Threat, Control and Vulnerabilities
<table>
<thead>
<tr>
<th>Threat</th>
<th>A set of circumstances that has the potential to cause loss or harm</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vulnerabilities</td>
<td>Weakness in security system</td>
</tr>
<tr>
<td>Control</td>
<td>Protective measure (action, device, procedure or technique)</td>
</tr>
</tbody>
</table>

Relationship:

A *threat* is blocked by *control* of a *vulnerability*. 
Non-malicious Program Errors

• Programmers make many mistakes – most of which are unintentional and non-malicious

• It causes program malfunctions but do not lead to more serious security vulnerabilities

• Three classic error types:
  o Buffer Overflows
  o Incomplete Mediation
  o Time-of-Check to Time-of-Use (TOCTTOU)
Buffer Overflows

- Computing equivalent of trying to pour two liters of water into a one-liter pitcher: Some water is going to spill out and make a mess!

- Definition: A buffer (or array or string) is a space in which data can be held. A buffer resides in memory. Because memory is finite, a buffer’s capacity is finite.
• For this reason, programmer must declare the buffer’s maximum size – compiler can set aside that amount of space.

• Consider:

```c
char sample[10];  // (sample[0] through sample[9])
sample[10] = 'B';  // (this subscript is out of bound)
```
• For example, suppose each of the ten elements of the array `sample` is filled with the letter A and the erroneous reference uses the letter B, as follow:

```plaintext
for (i=0; i<=9; i++)
    sample[i] = 'A';
sample[10] = 'B';
```
(a) Affects user’s data

(b) Affects user’s code

(c) Affects system data

(d) Affects system code
Security Implications:

- The attacker may replace code in the system space.
- Remember that every program is invoked by the OS, and the OS may run with higher privileges than those of regular programs.
- Thus, if the attacker can gain control by masquerading as the OS, the attacker can execute many commands in his/her own procedure.
Incomplete Mediation

• Another security problem that has been with us for decades.

• Consider the example of:


• The two parameters look like a telephone number and a date. Probably the client’s web browser enter those two values in their specified format for easy processing.
• What would happen if `parm2` were submitted as `1800Jan01?` Or `1800Feb30?` Or `2048Min32?` Or `1Aardvark2Many`?

• Something would likely fail!

• A routine’s failing on a data type error as it tried to handle a month named “Min” or even a year like 1800, that was out of range.
• Another possibility is that the receiving program would continue to execute but would generate a very wrong result.

• For example, imagine the amount of interest due today might have a default condition, deciding to treat the calculation since 1800. What a big amount!
• One way to address the potential problem is to try to anticipate them.
• For example, to prevent the use of nonsense data, the program can restrict choices only to valid ones.
• Or by using a drop-down box or choice list from which only the twelve conventional months would have been possible choices.
Security Implications:

• Unchecked data values represent a serious potential vulnerability.

• Consider this:


• What if…


• Surprise! It worked.
Time-of-Check to Time-of Use Errors

- It involves synchronization.
- To improve efficiency, modern processors and OS usually change the order in which instructions and procedures are executed.
- Instructions that appear to be adjacent may not actually be executed immediately after each other, either intentionally changed order or the effects of other processes in concurrent execution.
• We want to make sure that only those who should access an object are allowed that access.

<table>
<thead>
<tr>
<th>my_file</th>
<th>change byte 4 to “A”</th>
<th>Data Structure for File Access</th>
</tr>
</thead>
<tbody>
<tr>
<td>your_file</td>
<td>delete file</td>
<td>Modified Data</td>
</tr>
</tbody>
</table>

• To prevent: Ensure that critical parameters are not exposed during any loss of control. Or to allow no interruption during validation.
Viruses and Other Malicious Code

• Much of the work done by a program is invisible to users who are not likely to be aware of any malicious activity.
  • When was the last time you saw a bit?
  • Do you know in what form a document file is stored?
  • If you know a document resides somewhere on a disk, can you find it?
  • Can you tell if a game program does anything in addition to its expected interaction with you?
  • Which files are modified by a word processor when you create a document?
  • Which programs execute when you start your computer or open a web page?
• Users usually do not see computer data directly, malicious people can make programs serve as vehicles to access and change data and other programs.
• Why Worry About Malicious Code?
  • When you last installed a major software package, such as a word processor, a statistical package, or a plug-in from Internet, you ran one command, typically called INSTALL or SETUP.
  • From there, the installation program took control → create some files → write in other files → delete data and files → perhaps renaming a few
  • Thousands of bytes of programs and data are transferred, and hundreds of modification may be made to your existing files, all occurring without your explicit consent or knowledge.
### Types of Malicious Code

<table>
<thead>
<tr>
<th>Code Type</th>
<th>Characteristics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Virus</td>
<td>Attaches itself to program and propagates copies of itself to other programs</td>
</tr>
<tr>
<td>Trojan horse</td>
<td>Contains unexpected, additional functionality</td>
</tr>
<tr>
<td>Logic bomb</td>
<td>Triggers action when condition occurs</td>
</tr>
<tr>
<td>Time bomb</td>
<td>Triggers action when specified time occurs</td>
</tr>
<tr>
<td>Trapdoor</td>
<td>Allows unauthorized access to functionality</td>
</tr>
<tr>
<td>Worm</td>
<td>Propagates copies of itself through a network</td>
</tr>
<tr>
<td>Rabbit</td>
<td>Replicates itself without limit to exhaust resources</td>
</tr>
</tbody>
</table>
# Virus Effects and Causes

<table>
<thead>
<tr>
<th>Virus Effect</th>
<th>How It Is Caused</th>
</tr>
</thead>
</table>
| Attached to executable program | - Modify file directory  
                                 | - Write to executable program file                                               |
| Attach to date or control file | - Modify directory  
                                 | - Rewrite data  
                                 | - Append to data  
                                 | - Append data to self |
| Remain in memory             | - Intercept interrupt by modifying interrupt handler address table  
<pre><code>                             | - Load self in non-transient memory area                                      |
</code></pre>
<table>
<thead>
<tr>
<th>Section</th>
<th>Details</th>
</tr>
</thead>
</table>
| **Infect disks**       | • Intercept interrupt  
                        • Intercept OS call (to format disk, for example)  
                        • Modify system file  
                        • Modify ordinary executable program. |
| **Conceal self**       | • Intercept system calls that would reveal self and falsify result  
                        • Classify self as ‘hidden’ file |
| **Spread infection**   | • Infect boot sector  
                        • Infect systems program  
                        • Infect ordinary program |
| **Prevent deactivation** | • Activate before deactivating program and block deactivation  
                        • Store copy to re-infect after deactivation |
Prevention of Virus Infection

• Use only commercial software acquired from reliable, well-established vendors
• Test all new software on an isolated computer.
• Open attachments only when you know them to be safe.
• Make a recoverable system image and store it safely.
• Make and return backup copies of executable system files.
• Use virus detectors regularly and update them daily.
Controls Against Program Threats

Development Controls

• **Modularity**
  - Create a design or code in small, self-contained units called *components* or *modules*. It will isolate from the effects of other components.

• **Encapsulation**
  - Changes to an isolated component do not affect other components.
  - Easier to see where vulnerabilities may lie if they are isolated.

• **Information Hiding**
  - Each component hides its precise implementation or some other design decision from the others.
Anti-virus

- Malicious program
- Vulnerabilities
- Exploit (code)
- VIRUS
- Damage
- Isolate
- AV-vendor
**AV-vendor**

**Forensics – see why it happen**

**Distill characteristics**
- *Which apps it attacks?*
- *Which port it attacks?*
- *Binary sequence of that malicious code*

**AV-signature – to identify the existence of virus**

**Run application open file**
- *AV will check the binary sequence of the apps*

**AV-customer = YOU!!**
(we call it update)

Upload for their customer

**Zero-day Exploit**

Time between availability of exploit code and availability of counter-measure