Software Security and Secure Coding Practices

Assignment

1. From Introduction session:

What is CVSS?

The Common Vulnerability Scoring System, or CVSS for short, is the first and only open framework for scoring the risk associated with vulnerabilities. CVSS is designed to rank information system vulnerabilities and provide an end user with a composite score representing the overall severity and risk the vulnerability presents. CVSS was created by The National Infrastructure Advisory Council (NIAC). Over the years it has become a very widely adopted scoring system and is used by such heavy hitters as the Department of Homeland Security, CERT, Cisco, Union Pacific, and Symantec to name but a few. CVSS is currently maintained by the Forum of Incident Response and Security Teams (FIRST), http://www.first.org, and was a combined effort involving many companies, including:

- CERT/CC
- Cisco Systems
- eBay
- Internet Security Systems
- Microsoft
- DHS/MITRE
- Qualys
- Symantec

A CVSS score is made up of three possible metric groups. Each group receives a score from 0 to 10, with 10 being the most severe. The three groups are: Cisco uses the CVSS system for its IPS signatures and IntelliShield reports.

- Base Group – Mandatory Score by vendor or analyst
- Temporal Group – Optional score by vendor or analyst
- Environmental Group – Optional score by end-user

Each group is made up of multiple separate categories. The sum of these categories make up the 0-10 final value for the group. The base group is made up of six categories. Using the CVSS calculator the end-user can enter parameters for the environmental group. This allows the end-user to receive a 0-10 score of the risk posed by a particular vulnerability in their specific environment.

What is Mitre CVE?

CVE stands for Common Vulnerabilities and Exposures. CVE is a glossary that classifies vulnerabilities. The glossary analyzes vulnerabilities and then uses the Common Vulnerability Scoring System (CVSS) to evaluate the threat level of a vulnerability. A CVE score is often used for prioritizing the security of vulnerabilities.

CVE was launched in 1999 by the MITRE corporation to identify and categorize vulnerabilities in software and firmware. CVE provides a free dictionary for organizations to improve their cyber security. MITRE is a nonprofit that operates federally funded research and development centers in the United States.

The CVE glossary is a project dedicated to tracking and cataloging vulnerabilities in consumer software and hardware. It is maintained by the MITRE Corporation with funding from the US Division of Homeland Security. Vulnerabilities are collected and cataloged using the Security Content Automation Protocol (SCAP). SCAP evaluates vulnerability information and assigns each vulnerability a unique identifier.

Once evaluated and identified, vulnerabilities are listed in the publicly available MITRE glossary. After listing, vulnerabilities are analyzed by the National Institute of Standards and Technology (NIST). All vulnerability and analysis information is then listed in NIST's National Vulnerability Database (NVD).

The CVE glossary was created as a baseline of communication and source of dialogue for the security and tech industries. CVE identifiers serve to standardize vulnerability information and unify communication amongst security professionals. Security advisories, vulnerability databases, and bug trackers all employ this standard.

To be categorized as a CVE vulnerability, vulnerabilities must meet a certain set of criteria. These criteria includes:

Independent of other issues: You must be able to fix the vulnerability independently of other issues.

Acknowledged by the vendor: The vulnerability is known by the vendor and is acknowledged to cause a security risk.

Is a proven risk: The vulnerability is submitted with evidence of security impact that violates the security policies of the vendor.

Affecting one codebase: Each product vulnerability gets a separate CVE. If vulnerabilities stem from shared protocols, standards, or libraries a separate CVE is assigned for each vendor affected. The exception is if there is no way to use the shared component without including the vulnerability.

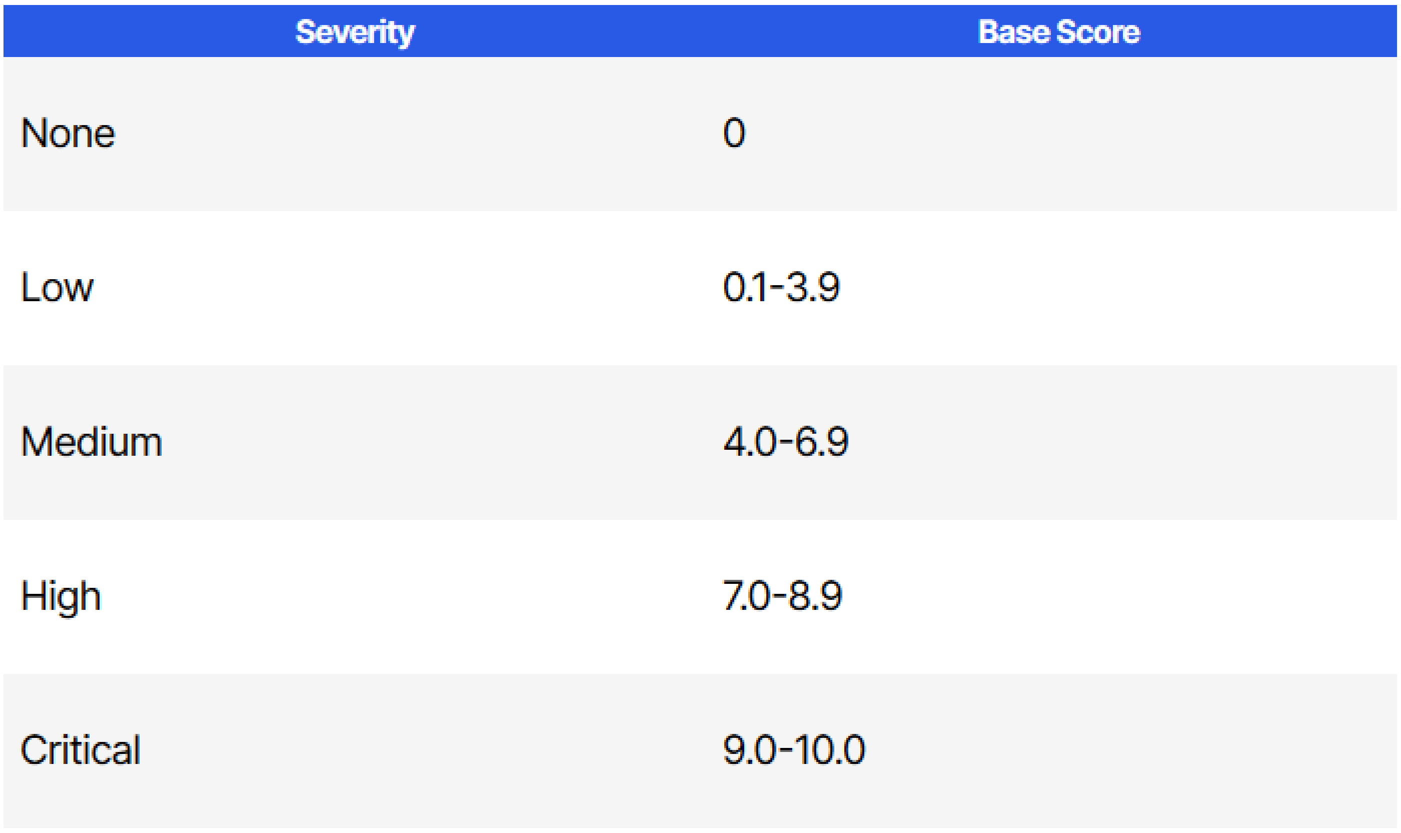
What is the difference between CVE and CVSS?

CVSS is the overall score assigned to a vulnerability. CVE is simply a list of all publicly disclosed vulnerabilities that includes the CVE ID, a description, dates, and comments. The CVSS score is not reported in the CVE listing – you must use the NVD to find assigned CVSS scores. The National Vulnerability Database (NVD) is a database, maintained by NIST, that is fully synchronized with the MITRE CVE list.

The CVE list feeds into the NVD, so both are synchronized at all times. The NVD provides enhanced information above and beyond what's in the CVE list, including patch availability and severity scores. NVD also provides an easier mechanism to search on a wide range of variables. Both CVE and NVD are sponsored by the US Federal Government and are available for free use by anyone.

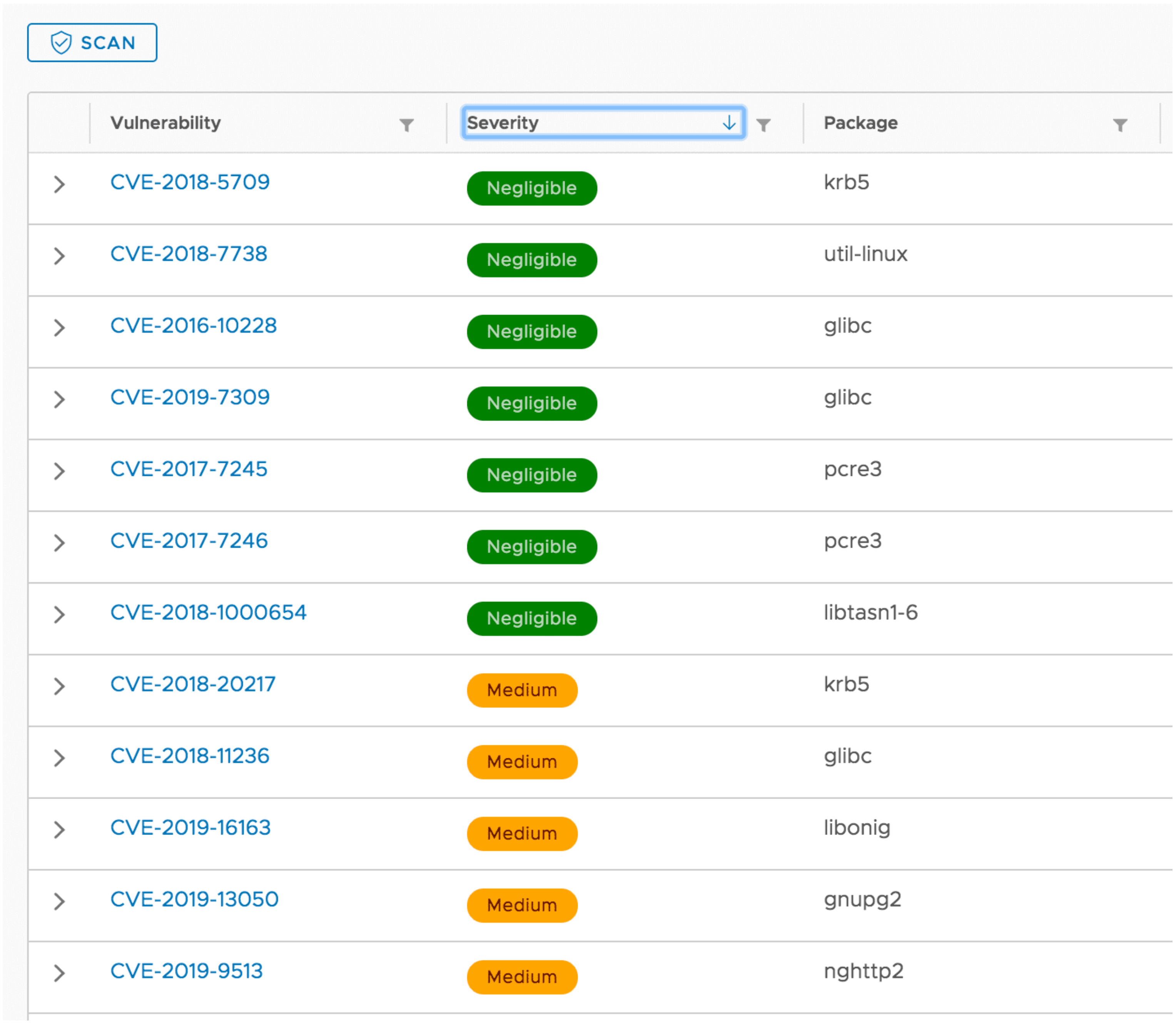
The CVSS score consists of three components – Base Metrics, Temporal Metrics, and Environmental Metrics. The NVD database includes all disclosed vulnerabilities, and includes a corresponding CVSS score. This score is typically comprised of Base Metrics only. Displayed only as the CVSS score, the fact that the reported number comprises only one of three CVSS metric groups can be misleading.

CVSS:



Severity	Base Score
None	0
Low	0.1-3.9
Medium	4.0-6.9
High	7.0-8.9
Critical	9.0-10.0

CVE:



	Vulnerability	Severity	Package
>	CVE-2018-5709	Negligible	krb5
>	CVE-2018-7738	Negligible	util-linux
>	CVE-2016-10228	Negligible	glibc
>	CVE-2019-7309	Negligible	glibc
>	CVE-2017-7245	Negligible	pcre3
>	CVE-2017-7246	Negligible	pcre3
>	CVE-2018-1000654	Negligible	libtasn1-6
>	CVE-2018-20217	Medium	krb5
>	CVE-2018-11236	Medium	glibc
>	CVE-2019-16163	Medium	libonig
>	CVE-2019-13050	Medium	gnupg2
>	CVE-2019-9513	Medium	nghttp2

Detailed exploration on WannayCry and Mirai

WannaCry Ransomware

- WannaCry is "ransomware" designed to spread quickly among computers on the same network, and encrypt files using strong encryption, enabling perpetrators to demand ransom from users to then decrypt the files.
- WannaCry is effective against computers with Microsoft Windows that do not have a security patch (patch "MS17-010") that Microsoft issued in March 2017. The Microsoft Windows vulnerability that WannaCry exploits – EternalBlue – was discovered by the NSA. The NSA developed it as an exploit to enable surveillance. This NSA hacking "tool" was stolen and released publicly on WikiLeaks earlier in 2017.
- These Microsoft Windows vulnerabilities – if unpatched – allow WannaCry code to spread quickly on computers that have not applied the security update. Once the vulnerability is exploited, the ransomware remotely accesses relevant computers and installs encryption software. WannaCry is able to find additional computers in the same network to infect by identifying and exploiting file-sharing arrangements a particular computer might have.
- After files are encrypted WannaCry demands ransom in the form of relatively small amounts of Bitcoin ($300-$600 per affected computer) to unlock the files.
- The ransomware is also understood to include additional malware (known as DoublePulsar), which allows hackers a "backdoor" to later gain further access to infected systems.
- WannaCry attacks were first recorded in Europe at 3:24 am EDT on the morning of Friday, May 12, 2017. Using social engineering, hackers embedded the WannaCry virus in .zip files sent to users as an email attachment.
- On Friday, May 12, 2017, a security researcher in London identified and purchased the domain of the web address where the first WannaCry strain was attempting to communicate. This effectively stopped the first attack, however over the weekend, hackers developed several additional strains. Some of these later versions did not have the "kill switch" requiring communication with a web address. These new strains exploit the same Windows vulnerability as the initial strain of WannaCry, but cannot be disabled as easily.

Known Affected Companies

As of the morning of May 17, 2017, WannaCry has affected at least 100,000 organizations in 150 countries. The following represents an abbreviated list of these companies:

1. **West Bengal State Electricity Distribution Company**
2. **Iberdrola**
3. **Petrobras**
4. **Gas Natural**
5. **PetroChina gas stations**
6. **Telefonica**
7. **Portugal Telecom**

8. **MegaFon**
9. **Telenor Hungary**
10. **FedEx**
11. **Renault**
12. **Nissan**
13. **Deutsche Bahn**
14. **Russian Railways**
15. **Sberbank**
16. **Bank Of China**
17. **Singapore malls**
18. **Sandvik**
19. **NHS**
20. **Russian Interior Ministry**
21. **Andhra Pradesh (Indian police)**
22. **Chinese traffic police, immigration and public security bureaus**
23. **Brazil Foreign Ministry, social security systems and court systems**
24. **Russia Central Bank**

Legal Issues and Response

- Organizations responding to this attack thus far appear to be viewing it as an IT/technical issue, not yet a legal one. Some companies have retained forensic firms to assist in responding and remediating their systems. There have not yet been reports of any exfiltration capabilities associated with this strain of ransomware.
- In the short term, potential legal liability may stem from data integrity and data availability risks and resulting business interruption (e.g., not having the necessary data to continue manufacturing or other industrial operations, perform surgeries or otherwise service customers), or inability to comply with legal or other obligations (e.g., a hospital's inability to substantiate medical procedures or obtain payment for procedures, or an employer's inability to pay employees).
- To the extent the attack includes the installation of back doors that may allow for broader unauthorized access to systems, personal and confidential information may be exposed, which could lead to compliance obligations, regulatory scrutiny and litigation risk. Moreover, at least one regulator – the U.S. Department of Health and Human Services – has indicated that the encryption of personal information itself could be a form of "unauthorized acquisition" (i.e., a "security breach"), which could trigger legal obligations (e.g. breach notification).
- In the long term, affected companies may face regulatory action and shareholder and director suits on allegations of having failed to maintain up to date security on their systems, and failure to disclose the resulting risks. The viability of such investigations and suits will depend on the ultimate financial (or bodily injury) impact of the attack on business operations.
- Moreover, for publicly-traded companies, based on SEC guidance on cyber security, to the extent that this attack poses a material financial risk to companies, or if companies are vulnerable to a similar attack that could have a material financial impact, it may be necessary to make certain cyber-related disclosures in the companies' financial statements.

Mirai

- The Mirai botnet is made up of IoT devices that have been infected with Mirai malware, a malware built to find and infect IoT devices using default passwords, and to launch distributed denial of service attacks. The Mirai malware is so serious about its dirty work that it will actually remove other malware found on the device.
- Some of 2016's most notorious DDoS attacks came courtesy of the Mirai botnet. First came the 620 Gbps attack on online security blogger Brian Krebs, then called the biggest DDoS attack in history. Then came the 1 Tbps attack on French hosting provider OVH, which replaced the Brian Krebs attack as the biggest attack in history. Following that was the 1.2 Tbps attack on DNS provider Dyn that yanked PayPal, Spotify, Netflix, Twitter and other major websites and platforms off the internet. The Dyn attack is still currently known as the biggest DDoS attack in history.
- The estimates of the number of IoT devices snared up in the Mirai botnet started around 50,000, jumping to 100,000 and then 150,000. A pair of hackers is now offering Mirai botnet-powered DDoS for hire services, claiming 400,000 infected devices.

As defined by DDoS mitigation provider Incapsula, a botnet is a group of internet connected devices that have been hijacked through malware so they can be remotely controlled, often without the owners' knowledge. Once a botnet has been assembled it can be used for a number of malicious purposes, most notably for distributed denial of service attacks, which use the tremendous number of devices in a botnet to direct malicious traffic at a target website or server in order to overwhelm it and render it unusable for legitimate users.

Traditionally, botnets have been largely made up of infected computers, but with the way the internet has evolved there are now many more internet-connected devices for attackers to choose from. The newest trend in botnets comes courtesy of the Internet of Things (IoT) – all those innovative internet-connected devices that are revolutionizing homes, commercial establishments and public spaces around the world.

This is a dangerous innovation because security on these devices is lacking, to say the least. Security so far has not proven to be a major priority when it comes to the development of these devices, and rare is the consumer who thinks to secure their smart appliances and fancy new gadgets. The Mirai botnet and other IoT botnets are taking advantage of these oversights, assembling massive zombie armies that are now being unleashed on the internet.

Vulnerabilities

The DDoS attacks being launched by Mirai and other IoT botnets have major consequences that ripple across the internet, costing organizations incredible amounts of money and causing widespread frustration and anger amongst users who are unable to access the websites they need.

Even if an IoT device owner were to somehow not care that their device is being used by remote attackers to wreak havoc across the internet, there are other aspects of malware infection that should be concerned. Major ones. If an attacker is able to use a default password to enlist a device as part of a botnet, an attacker is also able to use a default password to take control of the device, accessing data and other sensitive information, possibly even audio and video feeds in the case of CCTV cameras, baby monitors, nanny cams and more.

You can check if you have a device vulnerable to the Mirai malware by using this TCP/IP scanner. Regardless of what the scanner says, if your devices have default passwords, they need to be changed. No exceptions. This will protect you and your family as well as the internet at large. You also need to disable all remote or WAN access to your IoT devices. This open port finder is a good tool for checking for remote access capabilities on SSH , Telnet and HTTP/HTTPS ports.

2. From Secure Coding in C, C++ session:

Que. Implement a Linked List using C programming. Each node should maintain (i) username, (ii) password and (iii)session active / timedout parameters. Take the inputs from the user and properly sanitize before using them in the program. Use dynamic memory allocation for username and password of each node. Before quitting the application, clear sensitive information (passwords) from the linked list and assign NULL pointers after freeing them.

Program:

```
#include <stdio.h>
#include <stdlib.h>
typedef struct list *ptr;
typedef struct list{
   char *username;
   char *password;
   ptr next;
}node;  /*describes a linked list.*/
void insert(ptr *H, char c);
void freeList(ptr *H);
void printList(ptr *H);

int main(){
char c;
printf("enter a string\n");
ptr H=(ptr)malloc(sizeof(node));/*create an empty list. this is its head.*/
while ((c=getchar())!=EOF && c!='\n'){
   insert(&H,c);
   }
printList(&H); /*print the list*/
freeList(&H); /*free the list*/
printf("\n");
return 0;
}
void insert(ptr *H, char c){
   ptr p1;
   p1=*H;
   ptr T=(ptr)malloc(sizeof(node)); /*try to allocate a new memory cell*/
   if (!T)
   {
      printList(H);
      freeList(H); /*we should not free H because we will
         lose the list we've created.*/
   }
   else
   {

      printf("/n Enter Username" );
      scanf("%d",& T->username);
      printf("/n Enter Password" );
      scanf("%d",& T->username);

      while(p1->next)
      {
         p1=p1->next;
      }
      p1->next=T; /*add T to the end of the linked list*/

   }


}
void freeList(ptr *H){
   ptr p1; /*helper pointer*/
   while(*H){     /*while H points to a certain node*/
   p1=*H;
   (*H)=p1->next;
   free(p1);
   }
}
void printList(ptr *H){ /*a copy of H is sent so no problem working with it.*/
   ptr p1=*H; printf("string is: \n");
   while (p1) /*while H is not null       */
   {
      printf("%c", p1->data);
      p1=p1->next;
   }

}
```

3. From Defense Mechanisms session:

List all the protection mechanisms employed by GCC toolchain and Linux Operating System

STANDARD BASIC SECURITY FEATURES

For the basic security features, Linux has password authentication, file system discretionary access control, and security auditing. These three fundamental features are necessary to achieve a security evaluation at the C2 level . Most commercial server-level operating systems, including AIX (IBM), Windows NT, and Solaris, have been certified to this C2 level. By expanding the basic standard security features we have:

1. User and group separation
2. File system security
3. Audit trails
4. PAM authentication

LINUX SECURITY EXTENSIONS

The Linux family of products has provided a highly secure environment since its original delivery in early 2002. The features discussed in the following sections have been added to the Linux OS. For example, the Red Hat Enterprise Linux Update 3, shipped in September 2004 contains:

1. ExecShield – With the **N**o e**X**ecute (NX) , or e**X**ecute **D**isable (XD) and Segmentation features.
2. Position Independent Executables (PIE)

Then, in Red Hat Enterprise Linux v.4, shipped in February 2005 contains the following security features:

1. SecurityEnhanced Linux (SELinux)
2. Compiler and library enhancements
3. Advanced glibc memory corruption checker
4. Secure version of the printf and other string manipulation functions
5. gcc buffer bound checking

In term of the Linux OS security breaches, most of the problems originated from the buffer overflow issue. The buffer overflow exploits unprotected and or unchecked fixed sized buffers, overwriting the area beyond it. The overwritten area may be filled with the malicious codes, containing code that pointing to the customized return address. There are many buffer locations in the memory area. It is used to temporarily store data.

COMPILER AND LIBRARY ENHANCEMENTS
Another feature, during late 2004, Red Hat developed a new group of features that improve buffer management and security for inclusion in Red Hat Enterprise Linux v.4. After that, these features adopted by other Linux distribution that compiled using new GLIBC libraries.

THE GLIB MEMORY CORRUPTION CHECKS
The GLIBC memory allocator functions now perform a set of internal sanity check to detect double freeing of memory and heap buffer overflows. With these checks, regular application bugs and security exploit attempts that use these techniques are detected and the program will be instantly aborted to avoid the possibility of the exploit succeeding. With these checks, double free exploits become entirely impossible and all standard, generic heap type overflow techniques are blocked. These detection functions execute prior to the ExecShield features mentioned earlier. In many cases exploit attempts of this type would ultimately be blocked by ExecShield, but these memory corruption checks provide an extra level of security because the earlier an exploit attempt is detected and aborted the better.

THE printf() FORMAT STRING EXPLOIT PREVENTION
The printf() format string exploits were popular around several years ago when the technique was first exposed. Compared to other C functions, the printf() function is a variadic type function that can accept variable number of parameter. The printf() format string exploits abuse a bug in programs that have a faulty call to the standard printf() function, caused by a formatting parameter. When applications are compiled with the "-D_FORTIFY_SOURCE=2" compiler option, the printf() function will check that this rare formatting comes from guaranteed trusted sources and will abort the program if that is not the case, thus preventing printf() format exploits entirely. Othersecure version of the string and character manipulation, standard C and C++ functions also were introduced and implemented in newer version of compilers.

GCC BUFFER BOUND CHECKING
This feature detects many potential buffer overflow conditions and is the most important of the buffer management improvements. An enhancement has been added to the GCC compiler so that if the size of the destination buffer can be detected at compile time, function such as strcpy(), memcpy() and printf() will use a checking variant of these functions detects if the buffer will overflow. If that

happens the program is aborted immediately. While GCC cannot always detect the size of the destination buffer for example, it is not possible for dynamically allocated buffers. Buffer allocation errors usually occur with the types of buffer that can be detected by GCC. The result is that a large percentage of buffer overflow errors are prevented immediately.

In the meantime the secure version of the C functions [17] that involve the use of the buffer, have been added to the standard C library. The secure version can be recognized from the _s suffix for example, printf_s() and strcpy_s(). When using these secure versions, programmer need to include additional parameter for the buffer size. However, the implementation still depends on the programmers whether to use the secure version or not. You will find warning that stated a non-secure version will be marked as deprecated when using those functions for newer compilers.

DISCRETIONARY ACCESS CONTROL AND MANDATORY ACCESS CONTROLS

As discussed previously standard Linux file permissions use the Discretionary Access Control (DAC) model. Under DAC, files are owned by a user and that user has full control over them, including the ability to grant access permissions to other users. The root account has full control over every file on the entire system. An attacker who penetrates an account can do anything with the files owned by that user. For example, an attacker who compromises a web server has full control over all files owned by the web server account. If an application runs under the context of the root user, an attacker penetrating it now has full control over the entire system.

SELinux (discussed later) supplements Discretionary Access Control with Mandatory Access Control (MAC). Under MAC, the Administrator writes a security policy that defines access rights for all users and applications. MAC in effect provides each application with a virtual sandbox that only allows the application to perform the tasks it is designed for and explicitly allowed in the security policy to perform. For example, the web server process may only be able to read web published files and serve them on a specified network port. An attacker penetrating it will not be able to perform any activities not expressly permitted to the process by the security policy, even if the process is running as the root user. Files are assigned a security context that determines what specific processes can do with them, and the allowable actions are much more finely defined than the standard Linux read/write/execute controls. For example, a web served file would have a context allowing the apache process to read it but not execute or make changes to it, while the log files would be appendable but not readable or otherwise changeable by apache.

Network ports are also assigned a context, which can prevent penetrated applications from using ports not permitted to them by security policy. Standard Linux permissions are still present on the system, and will be consulted before the SELinux policy when access attempts are made. If the standard permissions would deny access, access is simply denied and SELinux is not consulted at all. If the standard file permissions would allow access, the SELinux policy is consulted and access is either allowed or denied based on the security contexts of the source process and the targeted object.

OTHER LINUX SECURITY EXTENSIONS

Many Linux distributions have been hardened by the security extensions. The main modifications to these systems were the addition of MAC model as discussed previously. It is called a multi-level security (MLS) model, and auditing capabilities. A MLS model is designed to prevent the leakage of data to unauthorized subjects, but does not address the integrity of the system. That is, such systems prevent the leakage of data, but do not prevent the exploitation of bugs by user on data from untrusted sources that may compromise the entire system.

Firstly, the TCPA consortium aims to provide hardware support for reliable system integrity verification, and tamperproof platforms, such as IBM's 4758 enable the installation of security services in hostile environments.

A large number of advances have been made for Linux as well. The Bastille Linux [30] and Immunix [31] provide hardening tools for preventing bugs from taking over system services that run as root. For example, Immunix is a family of tools designed to cause system services to fail safely when one of a variety of common vulnerability types such as buffer overflow attack happens.

Trustix , OpenWall and HP Secure OS Software for Linux provide a variety of tools to improve Linux security such as TripWire, and encrypted file systems.

Many of these systems provide MAC policy models for files, but only Argus PitBull provides a model that enables control of network objects as well. The MAC policy models used in these systems are either significantly limited or complex and lack supporting management tools.

Trustix does provide support for assisting system administrators in installing a system with minimal services and preventing accidental initiation of new services.

A fundamental problem with all of the approaches above is that they require kernel modifications to provide the desired authorization flexibility and performance. This is because the actual objects and operations performed are determined deep inside the kernel, so the kernel must be changed to ensure that the intended policy is enforced. Because each system uses different, adhoc kernel modifications none will be accepted into the base kernel.

In kernel 2.6, the Linux Security Modules (LSM), framework adds authorization hooks into the base Linux kernel that intends to cover every controlled operation in Linux kernel. The hooks are independent of the authorization policy, so a variety of MAC policies can be supported. Fundamentally, LSM is only an authorization framework, so many other features are necessary to build a secure Linux system. In this case, we need to identify the tasks that can be performed to install an initial Trusted Computing Base (TCB) for a secure Linux system. Then, we point out where the Linux extension systems provide a similar functionality.

The LSM community is led by Wirex with DARPA sponsorship. NAI Labs with NSA sponsorship has been another major contributor, but many other companies such as IBM and individuals from around the world have also provided input to the LSM framework.

Given a Linux binary search the internet for tools that would let you find the protection mechanism enabled on the binary (attached binaries filenames: onemore, layout)

Let's start with the ar utility, which is used to create, extract and modify archives. Its main use is in building static libraries, which we will examine with an example. We will create two .c files, with a function defined in each.

The first is foo.c:

```
void foo(int *i){
    *i=*i+5;
}
```

The second is bar.c:

```
void bar(int *i){
    *i=*i*5;
}
```

ld tool

Linking is the process of combining various pieces of code and data together to form a single executable image (that can be loaded) in memory. Linking can be done at compile time or runtime. GCC performs linking in the background; compile with the `-v` option, and you will see many background details, including the linking.
To understand what happened during compilation, you must know how to use a linker manually — so let's compile a simple "Hello World" C program without linking it (as before, `-c`) with `gcc -c hello.c`. Let's manually do linking of the resultant object file `hello.o`.

```
$  ld -o hello -dynamic-linker /lib/ld-linux.so.2 /usr/lib/crt*.o hello.o -lc
```

In the output executable `hello`, `/usr/lib/crt*.o`, `hello.o`, and the C library (`-lc`) are statically linked to (copied into) `hello`, and `/lib/ld-linux.so.2` are linked dynamically.

Objdump

The next utility I will cover is `objdump`. As a kernel developer, most of the time I have to deal with kernel crashes. Debugging the kernel needs a good understanding of the relation between C and assembly.

For learning, working on a kernel crash would be time-consuming, so let's start with a user-space program. Let's compile `test.c` with debugging symbols (the option `-g`) and generate its assembly code with `objdump` as follows:

```
$ gcc -g test.c -o test
$ objdump -S ./test > asm
```

strings

This tool prints the printable characters from a file and can help while debugging a C program. Also, if you get an executable file from the Internet or other source, which contains malicious code, you can get an idea of what it does by looking at printable characters. The tool `strings` is the solution.

Here, run it on your own test executable:

```
$ strings ./test
/lib/ld-linux.so.2
q?Hm
__gmon_start__
libc.so.6
_IO_stdin_used
printf
memcpy
__libc_start_main
GLIBC_2.0
PTRh
QVhv
[^_]
[test1] global_int2 = %d
Hello
[test2] global_int1 = %d
[test3] global_string = %s
```

Strip

Another important tool is `strip`. While working on embedded systems projects, you must use memory (both RAM and storage) carefully. A few extra kilobytes can create problems. When you develop code, you need the symbols in it for debugging — but when you deploy it, there is no sense leaving the symbols in the executable file and wasting precious kilobytes of memory. So strip these symbols before deploying.

Let's compile `test.c` again (`gcc test.c -o test`) and check:

```
$ file test
test: ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV),
dynamically linked (uses shared libs), for GNU/Linux 2.6.15, not
stripped

$ ls -lh test
```

```
-rwxr-xr-x 1 manoj manoj 7.2K 2011-09-14 00:44 test
```

size

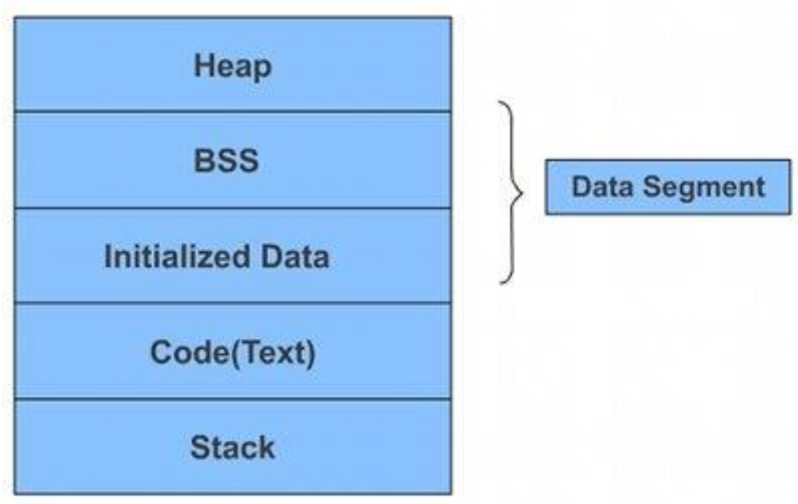
Finally, one more command that I need to cover is `size`, to get the sizes of different memory sections of an ELF file (executable):

```
$ size a.out
   text            data            bss      dec      hex filename
   1192      272      24     1488      5d0 a.out
```

In this output, `text`, `data` and `bss` should be familiar; `dec` is the total size of all sections (`text` + `data` + `bss`).

nm

The following diagram represents the memory segments of a C program (heap, data, code and stack), which C programmers must consider.



Compile the attached C code to disable stack canary (attached C file: layout.c)

Output without `-fstack-protector-all`:

```
$ objdump -dj .text test | grep -A7 "<test>:"
00000000004004ed <test>:
  4004ed:    55                          push   %rbp
  4004ee:    48 89 e5                    mov    %rsp,%rbp
  4004f1:    89 7d fc                    mov    %edi,-0x4(%rbp)
  4004f4:    8b 45 fc                    mov    -0x4(%rbp),%eax
  4004f7:    5d                          pop    %rbp
  4004f8:    c3                          retq
```

4. From Android session:

Develop an Android application with a public Broadcast Receiver. Receive a broadcast intent from another application to perform an action in your application. Properly sanitize the data received from the intent before using the data in your application.

Program to Create a Broadcast Receiver:

activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="https://schemas.android.com/apk/res/android"
   xmlns:tools="https://schemas.android.com/tools"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   android:paddingBottom="@dimen/activity_vertical_margin"
   android:paddingLeft="@dimen/activity_horizontal_margin"
   android:paddingRight="@dimen/activity_horizontal_margin"
   android:paddingTop="@dimen/activity_vertical_margin"
   tools:context="com.journaldev.broadcastreceiver.MainActivity">
   <Button
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:id="@+id/button"
      android:text="Send Broadcast"
      android:layout_centerVertical="true"
      android:layout_centerHorizontal="true" />
</RelativeLayout>
```

MainActivity.java

```
package com.journaldev.broadcastreceiver;
```

```java
import android.content.Intent;
import android.content.IntentFilter;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.util.Log;
import android.view.View;
import android.widget.Button;

import butterknife.ButterKnife;
import butterknife.InjectView;
import butterknife.OnClick;

public class MainActivity extends AppCompatActivity {
    ConnectionReceiver receiver;
    IntentFilter intentFilter;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        ButterKnife.inject(this);
        receiver = new ConnectionReceiver();
        intentFilter = new IntentFilter("com.journaldev.broadcastreceiver.SOME_ACTION");

    }
    @Override
    protected void onResume() {
        super.onResume();
        registerReceiver(receiver, intentFilter);
    }
    @Override
    protected void onDestroy() {
        super.onDestroy();
        unregisterReceiver(receiver);
    }
    @OnClick(R.id.button)

    void someMethod() {
        Intent intent = new Intent("com.journaldev.broadcastreceiver.SOME_ACTION");
        sendBroadcast(intent);
    }

}
```

AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="https://schemas.android.com/apk/res/android"
    package="com.journaldev.broadcastreceiver">

    <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />

            </intent-filter>
        </activity>

        <receiver android:name=".ConnectionReceiver">
            <intent-filter>
                <action android:name="android.net.conn.CONNECTIVITY_CHANGE" />
            </intent-filter>
        </receiver>

    </application>
</manifest>
```

ConnectionReceiver.java

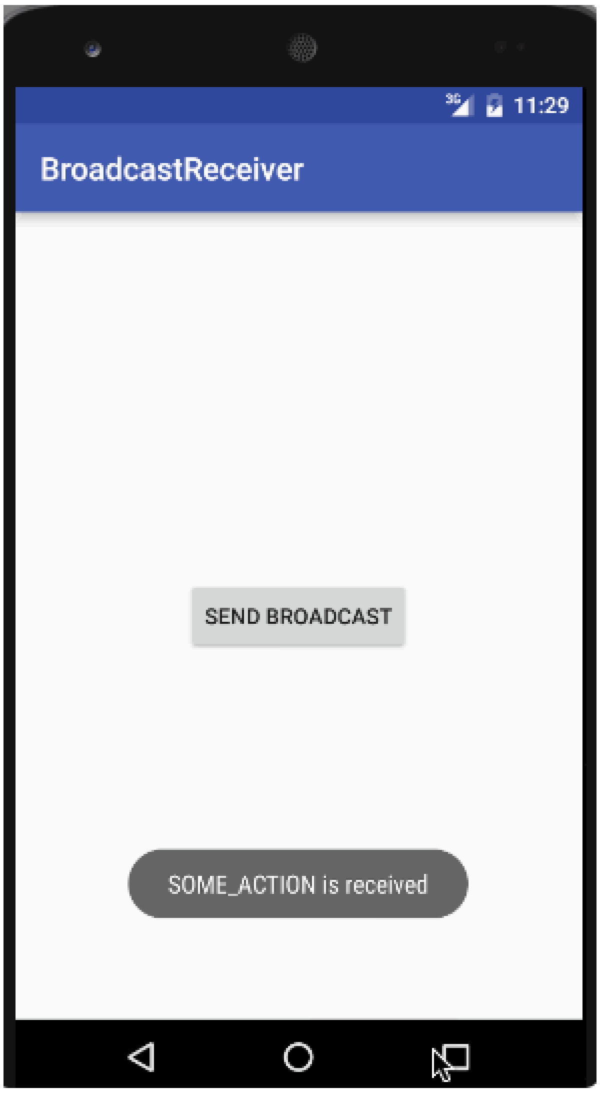
```
public class ConnectionReceiver extends BroadcastReceiver {
    @Override
    public void onReceive(Context context, Intent intent) {

        Log.d("API123",""+intent.getAction());

        if(intent.getAction().equals("com.journaldev.broadcastreceiver.SOME_ACTION"))
            Toast.makeText(context, "SOME_ACTION is received", Toast.LENGTH_LONG).show();

        else {
            ConnectivityManager cm =
                    (ConnectivityManager) context.getSystemService(Context.CONNECTIVITY_SERVICE);

            NetworkInfo activeNetwork = cm.getActiveNetworkInfo();
            boolean isConnected = activeNetwork != null &&
                    activeNetwork.isConnectedOrConnecting();
            if (isConnected) {
                try {
                    Toast.makeText(context, "Network is connected", Toast.LENGTH_LONG).show();
                } catch (Exception e) {
                    e.printStackTrace();
                }
            } else {
                Toast.makeText(context, "Network is changed or reconnected",
Toast.LENGTH_LONG).show();
            }
        }
    }
}
```



Now, make the broadcast receiver private to the application and try to invoke it from an external application. Observe and analyze the results.

To make the broadcast receiver unavailable to external applications, I added the attribute android:exported=false in the manifest. When we send a broadcast, it is possible for the external applications too to receive them. This is be prevented by specifying this limitation.

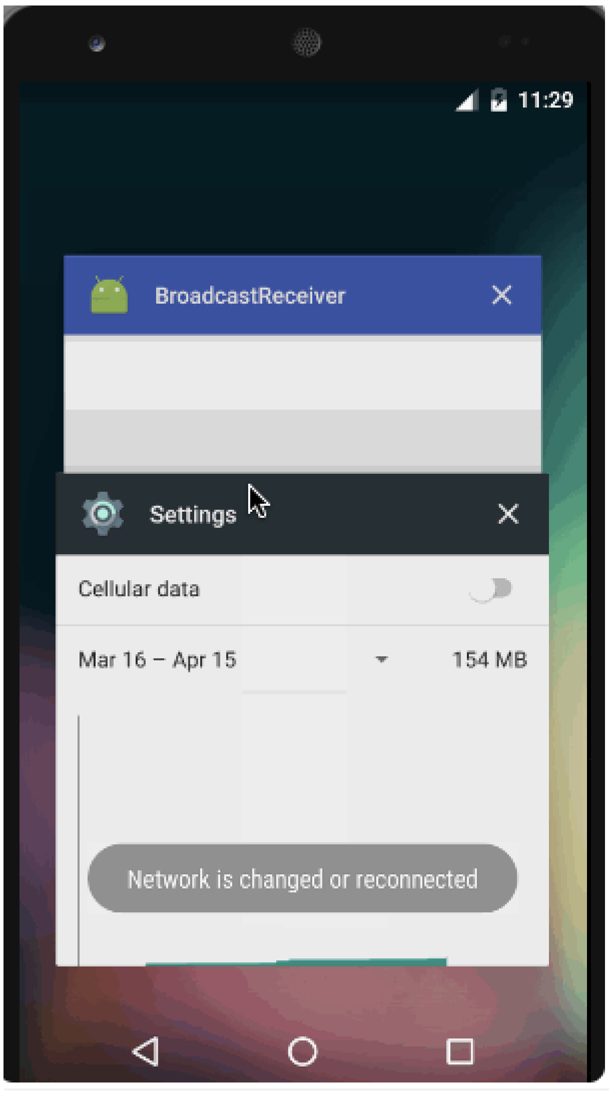
```
<?xml version="1.0" encoding="utf-8"?>
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="https://schemas.android.com/apk/res/android"
    package="com.journaldev.broadcastreceiver">

    <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity android:name=".MainActivity"
          android:export="false" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>

        <receiver android:name=".ConnectionReceiver">
            <intent-filter>
                <action android:name="android.net.conn.CONNECTIVITY_CHANGE" />
            </intent-filter>
        </receiver>

    </application>
</manifest>
```



Network is changed or reconnected

From Web Security Session:

From Web Security Session: a. Develop a secure registration form with fields, Name, Email Address, Phone number, Address, Gender and photo upload. Insert the data into the database after registration and upon successful registration, acknowledge the registration by showing the data entered by user in next page. i. HTML FORM WITH 5 FIELDS ii. INSERT INTO DATABASE iii. SHOW INSERTED DATA IN BROWSER iv. You can use any server side programming language of your choice

Registration Form

```
<form id='register' action='register.php' method='post'
   accept-charset='UTF-8'>
<fieldset >
<legend>Register</legend>
<input type='hidden' name='submitted' id='submitted' value='1'/>
<label for='name' >Your Full Name*: </label>
<input type='text' name='name' id='name' maxlength="50" />
<label for='email' >Email Address*:</label>
<input type='text' name='email' id='email' maxlength="50" />
<label for=’address’ >Address*:</label>
<input type='text' name='address' id='address' maxlength="50" />
<label for='gender' >Gender*:</label>
<input type='gender' name='gender' id='gender' maxlength="50" />
<label for=’phoneno’ >phoneno*:</label>
<input type='phoneno' name='phoneno' id='phoneno' maxlength="50" />

<label for='photo' >Uploadphoto*:</label>
<input type="file" id="inputImage" Name="photo">
<input type='submit' name='Submit' value='Submit' />
</fieldset>
</form>
```

Form validation

```
var frmvalidator  = new Validator("register");
frmvalidator.EnableOnPageErrorDisplay();
frmvalidator.EnableMsgsTogether();
frmvalidator.addValidation("name","req","Please provide your name");
frmvalidator.addValidation("email","req","Please provide your email address");
frmvalidator.addValidation("address","req","Please provide a address");
frmvalidator.addValidation("gender","req","Please provide a gender");
frmvalidator.addValidation("phoneno","req","Please provide a phoneno");
frmvalidator.addValidation("photo","req","Please provide a photo");
```

Saving the data in the database

```
function SaveToDatabase(&$formvars)
   {
      if(!$this->DBLogin())
      {
         $this->HandleError("Database login failed!");
         return false;
      }
      if(!$this->Ensuretable())
      {
         return false;
      }
      if(!$this->IsFieldUnique($formvars,'email'))
      {
         $this->HandleError("This email is already registered");
         return false;
      }
      if(!$this->IsFieldUnique($formvars,'address'))
      {
         $this->HandleError("This address is empty");
         return false;
      }
      if(!$this->IsFieldUnique($formvars,'gender'))
      {
         $this->HandleError(" gender is empty");
         return false;
      }



      if(!$this->IsFieldUnique($formvars,’phoneno’))
      {
         $this->HandleError("This phone is already used. Please try another phoneno");
         return false;
      }
      if(!$this->InsertIntoDB($formvars))
      {
         $this->HandleError("Inserting to Database failed!");
         return false;
      }
      return true;
   }
```

The database table structure

```
function CreateTable()
{
   $qry = "Create Table $this->tablename (".
         "id_user INT NOT NULL AUTO_INCREMENT ,".
         "name VARCHAR( 128 ) NOT NULL ,".
         "email VARCHAR( 64 ) NOT NULL ,".
         "phone_number VARCHAR( 16 ) NOT NULL ,".
         "gender VARCHAR( 16 ) NOT NULL ,".
         "address VARCHAR( 32 ) NOT NULL ,".
         "confirmcode VARCHAR(32) ,".
         "PRIMARY KEY ( id_user )".
         ")";

   if(!mysql_query($qry,$this->connection))
   {
      $this->HandleDBError("Error creating the table \nquery was\n $qry");
      return false;
   }
   return true;
}
```

Inserting the registration to the table

```
function InsertIntoDB(&$formvars)
{
   $confirmcode = $this->MakeConfirmationMd5($formvars['email']);


   $insert_query = 'insert into '.$this->tablename.'(
         name,
         email,
         address,
         phoneno,
         gender,
                 photo,
         )
         values
         (
         "' . $this->SanitizeForSQL($formvars['name']) . '",
         "' . $this->SanitizeForSQL($formvars['email']) . '",
         "' . $this->SanitizeForSQL($formvars['address']) . '",
         "' . $this->SanitizeForSQL($formvars['phoneno']) . '",
         "' . $this->SanitizeForSQL($formvars['gender']) . '",
         "' . $this->SanitizeForSQL($formvars[photo]) . '",
         )';
   if(!mysql_query( $insert_query ,$this->connection)){
      $this->HandleDBError("Error inserting data to the table\nquery:$insert_query");
      return false;  }
   return true;
}
```

From Secure SDLC session:

Consider the scenario of the Integrity checking application, which checks the integrity of the system in terms of registry entries, filesystem entries etc. The following is the description and the components involved:

- **It has an admin console and integrity calculating host component (each in two different systems)**
- **Admin console**
- **Builds from a configuration file**

- **Admin can instruct to get the integrity measurements of a system**
- **He can store them in the database and also verify with the previously stored details**
- **Can also register the integrity of a new system**
- **Integrity calculating host component**
- **Reads the registry and filesystem details, calculates the hash and sends it to the admin console on request.**

Do a threat model for the above scenario and also analyze the threats and propose the mitigation rules

Input

The following input is useful for threat modeling:

- Use cases and usage scenarios
- Data flows
- Data schemas
- Deployment diagrams

Although all of these are useful, none of them are essential. However, you do need to have knowledge of your application's primary function and architecture.

Output

The output of the threat modeling activity is a threat model. The main items captured by the threat model include the following:

- A list of threats
- A list of vulnerabilities

Summary of Steps

The five major threat modeling steps are shown in Figure 1.

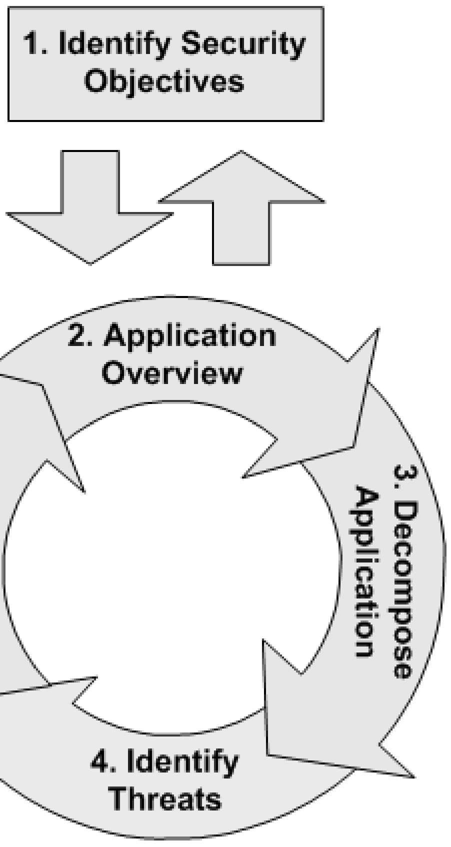


Figure 1. The iterative threat modeling process

These steps are:

1. **Identify security objectives.** Clear objectives help you to focus the threat modeling activity and determine how much effort to spend on subsequent steps.
2. **Create an application overview.** Itemizing your application's important characteristics and actors helps you to identify relevant threats during step 4.
3. **Decompose your application.** A detailed understanding of the mechanics of your application makes it easier for you to uncover more relevant and more detailed threats.
4. **Identify threats.** Use details from steps 2 and 3 to identify threats relevant to your application scenario and context.
5. **Identify vulnerabilities.** Review the layers of your application to identify weaknesses related to your threats. Use vulnerability categories to help you focus on those areas where mistakes are most often made.

You add progressively more detail to your threat model as you move through your application development life cycle and discover more details about your application design. Because key resources identified in threat modeling are also likely to be key resources from a performance and functionality perspective, you can expect to revisit and adjust your model as you balance all of your needs. This is normal and is a valuable outcome of the process.

-BY SAURAV CHAUHAN

(sc07596@gmail.com)