Homework 07

Md Moniruzzaman Monir, # 50291708

December 04, 2018

Problem 02

Solution:

My Reaction and thought :

I was really surprised after reading the paper "Reflections on trusting trust". I never thought of such self-reproducing program, and at first I didn't believe the idea of writing a self-reproducing program. But later when I saw the code in the paper and also run a simple python code which reproduced itself then I realize this is actually possible. After some exploration in the internet I found that this kind of program is known as "Quines". I read some blogs and articles about the concept of this type of program and discovered that this is actually happening all the time in human body. Similar concept is applicable when cell division occurs. A cell is actually reproducing itself, and the procedure is encoded in DNA. This is something which actually blew my head. I was also surprised how a buggy compiler can be compiled and the binary version can be used for compilation. Then if someone replace the bug from the source code then there is no way to detect the bug in binary version. So the moral is I can't cant trust code that I did not totally created by myself. No amount of source-level verification or scrutiny will protect me from using untrusted code. As the level of program gets lower and lower, these bugs will be harder and harder to detect. A well-installed microcode bug will be almost impossible to detect.

a) The self-reproducing programs are called "Quines". Writing a quine is not a hack that only works because the programming language has certain nice properties it is a consequence of the general so-called fixed-point theorem. The key to the program is :
It is impossible (in most programming languages) for a program to manipulate itself (i.e. its textual representation or a representation from which its textual representation can be easily derived) directly. So to make this possible anyway, we can write the program based on two parts, one which call the **code** and one which we call the **data**. The data represents (the textual form of) the code, and it is derived in an algorithmic way from it (mostly, by putting quotation marks around it, but sometimes in a slightly more complicated way). The code uses the data to print the code (which is easy because the data represents the code); then it uses the data to print the data (which is possible because the data is obtained by an algorithmic transformation from the code).

b) The article demonstrate a bug that was planted by the writer. The actual bug the writer planted in the compiler would match code in the UNIX **login** command. The replacement code would miscompile the login command so that it would accept either the intended encrypted password or a particular known password. Thus if this code were installed in a binary ,and the binary were used to compile the login command, the writer could log into that system as any user. So, the procedure is : First we compile the modified source with the normal C compiler to produce a bugged binary. We install this binary as the official C. We can now remove the bugs from the source of the compiler and the new binary will reinsert the bugs whenever it is compiled. Of course, the login command will remain bugged with no trace in source anywhere. This idea is related to **rootkits** in operating system security. A rootkit is a collection of computer software, typically malicious, designed to enable access to a computer or areas of its software that is not otherwise allowed (for example, to an unauthorized user) and often masks its existence or the existence of other software. So the idea of the bug implementation in C compiler for miscompiling the login command is similar to the strategy of rootkits.

Problem 03

Problem Statement :

Consider a distributed variant of the attack we explore in Problem 7.1. Assume the attacker has compromised a number of broadband-connected residential PCs to use as zombie systems. Also assume each such system has an average uplink capacity of **128 Kbps**.

a) What is the maximum number of **500-byte** ICMP echo request (ping) packets a single zombie PC can send per second?

b) How many such zombie systems would the attacker need to flood a target organization using a **0.5-Mbps** link? A **2-Mbps link**? Or a **10-Mbps** link?

c) Given reports of botnets composed of many thousands of zombie systems, what can you conclude about their controllers ability to launch DDoS attacks on multiple such organizations simultaneously?
Or on a major organization with multiple, much larger network links than we have considered in these problems?

Solution:

a) Assume that a single zombie PC can send maximum **x** packets of **500-byte**. The average uplink capacity of a single zombie PC is **128 Kbps**. The formula for finding the value of X is :

$X * 500 * 8 = 128 * 10^3$

So, a single zombie PC can send maximum **32 packets**.

b) The formula for calculating the required number of zombie PC to flood a target link is :

Number of zombie PC * uplink capacity of each PC = capacity of target link

0.5-Mbps :

$X * 128 * 10^3 = 0.5 * 10^6$
$=> X = 3.90625$

So, **4 zombie systems** are required to flood the target organization using a 0.5-Mbps link.

2-Mbps :

$X * 128 * 10^3 = 2 * 10^6$
$=> X = 15.625$

So, **16 zombie systems** are required to flood the target organization using a 2-Mbps link.

10-Mbps :

$X * 128 * 10^3 = 10 * 10^6$
$=> X = 78.125$

So, **78 zombie systems** are required to flood the target organization using a 10-Mbps link.

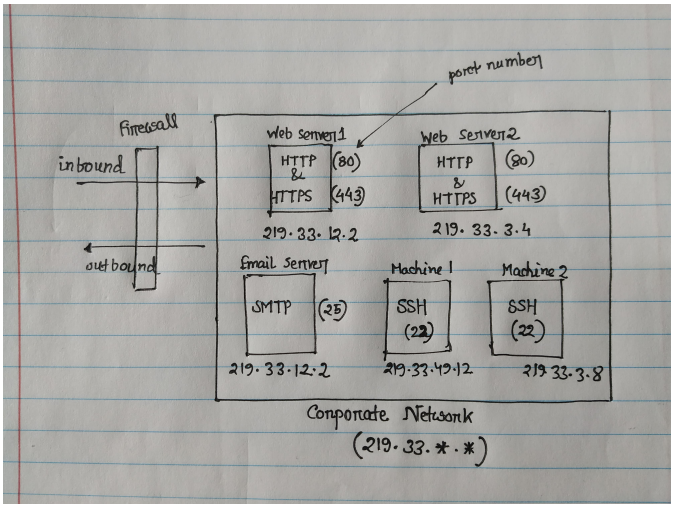
c) With given reports of botnets composed of many thousands of zombie systems, I can conclude that it is possible to launch the DDoS attacks on multiple organizations simultaneously.

It is also possible to launch the DDoS attacks on a major organization with multiple, much larger network links than we have considered in these problems.

From the problem I can easily say that if an organization has 3 network links of 0.5 Mbps, 2 Mbps and 10 Mbps, then botnets composed of (4+16+78) = 98 zombies can easily flood the whole organization. So, I can increase the number of zombie PCs and also their uplink capacity, it is possible to attack multiple larger network links simultaneously. 1000 zombies with X Kbps uplink can easily flood X Mbps of network link.

Problem 04

Solution:



Firewall Packet Filtering rules:

```
allow TCP * : */out ->  219.33.12.2 : 80/in
allow TCP * : */out ->  219.33.12.2 : 443/in
allow TCP * : */out ->  219.33.12.2 : 25/in
allow TCP * : */out ->  219.33.3.4 : 80/in
allow TCP * : */out ->  219.33.3.4 : 443/in
allow TCP * : */out ->  219.33.49.12 : 22/in
allow TCP * : */out ->  219.33.3.8 : 22/in
allow TCP 219.33.0.0/16 : */in -> * : */out
allow TCP * : */out -> 219.33.0.0/16 : */in (if ACK bit set)
drop  *   * : *  -> * : *
```