**1) Teoría de Programación Orientada a Objetos (POO)**  
  
  
1.1) Concepto de Objeto: Atributos y Métodos  
*Un objeto es cualquier cosa real o abstracta de la cual nos interesa su comportamiento y que tiene una identidad única que la distingue de las demás.* Un objeto es una unidad atómica formada por la unión de estado+comportamiento.  
Un Objeto se compone de:  
   - *Atributos* (datos o variables): Información que posee cada objeto y que identifica su estado.  
   - *Métodos*(operaciones o funciones): Conjunto de instrucciones que definen el comportamiento del objeto. Reciben unos argumentos y devuelven un resultado.  
Sólo se puede acceder a los atributos de un objeto a través de sus métodos.  
  
Imaginemos que nuestro objeto es un exploit y que su plantilla es la siguiente:  
  
------------------------------------------  
|                        Dame el lenguaje  |  
|   Dame la vulnerabilidad                 |  
|            ---------------               |  
|           |   Autor       |              |  
|           |               |Dame el autor |  
|           |Vulnerabilidad |              |  
|           |               |              |  
|           |     Lenguaje  |              |  
|           |               |              |  
|            ---------------               |  
|                                          |  
|     Introduzco nuevo offset              |  
|                                          |  
 ------------------------------------------  
  
Este diagrama representa la estructura de un objeto. La parte interna del objeto no es accesible y contiene los atributos. La parte externa o interfaz del objeto es accesible y contiene los métodos. 

Citar

Para los que no conocen mucho sobre los exploits, les recomiendo que se den una vuelta por <http://foro.elhacker.net/index.php/board,32.0>  
  
Para poder comprender este ejemplo, sólo es necesario saber que un exploit es un código (codificado por un autor) que explota una vulnerabilidad para lograr ejecutar código arbitrario en el sistema comprometido. La codificación puede haberse hecho en el lenguaje que prefiera el autor: C, C++, Perl...  
Antes de lanzar un exploit contra un objetivo remoto, por ejemplo, debemos conocer su sistema operativo y a veces es necesario cambiar la dirección offset dentro del código del exploit, en función del sistema operativo y la versión de service pack que el objetivo tenga instalada, para ajustar el exploit a las características del sistema objetivo.

Bien, nuestro **objeto-exploit**tiene algunos atributos como el **autor** del exploit (= Lion), la **vulnerabilidad** que explota (= Serv-U FTPD 3.x/4.x "SITE CHMOD" Command Remote stack buffer overflow) y el **lenguaje** en que se ha codificado el exploit (= Lenguaje C). Si otro objeto quiere acceder a estos atributos, no lo puede hacer directamente, sino a través de los métodos del objeto-exploit. Es decir,   
    · si quiere saber el autor que codificó el exploit, llamará al método "dame el autor" del objeto-exploit y recibirá como resultado el tipo string con el nombre del autor = "Lion".   
    · si quiere introducir una nueva dirección offset, llamará al método "introduzco nuevo offset" y le pasará como argumento el tipo string con la dirección offset = "0x7ffa4a1b" a introducir para que el exploit nos funcione.  
Generalmente, a estos métodos q "obtienen" o "introducen" se les llama métodos *getters* y *setters*. De esta forma, los métodos del objeto-exploit pasarán a codificarse bajo los nombres**getAutor, getVulnerabilidad, getLenguaje, y setOffset.**  
  
Continuando con el ejemplo...   
    · el prototipo del método getAutor será: String getAutor ()  
String (antes del método), indica que la llamada al método devuelve un tipo String, como es una cadena de caracteres con el nombre del autor. Sin embargo, no hace falta pasarle ningún argumento al método, por lo que dejamos los paréntesis () vacíos.  
    · el prototipo del método setOffset será: void setOffset (String offset)  
void (antes del método), indica que la llamada al método no devuelve ningún tipo. Sin embargo, es necesario pasarle un argumento tipo String, como es la cadena de caracteres de la dirección offset para que, por ejemplo, el código del método implemente esa dirección offset y entonces lance el exploit ya ajustado a las características del sistema objetivo.  
  
Si este ejemplo del objeto-exploit te ha resultado demasiado confuso por los tecnicismos empleados, quizás te resulte más fácil comprender un objeto-reloj. Tiene los atributos Hora (= 19:00), Fecha (= 31/12/2004) y Marca (=Casio) y sus métodos podrían ser getHora (para pedir la hora que marca el reloj), getFecha (para pedir la fecha que marca el reloj), getMarca (para pedir la marca del reloj), setHora (para introducir una nueva hora en el reloj) y setFecha (para introducir una nueva fecha en el reloj). Jejejeje, mejor así?? ;)  
  
  
1.2) Concepto de Clase.   
Justo antes del dibujo del objeto-exploit, he remarcado que ese diagrama representaba su "plantilla". Es decir, imagina que tenemos una plantilla para crear muchos objetos-exploit, todos ellos con los mismos tipos de atributos y los mismos métodos, pero cada uno con su **identidad propia**. Otro ejemplo de objeto-exploit diferente podría contener la siguiente información en sus atributos:   
Autor = fiNis,   
Vulnerabilidad = Jordan Telnet Server Buffer Overflow,  
Lenguaje = C  
y tener los mismos métodos para acceder a estos atributos, pero como ves, la identidad es distinta.  
  
Las clases son una especie de plantilla para los objetos, es decir, si se piensa en una clase como un molde de galletas, los objetos que se crean a partir de esa clase son las galletas.  
Definimos formalmente *clase como la representación de la estructura (atributos) y comportamiento (métodos) de un objeto.*  
La creación de un objeto a partir de una clase se denomina "instanciación". Cualquier objeto que se instancie de una clase creará una copia de la definición de atributos de la clase y dispondrá de los métodos definidos en ella.  
Para crear objetos, se invoca al constructor de la clase, que es un método que se llama igual que la clase.  
  
Tipos de clases:  
- *Clase abstracta*: ningún objeto pertenece directamente a dicha clase.  
- *Clase concreta*: exiten objetos que pertenecen directamente a dicha clase.  
  
  
1.3) Atributos y Métodos de Objeto/Clase  
Antes hemos visto que un Objeto se compone de:  
   - Atributos (datos o variables): Información que posee cada objeto y que identifica su estado.  
   - Métodos (operaciones o funciones): Conjunto de instrucciones que definen el comportamiento del objeto. Reciben unos argumentos y devuelven un resultado.  
  
Y hemos dicho que la única manera de acceder a los atributos de los objetos, es a través de los métodos.   
*Los métodos son un conjunto de instrucciones al cual se le pueden pasar unos argumentos y/o devolver unos resultados.*  
Existen dos tipos de métodos:  
- *Métodos de instancia*: invocados en los objetos.  
- *Métodos de clase*: invocados en las clases. Estos atributos y métodos serán compartidos por todos los objetos creados a partir de una misma clase.  
  
Para comprender el concepto de atributo/método de instancia o de clase, volvamos al ejemplo del objeto-reloj:  
Imaginemos una clase-reloj que crea objetos-reloj, todos de la misma marca = Casio. Estos objetos-reloj implementan los atributos Hora, Fecha y Marca; y los métodos getHora, getFecha, getMarca, setHora y setFecha. Sin embargo, algunos de estos atributos y métodos son de clase y otros son de instancia:  
  
- *Atributos de instancia:*  
    · Hora  
    · Fecha  
- *Métodos de instancia:*  
    · getHora  
    · getFecha  
  
- *Atributos de clase:*  
    · Marca  
- *Métodos de clase:*  
    · getMarca  
  
Se puede distinguir las diferencias entre unos y otros de manera sencilla:   
  
    · Si le preguntamos a dos objetos (relojA y relojB) con getHora o getFecha, nos pueden contestar un valor diferente para cada objeto. Estos atributos y métodos serán denominados*atributos de instancia*y *métodos de instancia.*  
    · Si le preguntamos a dos objetos (relojA y relojB) con getMarca, ambos contestarán el mismo valor = Casio, ya que la clase-reloj se lo ha asignado forzósamente. Estos atributos y métodos serán denominados *atributos de clase*y *métodos de clase*.   
En este caso, por ejemplo, si existiera el método setMarca y modificaramos el atributo Marca de cualquier objeto a través de él, este atributo se modificaría para todos los objetos, pues estamos tratando con métodos de clase.  
  
CONSECUENCIAS  
**Cualquier objeto creado a partir de una clase contendrá la definición de los atributos de clase y dispondrá de los método de clase. Por tanto, cuando se modifica un atributo de clase a través de un objeto, se modifica ese atributo en todos los objetos creados a partir de esa clase.**  
  
  
*Especificaciones sobre los métodos*:  
Un objeto de clase no puede acceder a un método de instancia, porque no sabemos si el objeto al que intenta acceder ha sido ya creado. Sin embargo, el resto de accesos son posibles:  
    · método de clase - método de clase  
    · método de instancia - método de instancia  
    · método de instancia - método de clase  
Asimismo, un método de instancia de un objeto A no puede acceder directamente a los atributos (variables) de un objeto B, pero puede hacer referencia a un método de instancia del objeto B, y que sea este el que acceda a los atributos del objeto B.  
  
  
1.4) Modificadores de acceso.  
Alcance de los modificadores de acceso prefijos a atributos y métodos:  
  
- *public*: accesible desde cualquier lugar de la aplicación (por el resto de clases).  
- *private*: sólo accesible desde la clase a la que pertenece.  
- *protected*: disponible para la clase actual, clases del mismo paquete y subclases derivadas de esa clase.  
  
  
1.5) Paradigmas de la POO  
  
1.5.1) **Abstracción**  
*Consiste en abstraer conceptualmente los atributos y comportamiento (métodos) comunes a un determinado conjunto de objetos y almacenarlos en una clase.*  
   
1.5.2) **Encapsulamiento**  
*El encapsulamiento es el principio por el cual se ocultan los detalles de implementación al usuario*.  
Cada clase tiene dos partes:  
    · El *Interfaz* es la parte pública con la que interactúa el usuario (métodos públicos)  
    · La *Implementación* es el código que realiza todas las operaciones de la clase (métodos privados)   
   
1.5.3) **Herencia**  
*La herencia permite crear una clase nueva (subclase o clase derivada) que tenga el mismo comportamiento que otra (superclase o clase base) y además extienda o adapte ese comportamiento a unas necesidades específicas.*  
**La nueva subclase heredará los atributos y los métodos de la clase base, los cuales se añadirán a los definidos en la propia subclase.**  
  
Continuando con el ejemplo de la clase-exploit, observamos que por Internet circulan bastantes exploits que afectan a vulnerabilidades en servicios de Microsoft Windows. ¿Por qué no agrupar todo este conjunto de objetos-exploit bajo una nueva clase-winexploit?. Esta nueva clase-winexploit herederá los mismos atributos (Autor, Vulnerabilidad y Lenguaje) y los mismos métodos (getAutor, getVulnerabilidad, getLenguaje y setOffset) de la clase-exploit. A su vez, "extenderá su comportamiento" implementando otros atributos y otros métodos específicos como, por ejemplo, el atributo CodigoMS (referente al Codigo Microsoft Security Bulletin asignado a la vulnerabilidad) y el método getCodigoMS.  
La clase-winexploit quedaría así:  
  
------------------------------------------  
|                        getLenguaje       |  
|   getVulnerabilidad                      |  
|            ---------------               |  
|           |   Autor       |              |  
|           |               | getAutor     |  
|           |Vulnerabilidad |              |  
|           |               |              |  
|           |     Lenguaje  |              |  
|           | CodigoMS      |              |  
|            ---------------               |  
|                                          |  
| setOffset                                |  
|                          getCodigoMS     |  
 ------------------------------------------  
  
Un ejemplo de objeto-winexploit podría ser el siguiente:  
- Atributos:  
    (atributos heredados de la clase-exploit)  
    · Autor: Wirepair  
    · Vulnerabilidad: Windows Workstation Service WKSSVC.DLL Buffer Overflow  
    · Lenguaje: C  
      
    (nuevos atributos implementados por la clase-winexploit)  
    · CodigoMS: MS03-049  
  
- Metodos:  
    (métodos heredados de la clase-exploit)  
    · getAutor  
    · getVulnerabilidad  
    · getLenguaje  
    · setOffset  
  
    (nuevos métodos implementados por la clase-winexploit)  
    · getCodigoMS  
  
      
La sintaxis en Java para indicar que una nueva clase hereda de otra superclase es:

Código:

class winexploit extends exploit

*Especificaciones sobre la Herencia:*  
Además de "extender" el comportamiento implementando nuevos atributos y nuevos métodos, una clase heredada puede "adaptar" el comportamiento sobreescribiendo la funcionalidad de la clase base, esto es, sobreescribiendo la implementación del algún método heredado de la clase base para que al llamar a ese método desde la nueva clase heredada, en vez de ejecutar el código original del método heredado, se ejecute un nuevo código adaptado al comportamiento de la nueva clase. (Polimorfismo)  
  
1.5.4) **Polimorfismo**  
*El Polimorfismo es la respuesta distinta frente a una llamada a un método dependiendo de la naturaleza del objeto*.  
Consiste en definir métodos distintos, que comparten el mismo nombre, pero que se aplican a clases diferentes.   
Por ejemplo, un método llamado breathe puede responder de manera distinta dependiendo de quien lo invoque:

Código

**class** animal  
{  
    **public** **void** breathe()  
    {  
        [System](http://www.google.com/search?hl=en&q=allinurl%3Asystem+java.sun.com&btnI=I%27m%20Feeling%20Lucky).out.println("Respirar...");  
    }  
}  
   
**class** pez **extends** animal  
{  
    **public** **void** breathe()  
    {  
        [System](http://www.google.com/search?hl=en&q=allinurl%3Asystem+java.sun.com&btnI=I%27m%20Feeling%20Lucky).out.println("Burbujear...");  
    }  
}

Si se invoca el método breathe desde un objeto-perro, imprimirá en pantalla "Respirar...", pero si es invocado desde un objeto-trucha, imprimirá en pantalla "Burbujear...".  
  
  
Otra forma de utilizar polimorfismo es mediante la *sobrecarga de funciones*:

Código:

char convertir\_numero\_cadena(int valor, char cadena, int base)  
char convertir\_numero\_cadena(long valor, char cadena, int base)

En este caso, cuando se invoca la función, el compilador debe analizar los argumentos de la llamada y, en función de su tipo, deducir cual de las distintas versiones del método con ese nombre debe recibir el control.