INSTITUTE FOR SECURITY AND OPEN METHODOLOGIES

SPSMM

STANDARD DE PROGRAMACIÓN SEGURA

Created by Victor A. Rodriguez

CURRENT VERSION:	v0.5.1
NOTES:	Spanish
CHANGES:	-
DATE OF CURRENT VERSION:	May 2002
DATE OF ORIGINAL VERSION:	-

Any information contained within this document may not be modified or sold without the express consent of the author.
Copyright 2000-2003, Victor A. Rodriguez, the Institute for Security and Open Methodologies. All Rights Reserved, available for free dissemination under the Open Methodology License (OML).

STANDARD DE PROGRAMACIÓN SEGURA - Victor A. Rodriguez (Bit-Man)

Cuando se expone una interfaz (de programacíon, usuario, etc.) subuso puede ser subvertido (uso abusivo). Es este abuso el que debe ser evitado (seg'un en que lado uno se encuentre parado). Ya que planeamos realizar un standard de esta metodolog'ia desde el lado de la programacíon, y seguramente Ud. es un programador, arquitecto de sistemas, team leader u otra cosa por el estilo querr'a mantener la interfaz limpia, usable y sin componentes que puedan ser abusadas.

A medida que las aplicaciones de Internet comienzan a ser m'as y más complejas, y la generación de código comienza a ser cada vez más automatizada a través del uso de wizards, ambientes de programación, frameworks y otras herramientas por el estilo comenzamos a estar en franca desventaja. La técnicas para programación segura es algo que ha sido abordado desde muchas formas y por muchos lenguajes. Lo que se trata de hacer es una única metodología independiente de lenguajes de programación, ambientes de programación y herramientas de desarrollo.

La idea de convertirlo en un standard abierto nos da el inmediato beneficio que puede ser usado por cualquiera, pero esto también incluye a los *chicos malos*. Sin embargo, si se sigue una metodología seguramente los errores más comunes ya no aparecen. La contrapartida es que cuanto más programadores lean y entiendan este manual, mayor es la posibilidad de desarrollar aplicaiones en forma segura y ayudar a mejorar esta tecnología.

Agradecimientos

Quisiera agradecer a David A. Wheeler por escribir "Secure Programming for Linux and Unix HOWTO" [1] que tanto me inspiró para hacer este trabajo. También una agrdecimiento a quienes escribieron "SECPROG@SecurityFocus.com working document" [9], algo así como el "Primer Diccionario" sobre "Programación Segurá.

Debo también dar un agradecimiento muy especial a Pete Herzog, por toda la ayuda brindada en muchos de los aspectos de este trabajo.

Todos aquellos que contribuyeron a este manual en alguna forma son emncionados al principio de este documento. Cada quien recibe un reconocimiento por el tipo de contribución, y no por lo que ha contribuído. El uso de este ocultamiento es para prevenir la polarización al momento de la interpretación. Sólo en la traducción del documento se saben las personas que realizaron la contribución, ya que el traductor se convierte en el contacto del documento traducido que incluye, no sólo el mantener actualizado el documento, sino también las FAQs, agregados y comentarios hechos en el original, en Inglés, y traducido al lenguaje deseado.

Términos

- **DoS (Denial of Service):**Ataque cuyo porpósito es no permitir el uso del sistema. No afecta los datos almacenados en el sistema.
- **out-of-the-box:**instalación standard, utilizando los parámetros por defecto.

Copyright 2000-2003 Victor A. Rodriguez, ISECOM – The Institute for Security and Open Methodologies – www.isecom.org - www.osstmm.org .
ISECOM is the OSSTMM Professional Security Tester (OPST) and OSSTMM Professional Security Analyst (OPSA) certification authority.

- **super-user:** es un usuario especial (conocido como root o administrador) que no tiene restricciones de acceso dentro del sistema.

Audiencia

Este manual fue escrito para profesionales de la programación que quieren establecer lineamientos y/o procedimientos para la programación segura en cualquier proyecto de Tecnología de la Información (IT), independiente del lenguaje de programación, ambiente de ejecución y herramientas de desarrollo.

Este manual no examina la forma correcta de usar una herramienta de desarrollo en particular o cómo programar en un lenguaje dado.

Los desarrolladores encontrarán muy útil este manualen realizar mejores programas, más resistentes a ambientes hostiles tales como Internet.

Alcance

El objetivo es realizar un standard de metodología de programación segura que pueda ser usado en cualquier proceso, manual o automático, y permita alcanzar los requerimientos de seguridad para maximizar el uso y evitar su abuso. El resultado indirecto es la creación de una disciplina que pueda actuar como un punto central en todas las pruebas de seguridad independientemente del lenguaje de programación, ambiente de ejecución y herramientas de desarrollo.

Excepciones

Los casos no cubiertos, aún, en este trabajo son :

- Programas cuyo propósito es el manejo de la seguridad
- Programas que deben correr como super-user

Preparación para trabajar en un ambiente hostil

Esto no es tarea fácil, y fue hecho en diferentes maneras, cada una cubriendo su punto de vista, comenzando en diferentes formas, etc. Aquí estableceré las mías propias.

Si le damos una mirada a Secure Programming for Linux and Unix HOWTO [1] (sección 2.2, Security principles) la programacín segura puede ser interpretada como el cumplimiento de los siguientes objetivos: confidencialidad, integridad y disponibilidad. Esto es correcto y puede ser moldeado para cualquier ambiente en particular, pero es una muy buena respuesta genérica. El problema comienza en cómo alcanzar esto objetivos. Lo que se tratará de enfatizar no es el arreglo de los mismos, sino en una metodología open source que provea la base para el desarrollo de m'etodos y/o procedimientos.

El trabajo de un sistema es implementar requerimientos funcionales que, debidamente acordados, puedan ser probados. En una aproximación más detallada, la codificación se hace para cumplir con este objetivo pero en una escala atómica (a nivel de subrutina o procedimiento de programación). Lo que no es parte de los requerimientos funcionales es la disponibilidad del sistema, la que establece las condiciones de uso tales como en qué días será usado (laborables, feriados,

Copyright 2000-2003 Victor A. Rodriguez, ISECOM – The Institute for Security and Open Methodologies – www.isecom.org - www.osstmm.org .
ISECOM is the OSSTMM Professional Security Tester (OPST) and OSSTMM Professional Security Analyst (OPSA) certification authority.

etc.), ventana diario de uso (9:00 a 17:00 hs, etc.), m'aximo tiempo permitido sin acceso al sistema (2 horas mensuales, etc.) y toda una serie de requerimientos que definen el denominado Acuerdo de Nivel de Servicio (SLA o Service Level Agreement).

Es en este punto donde, ejecutando en ambientes hostiles como Internet, poder brindar los niveles requeridos del SLA (integridad), imposibilitar el robo de información (seguridad) o proveer la información a la persona correcta (confidencialidad) es de particular interés.

Una vez que conocemos sobre este y su impacto negativo, hay una gran preocupación en cóomo adoptar una aproximación sistemática para resolverlo. Para comenzar, veamos los factores que dan comienzo a esta problemática :

- **qué no hacer:** es muy difícil construir sobre lo que no debe hacerse. Protegerse de algo que no se sabe su forma exacta es una tarea muy dura (un muy buen artículo sobre este tema puede verse en el newsletter Crypto-gram de Noviembre de 1999 [2]). Es similar a la defensa contra los microbios, uno no puede protegerse a menos que se sepa de su existencia y de cómo derrocarlos.
- **complejidad:** esto da origen a uno de los mas viejos y difíciles problemas de la humanidad. Se llama comunicación, y según se agrega complejidad a un sistema este puede manejarse mejor si se divide en tareas ma;s pequeñas, se asignan a distintos equipos que deben comunicarse entre si para alcanzar el objetivo: cuanto m'as equipos hay mayor cantidad de comunicaciones deben establecerse y, como es sabido, convierte a la construcción del sistema en un problema de crecimiento exponencial (lo mejor s dar una mirada al título *software essential difficulties* de "No silver bullet" [3]). Esto favorece la existencia de errores ocultos que son cubiertos por el síndrome de "hay muchos lugares donde mirar".

Methodología

Como este trabajo es parte de "The Open-Source Security Testing Methodology Manual" (OSSTMM), la misma metodología será usada, estableciendo los Parámetros y Tareas orientadas a la Programación Segura, pero con una aproximación un tanto diferente. En OSSTMM se descubre que hay un error, y ahora queremos saber dónde se haya ubicado, cóomo arreglarlo y evitar su aparición en futuras codificaciones.

Esta sección trata de los dos primeros objetivos (descubrimiento y reparación) mientras que el trabajo de cómo evitar estos problemas es un tema a ser tratado en un trabajo a ser desarrollado (no incluido en este).

Datos entrantes

Debe notarse que las interfaces se basan en la implementación de alguna tecnología y que la explotación de los problemas de seguridad deben ser arreglados para obtener una comp[leta solución al problema. No es suficiente con arreglar la implementación de la interfaz o el programa bajo prueba.

Algo a tener en cuenta es que los dato de entrada no son sólo los adquiridos a través del usuario o de otros aplicaciones, sino también los obtenidos desde el sistema (fecha, variables de ambiente,

Copyright 2000-2003 Victor A. Rodriguez, ISECOM – The Institute for Security and Open Methodologies – www.isecom.org - www.osstmm.org .
ISECOM is the OSSTMM Professional Security Tester (OPST) and OSSTMM Professional Security Analyst (OPSA) certification authority.

etc.) y otro software que forman la base para el sistema. Resumiendo, cada porción de datos que alimenta a nuestro programa o sistema debe ser analizado.

Validación de patrones de datos de entrada

Deben probarse los datos de entrada contra patrones de aceptación y rechazo. Esto puede dividirse entre diferentes pantallas, scritps, URLs, etc. para manejarlo según los recursos (tiempo, costos) y la gente disponible. Se recomienda probar todas las interfaces disponibles (GUI, API, etc.) y no solamente los que son expuestos al usuario.

En este punto quien realiza las pruebas debe hacer foco en cómo elaborar los grupos de datos que no solamente no coincidan con los patrones establecidos, sino que trate de explotar algún elemento proveniente de la arquitectura en la que el sistema se ahay basado (sistema operativo, lenguaje de programación, base de datos, etc.)

Resultados esperados:

- Una lista patrones de datos aceptados y rechazados

Tareas a realizar:
Acetación/rechazo de datos:

- Verificar los datos aceptados que deben ser aceptados.
- Verificar los datos rechazados que deben ser aceptados (falso negativo).
- Verificar los datos aceptados que deben ser rechazados (falso positivo).
- Verificar los datos rechazados que deben ser rechazados.

Trabajo sobre el código:

- Identificar los segmentos de código que manejan cada dato entrante y su validación
- Arreglar el código

En lo referente al arreglo de código, y sólo para esta prueba, la clave es no sólo arreglar código para rechazar los datos nocivos, sino extrapolarlos en patrones genéricos. En este punto el método usado para filtrar y rechazarlos debe ser discutido. El método elegido depende de las limitaciones del ambiente escogido, como se explica en "Inggreso de Datos" en el Apéndice C, pero aún hay algo sobre el filtrado que debe mencionarse: el método de aceptación y rechazo.

Hay dos métodos básicos :

- **Patrón de aceptación**: los datos ingresados se prueban contra patrones de aceptación, haciendolos m'as selectivos según se descubren nuevas vulnerabilidades. Si ninguno de los patrones es satisfecho los datos son rechazados.
- **Patrón de rechazo**: los datos ingresados se prueban contra patrones de rechazo, agregando patrones a medida que se descubren nuevas vulnerabilidades. Si ninguno de los patrones es satisfecho los datos son aceptados.

Copyright 2000-2003 Victor A. Rodriguez, ISECOM – The Institute for Security and Open Methodologies – www.isecom.org - www.osstmm.org .
ISECOM is the OSSTMM Professional Security Tester (OPST) and OSSTMM Professional Security Analyst (OPSA) certification authority.

Si se presta atención cad método tiene sus pros y contras. El segundo es m'as facil para agregar nuevos patrones, pero requiere prubas adicionales para cada patrón de rechazo. El primero es m'as limpio y genérico. Por esto el método del **Patrón de aceptación** es el preferido, b'asicamente porque se trabaja en reducir los patrones de aceptación, siendo así un esquema más proactivo más fácil de mantener y menos propensa a errores.

Un ejemplo típico de los bugs encontrados con esta prueba pueden encontrarse en el "Apéndice A" baj el título "Compromiso Remote".

Validación de los límites de los datos de entrada

Una vez que los datos de entrada se prueban contra los patrones correspondientes, puede ser que los que se obtenga no sea tan bueno. Por ejemplo si un identificador de una cuenta bancaria es numérica, no todas las combinaciones ni todas las longitudes de cuenta son son válidas. Notar que estos límites no están relacionados con el significado en el mundo real, sino con las implicancias de seguridad en cuanto a la longitud de las variables. Dicho de otra forma, es cuando la implementación de la tecnología (lenguajes, sistemas operativos, etc.) se lleva a límites de stress que hacen desviarlos de su comportamiento normal (un forma común de probarlos es llamar a funciones con parámetros que estén en el orden de los kilobytes y que se supone no deben ser más de unos cuantos bytes).

resultados esperados:

- Comportamiento del sistema a entradas validadas pero no esperadas

Tareas a realizar:
Monitoreo de sistema:

- Uso CPU (porcentaje, tiempo, picos de uso, etc.)
- Uso deMemoria (real, virtual, cache, etc.)
- Uso I/O (espacio en disco, acceso en ráfagas, etc.)
- Uso de red (rechazo de paquetes, generación, etc.)

Trabajo sobre el código:

- Identificar los segmentos de código que manejan cada dato entrante y su validación
- Arreglar el código

En este punto, el análisis y codificación es una especie de arte, porque hay más importancia en el comportamiento del sistema que en los valores devueltos. Esto requiere un amplio conocimietno de la tecnología que se utiliza, principalmente de las manejos internos del lenguaje tales como manejo de memoria, generación de procesos y threads, bloqueos, IPC, etc.

Un ejemplo de los bugs descubiertos con este tipo de pruebas puede ser visto en el "Apéndice A" bajo el título "Bug de Format String". su efecto puede ser minimizado si las pruebas correspondientes sobre el tamaño y formato son realizadas.

Copyright 2000-2003 Victor A. Rodriguez, ISECOM – The Institute for Security and Open Methodologies – www.isecom.org - www.osstmm.org .
ISECOM is the OSSTMM Professional Security Tester (OPST) and OSSTMM Professional Security Analyst (OPSA) certification authority.

Procesos

una vez que nos protegemos del mundo exterior, debemos hacerlo de nuestro propia forma de codificación y prácticas internas.

Un análisis de cómo proteger el proceso es básicamente la protección de todos los r3ecursos usados por el sistema: memoria, disco, red, etc. para evitar su abuso por el mismo o por terceros.

Consumo de recursos

Cada programa hace uso de recursos (tiempo de CPu para cálculos, memoria para almacenamiento interno y externo, archivos para almacenamiento, etc.) para sus propósitos. Bajo ciertas condiciones es posible hacer que el programa comience un ciclo de consumo que termine en una caída del sistema (system thrashing) o una finalización abrupta (program dumping). Estos son los llamados negaciones de servisio (DoS - Denial of Service).

Es práctica normal el realizar las pruebas sólo con los datos provenientes de patrones aceptados, pero es una buena práctica hacerlo también con los provenientes de patrones rechazados y variaciones de estos (conocido como *prueba sucia* o dirty test).

Resultados esperados:

- Consumo de recursos y uso para cada patrón de entrada y estado (referido a la máquina de estados implementada).

Tareas a realizar:
Monitoreo del sistema: por cada clase de recurso (CPU, memoria, etc.)

- Conveniente adquisición de recursos
- Conveniente uso de recursos
- Conveniente liberación de recursos
- Análisis/monitoreo de abrazos mortales (deadlocks)

Trabajo sobre el código:

- Identificar los segmentos de código que manejan cada dato entrante y su validación
- Arreglar el código

Para evitar estas condiciones de DoS es buena práctica limitar la cantidad de recursos usados, y rechazar peticiones cuando se alcanza un imbral de consumo de recursos es alcanzado.

Un ejemplo de bugs descubiertos con este tipo de pruebas puede ser visto en el "Apéndice A" bajo el título "Retención de Recursos".

Prediciión del uso de recursos

Copyright 2000-2003 Victor A. Rodriguez, ISECOM – The Institute for Security and Open Methodologies – www.isecom.org - www.osstmm.org .
ISECOM is the OSSTMM Professional Security Tester (OPST) and OSSTMM Professional Security Analyst (OPSA) certification authority.

A veces el ser muy predictivo en el uso de recursos (lo que significa tener un patrón de uso) no es bueno, básicamente porque ayuda a predecir el siguiente movimiento a ser hecho por el sistema. Esto puede ser usado por un atacante para personificar al sistema (ofreciendo la misma respuesta que el sistema original, conociendo el siguiente nombre de archivo a ser usado y redireccionarlo para sobreescribir un archivo sensible, predecir identificadores de sesión, etc.)

Resultados esperados:

- Patrón del uso de recursos

Tareas a realizar:
Monitoreo del sistema: por cada clase de recurso (CPU, memoria, etc.)
Análisis de uso de recursos: Buscar patrones en el uso de recursos.

Trabajo sobre el código:

- Identificar los segmentos de código que manejan cada dato entrante y su validación
- Arreglar el código

Prevención de ataques

Cuando se procesan los datos obtenidos tendemos a creer que los datos disponibles son seguros, pero ... pasaron los datos por los controles de seguridad si el programa no es ejecutado bajo el ambiente especifico para el que fue programado ?? Que ocurre si un atacante encontró la forma de ejecutar el módulo de procesamiento pasando por alto estos controles ??

Algunos programas hacen pruebas adicionales tales como de funcionamiento en el ambiente indicado y modificación del código.

Resultados esperados:

- Respuesta detallada del sistema bajo condiciones anomales de funcionamiento.

Tareas a realizar:
Ejecución del sistema:

- Usar puntos de entrada al sistema distintos de los especificados
- Inyección y modificación del código
- Modificando las condiciones de funcinamiento

Trabajo sobre el código:

- Identificar los segmentos de código que manejan cada dato entrante y su validación
- Arreglar el código

La primera forma de reparación que se nos viene a la mente es la de incluir un código similar al de validación de datos de entrada, pero esta tarea está reservada a los módulos de verificación de

Copyright 2000-2003 Victor A. Rodriguez, ISECOM – The Institute for Security and Open Methodologies – www.isecom.org - www.osstmm.org .
ISECOM is the OSSTMM Professional Security Tester (OPST) and OSSTMM Professional Security Analyst (OPSA) certification authority.

datos, por lo que esta tarea no debe ser duplicada. La tarea a ser llevada a cabo es distinta: detección de condiciones de ejecución anormales y **no** la de detección de datos impropios.

Salida

Todos los datos de entrad están filtrados, se tomaron precauciones sobre el uso de recursos pero ... no es suficiente. Que tal el decirle a sus competidores la clave del éxito ?? A veces no es tán drástico, pero el darle algunas pistas es solo una cuestión d tiempo para obtener la información completa.

Algo debe tenerse en cuenta, y es que algunos errores pueden ser vistos como errores de proceso porque producen información sutil, producto de una salida no esperada (espúrea). Lo que se trata de mostrar es el pasaje de información en los cabales normales de salida, tales como tags de HTML ocultos, errores que pueden dar información sobre la estructura interna, y problemas por el estilo.

Niveles de autorización de los datos

Este concepto es muy claro en el uso militar. En forma resumida, se tienen categorias (ultra secreto, secreto, público, etc.) que marcan cada objeto (información, gente, etc.) y una persona puede acceder cualquier recurso marcado con el mismo o menor nivel que el suyo propio (por ejemplo, una persona con nivel "secreto" puede acceder recursos marcados como "secreto" y "público" pero no los marcados "ultra secretos"). Además de esto, un recurso tiene el mismo nivel que cualquiera de los objetos que contiene (por ejemplo, un documento que contiene datos marcados como "publico" y "secreto" es clasificado como "secreto"; si se agrega una parte "ultra secreta" se convierte en un documento "ultra secreto"). Lo mismo se aplica a los canales de comunicación, no se puede transmitir un documento "ultra secreto" a través de un canal "secreto" (el canal tiene menos mecanismos de protección que los que el documento puede aceptar).

Este mismo concepto debería ser aplicado aquí.

Expected results:

- Clearance levels for each output object (data, persons, channels, etc.)

tareas a realizar:

- Identificaión de los segmentos de código que realizan la salida de datos.
- Claisificación del sujeto al que está destinada la salida
- Clasificación de los datos de salida
- Clasificación de los datos a enviar por la salida

Trabajo sobre el código:

- Marcado de los datos con niveles de autorización
- Arreglar el código

Copyright 2000-2003 Victor A. Rodriguez, ISECOM – The Institute for Security and Open Methodologies – www.isecom.org - www.osstmm.org .
ISECOM is the OSSTMM Professional Security Tester (OPST) and OSSTMM Professional Security Analyst (OPSA) certification authority.

A veces se necesita brindar una buena cantidad de información, y no siempres una buena práctica, y la misma contiene los errores que nos da el sistema operativo tales como nombre de archivos, que pueden dar alguna pista a personas cuya intención es no brindarla.

Un claro ejemplo de esto son los web servers. Un server MS IIS (Internet Information Server) instalado out-of-the-box muestra un mensaje de error indicando que la página no existe y mostrando la supuesta ruta de acceso (file path) al archivo (en caso que este no exista) o la salida de un script en caso de error (que puede contener estructuras internas de datos o información de debugging cuya divulgación no es deseable). Su contrapartida (Apache) muestra un mensaje indicando que ha ocurrido un error, el administrador será notificado y la información más jugosa es almacenada en el archivo error_log.

Apéndice A - Errores más frecuentes

En esta sección las fallas serán mostradas, a veces en detalle y otras sólo una idea de las mismas, mayormente en el caso donde hay estudios o papers que las discuten.

No se pretende que esta sea una lista exhaustiva de fallas, sino las más frecuentes y representativas, para poder brindar una guía en cuanto a información, herramientas y resultados que pueden encontrarse.

Stack smashing

En esta versión en español se ha decidido mantener el nombre en inglés debido a que su traducción no es muy utilizada y hay más de una acepción del mismo. También se la conoce como "buffer overflow".

Será descripta muy brévemente, porque hay muy buenos artículos que las describen (ver [5] y [6]).

Cuando se definen dos variables en un lenguaje de alto nivel y están adyacentes en el código fuentes, hay una muy alta probabilidad que usen areas de memoria adyacentes. Por esto si se escribe una de estas y se sobrepasan sus límites, entonces se está escribiendo en el area de memoria perteneceiente a la otra. Si la primera tiene 30 bytes, cuando se accede al byte 31 es en realida el primer byte de la segunda variable.

No parece ser de mucho uso, pero hagamos una pequeña investigación. Cuando se usa un lenguaje de alto nivel mayormente usa subrutinas (también llamadas funciones, procedimientos, etc.) que usan sus propias variables locales. Es práctica común que los compiladores usan una porción de memoria, llamada stack, donde es almacenada la dirección de la próxima instrucción a ser ejecutada, llamada dirección de retorno, cuando la subrutina termina, y también se almacenan allí las variables locales. Lo que podemos hacer es utilizar esta localidad porque si accedemos las variables locales podemos ir más allá de sus límites y acceder la dirección de retorno. No se puede hacer magia para adivinar esta dirección, pero si podemos inyectar una nueva pieza de código y apuntar la direccón de retorno a este podemos hacer que al terminar ejecute nuestro código en lugar del original.

Esto puede evitarse usando algunos trucos tales como insertar algunas prubas de integridad para la dirección de retorno. Esto también puede ser evitado, pero hay menos probabilidades de ocurrencia de ataques. Un muy buen artículo sobre esto puede ser leído en [7].

Copyright 2000-2003 Victor A. Rodriguez, ISECOM – The Institute for Security and Open Methodologies – www.isecom.org - www.osstmm.org .
ISECOM is the OSSTMM Professional Security Tester (OPST) and OSSTMM Professional Security Analyst (OPSA) certification authority.

Format String bug

Hay funciones y funciones ... algunas más flexibles que otras, pero ... la flexibilidad tiene sus costos y limitaciones, y una de ellas es la seguridad. hay un conjunto de funciones en C (la familia de los printf) que uno de los parámetros es un string conteniendo el formato, y orden, en que los parámetros serán pasados a la función. Por ejemplo, la siguiente llamada a la función:

```
printf( "Hola, %s. Nos ha visitado %u veces", nombre, contador );
```

nos dice que los parámetros son tres (el string y las variables nombre y contador), y la información relativa a la cantidad y tipo de parámetros puede leerse del primer parámetro (también llamado "format string"): un string (%s) y un entero sin signo (%u).

Ahora que este "format string" se encuentra como constante en la aplicación, que hay acerca de darle elcontrol de este string al mundo exterior ?? Tomemos la siguiente porción de código :

```
scanf( "%s", *format_string );
 printf( format_string, name, counter );
```

donde format_string es provista por un usuario no verificado y/o no confiable (a través de la función scanf). Esto le podría permitir ver alguna información interna, muy útil para obtener datos y explotarlos posteriormente (tales como un "stack smashing" o "buffer overflow") a través de la manipulación del string, tales como el cambio de %s o %u por una forma más conveniente de ver la información de memoria (por ejemplo, usando un contador hexadecimal y declararlo en el format_string que es un puntero a un string o un char). Para una completa descripción, y ejemplos "in the wild", ver la referencia [10].

Compromiso remoto

Este ataque combina dos errores muy usuales: ausencia de filtrado (o filtrado defectuoso) y ejecución de un programa con privilegios excesivos.

A continuación se muestra un breve resúmen extraído del aviso de seguridad indicado en [11]:

"El motor de base de datos MS Jet (que permite acceder a bases Access) permite la inclusión el embeber expresiones de VBA en sus comandos, lo que permite ejecutar comandos de Windows NT. Esto, combinado con un error en IIS ejecutando comandos de ODBC como el usuario system_local permite a un atacante remoto poseer un control total del sistema. Otros web servers pueden ser afectados. Muchos motores de MS Jet son afectados, pero pueden no conducir a la elevación de privilegios"

Ba;sicamente se puede embeber comandos VBA (Visual Basic for Applications) dentro de una sentencia de SQL y dejar que ODBC los ejecute (filtrado inadecuado), tales como comandos del sistema a través de un intérprete de comandos (shell). Esto puede minimizarse si es ejecutado con un usuario que tiene privilegios mínimos para ejecutar esta tarea en una forma eficiente. Este no era el caso.

Copyright 2000-2003 Victor A. Rodriguez, ISECOM – The Institute for Security and Open Methodologies – www.isecom.org - www.osstmm.org .
ISECOM is the OSSTMM Professional Security Tester (OPST) and OSSTMM Professional Security Analyst (OPSA) certification authority.

No se harán más explicaicones. El aviso de seguridad mencionado es una obra maestra que merece ser leida en detalle.

Retención de recursos

Cuando se inicia una conexión de TCP la máquina de estados establece que deben seguirse tres pasos :

- Requerir la conexión al extremo remoto
- El extremo remoto responde con una aceptación (acknowledge), o la conexión se rechaza
- El extremo local responde con un acknowledge

Debido a la simplicidad de su mecanismo no posee un control para evitar la contención de recursos. Supongamos que cuando su computadora requiere una conexión a la computadora remota se cuelga, entonces el acknowledge será enviado pero la computadora local no lo recibirá, por lo que la computadora remota espera hasta que cierto time-out se cumpla y considera la conexión como cerrada. debido a que TCP/IP fue diseñado para trabajar sobre vínculos no confiables y de alta latencia, este esperará durante mucho tiempo hasta que considere la conexión cerrada (o fallida). Esto usa recursos en la computadora remota (mayormente CPU y memoria) que pueden ser reducidos a su mínima expresión si muchas conexiones permanecen en este estado.

Un reporte muy completo puede encontrarse en el CERT, en la referencia [12].

Apéndice B - Herramientas

Qué vuene a su mente cuando se dice herramientas. Si la respuesta es "programas para reforzar la programación segura" es más o menos correcto, solamente si, no significa sólo listas para usar. Se debe estar preparado para algún tipo de trabajo interno o la adaptación de algún que otro programa a sus necesidades.

Logging extensivo

Despues' de todos estos problemas a ser resueltos algo debe estar claro, y es que a pesar de todo el esfuerzo los programas no son perfectos, y las imperfecciones puede y, seguramente, van a aparecer. Para descubrir estos problemas deben usarse las herramientas apropiadas tales como debuggers y analizadores de dump, pero a veces no son los suficientemente útiles, particularmente cuando no se sabe por dónde comenzar la búsqueda.

Recuerdan el cuento de "Hansel y Gretel" ? Iban tirando migas de pan durante su camino, y así podían volver más tarde. Nostros usaremos el mismo truco, dejaremos una marca de cada pedazo de código que es ejecutado junto con los datos más importantes durante un ataque (o una simple falla), tal que se puedan rehacer los pasos seguidos.

Copyright 2000-2003 Victor A. Rodriguez, ISECOM – The Institute for Security and Open Methodologies – www.isecom.org - www.osstmm.org .
ISECOM is the OSSTMM Professional Security Tester (OPST) and OSSTMM Professional Security Analyst (OPSA) certification authority.

FIX ME: What should and shouldn't (DoS attacks, special chars, passwords, ...)

Listos para usar

Esto es una simple lista de las herramientas conocidas para asegurar la programación segura, siendo los predilectos los del tipo Open Source.

Nombre:	RATS (Rough Auditing Tool for Security)
Resúmen:	Es una herramienta de seguridad para C y C++ que hace una búsqueda sobre el código fuente, buscando llamadas a funciones potencialmente peligrosas.
Licencia:	Version 2 de la Licencia Pública GNU (GPL - GNU Public License).
Link:	http://www.securesw.com/rats/

Nombre:	Flawfinder
Resúmen:	examina el código fuente buscando vulnerabilidades en el código C o C++.
Licencia:	Licencia Pública GNU (GPL - GNU Public License).
Link:	http://www.dwheeler.com/flawfinder/

Nombre:	ITS4
Resúmen:	Es una herramienta simple que busca en el código fuente C o C++, en forma estadística, por potenciales vulnerabilidades de seguridad.
Licencia:	ITS4 NO-COMMERCIAL para el código fuente
Link:	http://www.cigital.com/its4

Nombre:	LCLint
Resúmen:	Prueba estadística de programas en C
Licencia:	Licencia Pública GNU (GPL - GNU Public License).
Link:	http://lclint.cs.virginia.edu/

Nombre:	StackGuard
Resúmen:	StackGuard es un compilador que genera programas mas' resistentes a los ataques del tipo "stack smashing" o "buffer overflow".
Licencia:	StackGuard es Free Software: es una mejora a GCC y se distribuye bajo licencia GPL enforma def uentes y binarios.
Link:	http://www.immunix.org/stackguard.html

Nombre:	FormatGuard
Resúmen:	FormatGuard protege programas C y C++ contra el "format bug", mayormente usado en la familia de funciones printf.
Licencia:	FormatGuard es Free Software: es una mejora a GCC y se distribuye bajo licencia GPL enforma def uentes y binarios.
Link:	http://www.immunix.org/formatguard.html

Nombre:	RSX
Resúmen:	RSX es un extensor del espacio de direcciones en tiempo de

Copyright 2000-2003 Victor A. Rodriguez, ISECOM – The Institute for Security and Open Methodologies – www.isecom.org - www.osstmm.org .
ISECOM is the OSSTMM Professional Security Tester (OPST) and OSSTMM Professional Security Analyst (OPSA) certification authority.

	ejecución (Runtime addressSpace eXtender) que provee reubicación de código en tiempo de ejecución para binarios Linux, que implementa un stack no ejecutable y areas de heap pequeñas y grandes. ataca el problema de "buffer overflow" previeniendo que se ejecute código en las areas reubicadas (definidas de solo lectura).
Licencia:	Licencia propietaria que incluye los fuentes
Link:	http://freshmeat.net/projects/rsx

Nombre:	PageExec
Resúmen:	Implementación de páginas no ejecutables para procesadores IA-32
Licencia:	???
Link:	http://pageexec.virtualave.net/

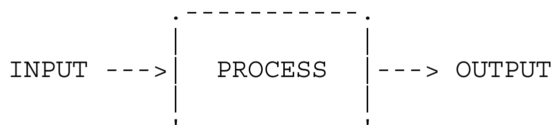
Nombre:	Libsafe
Resúmen:	Permite evitar los "buffer overflow" y "format string", empaquetado en forma de biblioteca (library) que intercepta las llamadas conocidas como vulnerables.
Licencia:	Libsafe version 2.0 (código fuente) es licenciado pajo la licencia GNU Lesser.
Link:	http://www.avayalabs.com/project/libsafe/index.html

Apéndice C - Fallas

Una vez que hemos navegado a la vieja usanza, cerca de la cosat, necesitamos trazar un mapa más detallado. Es mi preferencia personal ir de lo genérico hacia lo particular, así que comencemos.

Datos entrantes

Comencemos con lo básico sobre programas o manipulación de la información. El esuqema típico es obtener datos, procesarla (tranformarla) y ofrecer los resultados: entrada-proceso-salida.



No es gran cosa, pero para evitar funcionamientos incorrectos y efectos colaterales (algunos relacionados con las dependencias del sistema) deben ser filtrados para obtener los datos apropiados.

Supongamos que lidiamos con un sistema bancario donde debe ingresarse un número de cuenta y que debe obtenerse el balance de la cuenta, entonces la entrada (número de cuenta) debe tener un formato determinado (digamos tres

Copyright 2000-2003 Victor A. Rodriguez, ISECOM – The Institute for Security and Open Methodologies – www.isecom.org - www.osstmm.org .
ISECOM is the OSSTMM Professional Security Tester (OPST) and OSSTMM Professional Security Analyst (OPSA) certification authority.

números, cinco letras y ocho números), los que deben ser validados (probados contra una plantilla) y revisados (rechazarlo si el formato no es el apropiado o si la cuenat no existe).

En este ejemplo las opciones del formato de la cuenta son realmente simples, solo no puede darse el balance de una cuenta si esta no existe, pero hay problemas más sutiles (que se mostrarán más adelante) que no son del tipo funcional. Puede tomarse esta regla: todo lo que no sea usado debe ser evitado.

Hay muchas formas de hacerlo, y cada una depende de los objetivos de cada sistema:

- **Finalización anormal:** esta es la más radical; en caso de detección de una dato no válido el sistema interpreta que está bajo ataque (alguien buscando formatos de datos impropios) y finaliza.

- **Rechazo:** a mitad de camino, cuando un dato no válido es detectado se pregunta nuevamente por el mismo. esto tiene una ventaja para una atacante, y es que puede preguntarse una cantidad ilimitada de veces hasta que un dato válido es detectado.

- **Utilizable:** las partes impropias son quitadas y solo lo que coincide con el formato apropiado es usado. Tipicamente se usa para datos de longitud variable tales como direcciones de e-mail, URLs y direcciones postales.

Un lenguaje de programación que tiene capacidades para la programación segura es Perl. Puede ser ejecutado utilizando el "tainted mode" donde, esencialmente, cada dato obtenido desde el mundo exterior es marcado como "sucio" y no puede ser usado hasta que sea limpiado. Normalmente la máquina lavadora para estos "datos sucios" está hecha de filtros, y si estos datos son usados sin lavar Perl interrumpe su ejecución (para una explicación más profunda ver la referencia [4]).

Procesamiento (La caja de Pandora)

A veces se tiene mucho cuidado sobre los datos obtenidos pero, odio decirlo, no es suficiente. Porque no somos perfectos y cometemos errores debemos proteger al programa, y el sistema, de nosotros. Recordemos que en cualquier programa hay muchos factores a tener en cuenta (es tan cierto que en el análisis de confiabilidad de software el comportamiento de los programas se estudian en forma estadística) y no todos ellos pueden ser tomados en cuenta a la vez, por lo que la posibilidad de introducir errores es realmente alta (esta probabilidad decrece a medida que el programa es probado y arreglado con el correr del tiempo).

Copyright 2000-2003 Victor A. Rodriguez, ISECOM – The Institute for Security and Open Methodologies – www.isecom.org - www.osstmm.org .
ISECOM is the OSSTMM Professional Security Tester (OPST) and OSSTMM Professional Security Analyst (OPSA) certification authority.

No es fuera de lo común que en todas las empresas e softwarer bien establecidas el uso de ambientes de prueba, ejecución, etc. queb simulan las condiciones de campo (ejecucuión en su destino final) y ejecutándolo en una ambiente protegido, donde este falle, arroje errores, o lo que se le ocurra. Lo mismo ocurre en su destino final, es ejecutado con mucha precacución hasta que se ejecuta como un proceso productivo, siendo su monitoreo un proceso crítico en las primeras semanas.

La mayoría de los lenguajes modernos de programación como C++, Java, Perl y Phyton contienen mecanismos de protección. Basicamente se ejecutan partes del código en un ambiente protegido, y se atrapa cualquie error (o excepción) que pueda ser generado por el programa.

Por ejemplo, se puede rodear al código con un bloque que lo aisle de su ambiente en caso de falla, y devuelva el control a una porción de código que maneje este error:

```
try() {
    Code_under_suspicion();
} catch (Exception e) {
    System.println "Code_under_suspicion(),
                Exception generated : " + e;
}
```

Ahora supongamos que estamos desarrollando una serie de prográmas de uso crítico, como para medicina, aeronáutica, etc. donde deben continuar corriendo aún en condiciones de falla. Imaginemos que durante su vuelo, el programa que sigue la ruta hacia el aeropuerto envía el siguiente mensaje a la consola :

"Excepción en 0x3F745DE9. El sistema está siendo bajado."

Estos programas están hechos con estas situaciones en mente, y deben proseguir su ejecución aún bajo condiciones especiales (digamos que se aisla el código qué causó el problema, realiza algún tipo de prueba sobre el hardware y software, etc.)

Para leer una muy buena revisión sobre excepciones aplicado a Java, ver la referencia [8].

A veces no es una buena opción hacerlo de esta forma, pero el colocar algunas vallas de contención alrededor del programa suele ser un poco más fácil: no se requiere programación extra, ni recompilación para su activación. Para este casoe pueden usarse una serie de herramientas standard tales como dump analyzers (análisis post-mortem) o debuggers that envuelven al programa, y cuando se produce un error este es atrapado y manejado por el debugger. Estos pueden imprimir la situación del stack, contenidos de variables, u ofrecer al programador/operador un intérprete de comandos (shell) que permita la inspección del programa, su reinicio, dumping, etc.

Salida de datos

Copyright 2000-2003 Victor A. Rodriguez, ISECOM – The Institute for Security and Open Methodologies – www.isecom.org - www.osstmm.org .
ISECOM is the OSSTMM Professional Security Tester (OPST) and OSSTMM Professional Security Analyst (OPSA) certification authority.

Y hablando de efectos laterales, tiene en mente que su salida es la fuente de entrada de otro proceso ? O aún peor ... qué hay acerca de la basura que se genera ?

Comencemos por el principio. El mismo problema que tenemos con nuestros datos de entrada se transfiere al proceso que manejará nuestros datos de salida y los procesará. Este proceso no manejará datos en crudo porque nuestro programa/sistema actuará como filtro, pero asimismo puede generar problemas (no a porpósito, por supuesto) que pueden dañar un buen trabajo.

Tomemos un ejemplo donde se procesan una serie de datos y, como parte del proceso de salida, se genera un archivo. Todo parece estar correcto, todo está en las especificaciones y estamos siguiendo las reglas del juego. Seguro ??. Si, por ejemplo, generamos un archivo con un nombre fijo, digamos output.data, sencillamente :

```
filehandle = open( "output.data", w );
write( filehandle, data );
close( filehandle );
```

Seguramente se habrán agregado algunas precauciones sobre la existencia del archivo, preguntar al operador si el archivo puede ser borrado en caso que ya exista, hacer un backup, y todo ese tipo de cosas, pero ... qué ocurre si la infraestructura en la que nos basamos fue comprometida y no estamos escribiendo el archivo que realmente pensamos ?? Hay un "acceso limpio" al archivo ??

Además, como seguramente sabrá, una fuente importante de información son el análisis y recolección de los contenedores de basura (después de todo por esto existen los trituradores de papel). El ejemplo más claro de esto es la respuesta a la pregunta "dónde envía usted sus mensajes de advertencia y error ?". Seguramente habrá notado que en algunos web servers cuando ocurre un error, tales como una página no encontrada o una falla en un CGI, el error se envía al browser, y mensajes como el siguiente aparecen :

> The requested CGI program in '/home/httpd/cgi-bin/script' failed to execute. The next lines contain the error:
>
> Couldn't connect to database 'Customers' in server 'internal_db'.
> Send a message to hostmaster@this_domain.com

Qué fuente de información excelente !!! Esto DEBERíA redirigirse al log de errores o hacia algún otro lado donde pueda ser leído por el administrador/operador. Después de todo esto no tiene sentido para la mayoría de los usuarios y esa una muy buena fuente de información para los crackers. No lo cree así ?? La parte de triste de la historia es que la mayoría de los web servers están configurados en forma insegura out-of-the-box.

Copyright 2000-2003 Victor A. Rodriguez, ISECOM – The Institute for Security and Open Methodologies – www.isecom.org - www.osstmm.org .
ISECOM is the OSSTMM Professional Security Tester (OPST) and OSSTMM Professional Security Analyst (OPSA) certification authority.

Apéndice D - Links

[1] **Title:** Secure Programming for Linux and Unix HOWTO
Author: David A. Wheeler
Location: http://www.dwheeler.com/secure-programs

[2] **Title:** Why Computers are Insecure
Author: Bruce Shneier
Location: http://www.counterpane.com/crypto-gram-9911.html# WhyComputersareInsecure

[3] **Title:** No silver bullet
Author: Frederick P. Brooks, Jr.
Location: http://www.undergrad.math.uwaterloo.ca/~cs212/resource/Articles/SilverBullet.html

[4] **Title:** Perl security
Location: http://www.perl.com/pub/doc/manual/html/pod/perlsec.html

[5] **Title:** Smashing The Stack For Fun And Profit
Author: Aleph One
Location: http://www.phrack.com/search.phtml?view&article=p49-14

[6] **Title:** Avoiding security holes when developing an application
Author: Frederic Raynal, Christophe Blaess, Christophe Grenier
Location: http://www.linuxfocus.org/English/March2001/article183.meta.shtml

[7] **Title:** Bypassing StackGuard and StackShield
Author: Bulba and Kil3r
Location: http://www.phrack.com/search.phtml?view&article=p56-5

[8] **Title:** What's an Exception and Why Do I Care?
Author: Sun Microsystems
Location: http://java.sun.com/docs/books/tutorial/essential/exceptions/definition.html

[9] **Title:** SECPROG@SecurityFocus.com working document
Author: Secure Programming list contributors
Location: http://www.securityfocus.com/forums/secprog/secure-programming.html

[10] **Title:** Analysis of format string bugs
Author: Andreas Thuemmel
Location: http://www.securityfocus.com:80/data/library/format-bug-analysis.pdf

[11] **Title:** NT ODBC Remote Compromise
Author: Matthew Astley & Rain Forest Puppy
Location: http://www.securityfocus.com/archive/1/13882

[12] **Title:** CERT Advisory CA-1996-21 TCP SYN Flooding and IP Spoofing Attacks
Author: CERT
Location: http://www.cert.org/advisories/CA-1996-21.html

Copyright 2000-2003 Victor A. Rodriguez, ISECOM – The Institute for Security and Open Methodologies – www.isecom.org - www.osstmm.org .
ISECOM is the OSSTMM Professional Security Tester (OPST) and OSSTMM Professional Security Analyst (OPSA) certification authority.

Open Methodology License (OML)

Copyright (C) 2000-2003 Institute for Security and Open Methodologies (ISECOM).

PREAMBLE

A methodology is a tool that details WHO, WHAT, WHICH, and WHEN. A methodology is intellectual capital that is often protected strongly by commercial institutions. Open methodologies are community activities which bring all ideas into one documented piece of intellectual property
which is freely available to everyone.

With respect the GNU General Public License (GPL), this license is similar with the exception for the right for software developers to include the open methodologies which are under this license in commercial software. This makes this license incompatible with the GPL.

The main concern this license covers for open methodology developers is that they will receive proper credit for contribution and development as well as reserving the right to allow only free publication and distribution where the open methodology is not used in any commercially printed material of which any monies are derived from whether in publication or distribution.
Special considerations to the Free Software Foundation and the GNU General Public License for legal concepts and wording.

TERMS AND CONDITIONS

1. The license applies to any methodology or other intellectual tool (i.e. matrix, checklist, etc.) which contains a notice placed by the copyright holder saying it is protected under the terms of this Open Methodology License.

2. The Methodology refers to any such methodology or intellectual tool or any such work based on the Methodology. A "work based on the Methodology" means either the Methodology or any derivative work by copyright law which applies to a work containing the Methodology or a portion of it, either verbatim or with modifications and/or translated
into another language.

3. All persons may copy and distribute verbatim copies of the Methodology as are received, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and creator or creators of the Methodology; keep intact all the notices that refer to this License and to the absence of any warranty; give any other recipients of the Methodology a copy of this License along with the Methodology, and the location as to where they can receive an original copy of the Methodology from the copyright holder.

4. No persons may sell this Methodology, charge for the distribution of this Methodology, or any medium of which this Methodology is apart of without explicit consent from the copyright holder.

5. All persons may include this Methodology in part or in whole in commercial service offerings, private or internal (non-commercial) use, or for educational purposes without explicit consent from the copyright holder providing the service offerings or personal or internal use comply to points 3 and 4 of this License.

6. No persons may modify or change this Methodology for republication without explicit consent from the copyright holder.

7. All persons may utilize the Methodology or any portion of it to create or enhance commercial or free software, and copy and distribute such software under any terms, provided that they also meet all of these
conditions:

Copyright 2000-2003 Victor A. Rodriguez, ISECOM – The Institute for Security and Open Methodologies – www.isecom.org - www.osstmm.org .
ISECOM is the OSSTMM Professional Security Tester (OPST) and OSSTMM Professional Security Analyst (OPSA) certification authority.

a) Points 3, 4, 5, and 6 of this License are strictly adhered to.

b) Any reduction to or incomplete usage of the Methodology in the software must strictly and explicitly state what parts of the Methodology were utilized in the software and which parts were not.

c) When the software is run, all software using the Methodology must either cause the software, when started running, to print or display an announcement of use of the Methodology including an appropriate copyright notice and a notice of warranty how to view a copy of this License or make clear provisions in another form such as in documentation or delivered open source code.

8. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on any person (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If said person cannot satisfy simultaneously his obligations under this License and any other pertinent obligations, then as a consequence said person may not use, copy, modify, or distribute the Methodology at all. If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

9. If the distribution and/or use of the Methodology is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

10. The Institute for Security and Open Methodologies may publish revised and/or new versions of the Open Methodology License. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.
NO WARRANTY

11. BECAUSE THE METHODOLOGY IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE METHODOLOGY, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE METHODOLOGY "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE IN USE OF THE METHODOLOGY IS WITH YOU. SHOULD THE METHODOLOGY PROVE INCOMPLETE OR INCOMPATIBLE YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY USE AND/OR REDISTRIBUTE THE METHODOLOGY UNMODIFIED AS PERMITTED HEREIN, BE LIABLE TO ANY PERSONS FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE METHODOLOGY (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY ANY PESONS OR THIRD PARTIES OR A FAILURE OF THE METHODOLOGY TO OPERATE WITH ANY OTHER METHODOLOGIES), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

Copyright 2000-2003 Victor A. Rodriguez, ISECOM – The Institute for Security and Open Methodologies – www.isecom.org - www.osstmm.org .
ISECOM is the OSSTMM Professional Security Tester (OPST) and OSSTMM Professional Security Analyst (OPSA) certification authority.