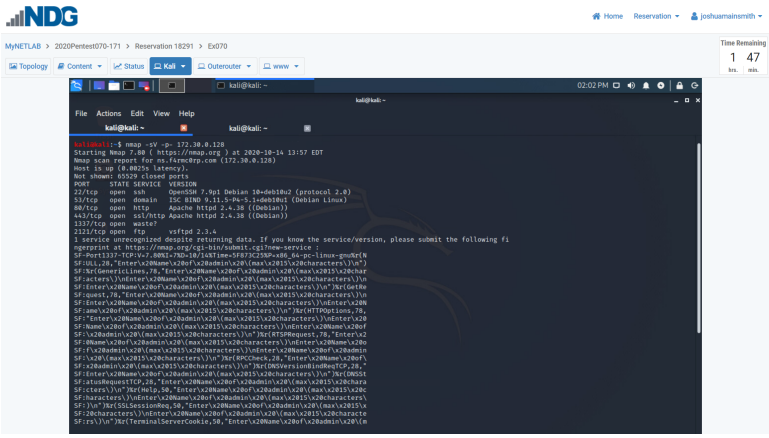
Vulnerability Description

The new service that Brian Oppenheimer has deployed is vulnerable to a buffer overflow attack. Entering 32 bytes of characters into the username field followed by arbitrary shell commands overwrites the populated commands buffer implemented by Oppenheimer's service. This can be mitigated by adding a field of authentication, such as a password. Additionally, this could further be mitigated by populating the commands after the user has signed in and to sanitize any input that a potential attacker may employ.

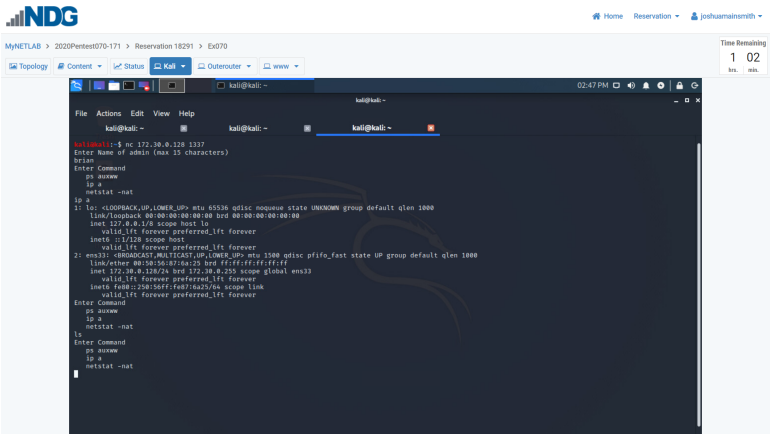
Attack Narrative

Gaining Access

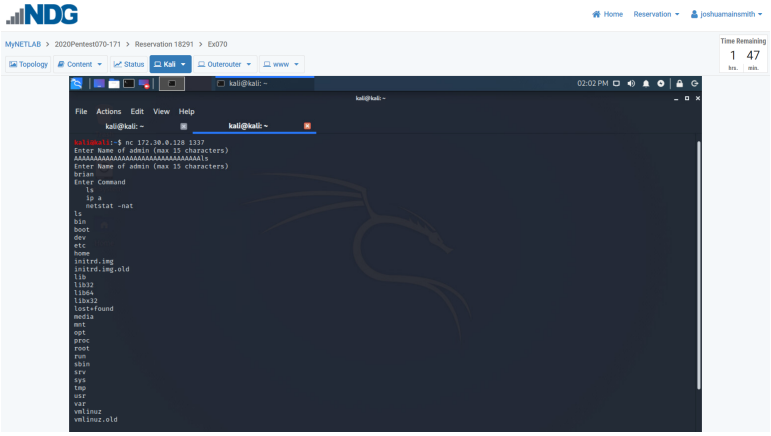
Given the post by Brian Oppenheimer on 42chan, we scanned 172.30.0.128 for any new services that may be available. After executing the command **nmap -sV -p- 172.30.0.128** (to scan all ports) we found a new service running on port **1337**.



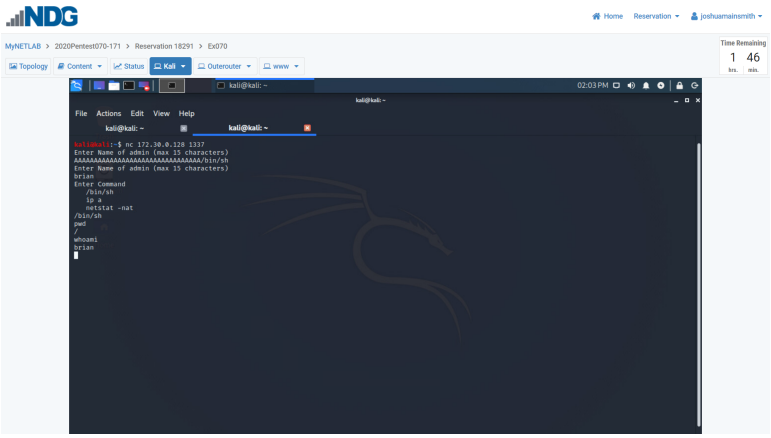
According to Oppenheimer's post, one can sign in simply with a username. So, we attempted to connect to the port with Netcat with **nc 172.30.0.128 1337**, which let us connect successfully. Upon connecting, we were asked to input a username (with a max character count of 15). We found that brian is a valid username, probably for Brian Oppenheimer. Upon entering, we were greeted with three commands we could execute, which are **ps auxww, ip a,** and **netstat -nat**. Further, we were unable to execute other shell commands outside of these. This was probably the extra security features that Brian had alluded to in his post.



An interesting feature that we had noticed is the indication that the admin must be 15 characters or less. Attempting input of various sizes then signing in as Brian again, we have found that we are able to overwrite the supplied commands allowed by the service in exchange for our own input. An example of this is given below.



One command we can issue that will give us shell access on the service is **/bin/sh**. This gave us full access to the shell commands available on the service.



As can be seen in the image above, we have full access to the service and are registered as Brian, which gave us permission to access his folder.

Source Code

We were able to access the source code Brian had used for his new service under **/home/brian/tool.c**. We can view the source code with **cat tool.c**. Our team determined that we were able to gain access because the the command list is populated before input of the admin is processed. We were effectively able to overwrite the buffer into the command buffer with shell code of our choosing.

```
28 #define BUFLEN 1024
29 #define NAMELEN 16
30 #define CMDLEN 12
31
32 char *buffer[BUFLEN] = {0};
33 char *enter_name = "Enter Name of admin (max 15 characters)";
34 char *enter_command = "Enter Command";
35 char admin[NAMELEN];
36 char next_command[CMDLEN+1];
37 char commands[37];
38
39 int  main(int argc, char const **argv, char const **envp)  {
40
41   pid_t child_pid;
42   int server_fd, new_socket, valread;
43   struct sockaddr_in sock_address;
44   int opt = 1;
45   int addrlen = sizeof(sock_address);
46
47   // Populate commandlist
48   strcpy(commands, "ps auxww");
49   strcpy(commands+CMDLEN, "ip a");
50   strcpy(commands+CMDLEN*2, "netstat -nat");
```

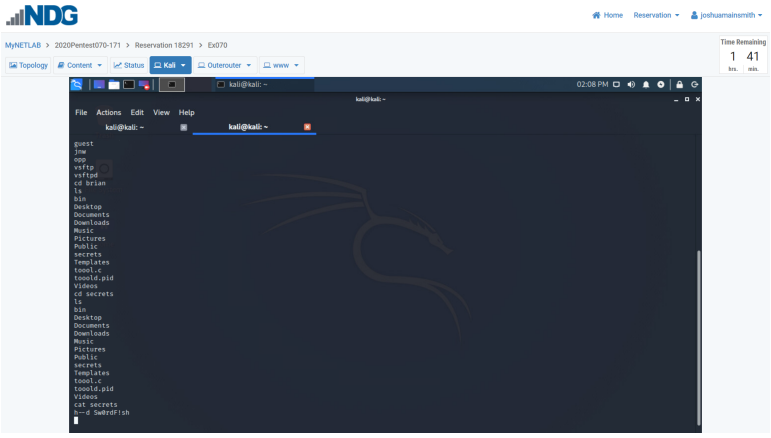
To overwrite successfully, we need to input at least 32 bytes of characters before using our own shell commands (hence the 32 A's before our shell command above). This is because BUFLEN on line 28 is defined as 1024 bits (32 bytes). The command list is populated on starting on line 48, then the input for admin

is processed on line 100. Following, the commands are iteratively displayed starting on line 108 (which have been overwritten).

```
96      // get admin user credential
97      while (strcmp(admin, MY_NAME) != 0) {
98        printf("%s\n", enter_name);
99        fflush(stdout); // Required for user interaction
100       fgets(admin, BUFLEN, stdin);
101       admin[strlen(admin)-1] = '\0';
102     }
103
104     // Process commands
105     while(1) {
106
107       // list available commands
108       printf("%s\n", enter_command);
109       for (int i=0; i < 3; i++) {
110         printf("   %s\n", commands + CMDLEN*i);
111       }
112       fflush(stdout);
113
114       // read user command, terminate on EOF
115       if (fgets(next_command, BUFLEN, stdin) == NULL) {
116         exit(EXIT_SUCCESS);
117       }
118       next_command[strlen(next_command)-1] = '\0';
```

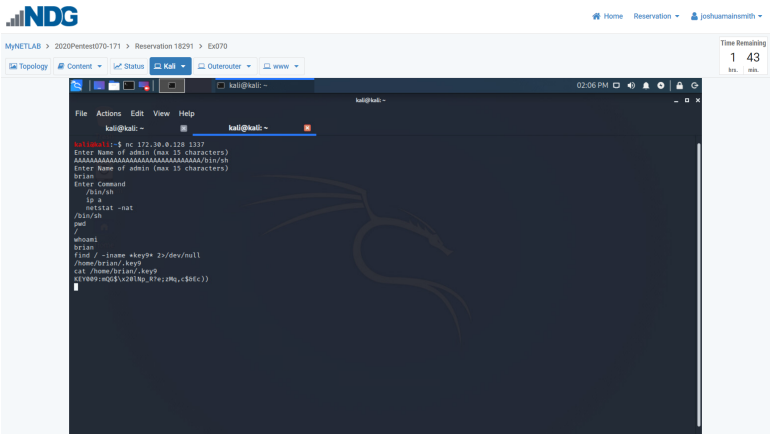
Other Findings

We were further able to find what appears to be a password under **/home/brian/secrets**.



We attempted to sign through the SSH and FTP ports with Brian's username and the password found without any luck.

We were also able to find a key as a hidden file under **/home/brian/.key9**. To find this in the main directory, we issued the command **find / -iname *key9* 2>/dev/null**.



The value of this key is **KEY009:mQG$\x20lNp_R?e;zMq,c$&Ec))**

Zoom Links

October 10, 2020 October 12, 2020