



ADVANCED VULNERABILITY ASSESSMENT

Andres Tarascó Acuña

SIA

atarasco@sia.es

RESUMEN

A lo largo de esta charla iremos examinando cuales son las principales desventajas de las metodologías comúnmente utilizadas durante una prueba de intrusión. Entre estos problemas se encuentra la falta casi total de comunicación entre los diferentes módulos o herramientas utilizados para realizar estas pruebas. Esto provoca, en mayor o menor medida, la necesidad por parte del auditor de volver a repetir fases de “*Information Gathering*” y, aumenta por ello la duplicidad de la información aumentando el riesgo de contar con un numero mayor de “*Falsos Positivos*”.

Palabras Clave: Metodología Análisis de Seguridad, Knowledge Base, API Proxy.

1. INTRODUCCIÓN:

Es pues necesario definir un modelo de datos común que pueda ser utilizado por la mayoría de las herramientas de auditoria. Este modelo nos permitiría exportar y compartir los resultados minimizando el tiempo de análisis de un sistema.

Este análisis, de prolongarse mucho en el tiempo, podría estar impactando en el funcionamiento normal de la plataforma, por superarse la ventana horaria necesaria para realizar pruebas de seguridad sobre un sistema y, en el peor de los casos, pudiendo quedar incompleto.

Otro de los objetivos que deberíamos poder cubrir es verificar de forma automática y de la forma más sencilla posible la viabilidad de explotación de los agujeros de seguridad encontrados en los sistemas; Dejando de lado al consultor en tareas como puede ser la revisión manual de versiones del sistema operativo y parches de seguridad para ajustar *offsets* en *exploits* o pruebas de fuerza bruta.

1.1 DUPLICIDAD DE INFORMACIÓN.

La utilización de diferentes herramientas de análisis de seguridad entre las que se encuentra *Nessus* permite exportar resultados en XML. Son precisamente estos resultados los pueden ser fácilmente analizados para incorporar toda su información en un modelo de base de datos sobre el que podamos realizar consultas del estado de cada sistema.

Al no existir un modelo de datos común entre diferentes herramientas, esta información no podrá ser compartida por ninguna otra herramienta provocando por ello que las pruebas tengan que ser repetidas. Estas tareas repetitivas pueden provocar la alerta de elementos IDS e IPS situados en el segmento de red analizado y dificultar nuestro trabajo.

Queda claro que es necesaria la reutilización de la información de forma eficiente. Una forma de conseguir esto es mediante un desarrollo de una herramienta de análisis de seguridad que permita integrar las ventajas de las soluciones de seguridad existentes en el mercado.

2. METODOLOGÍA DE ANÁLISIS:

Como primera aproximación a la problemática de cómo compartir información entre aplicaciones nos fijamos en como implementa Nessus su Knowledge Base. Mediante el uso de una serie de Apis, Nessus permite a cada plugin reportar en tiempo de ejecución los resultados de sus análisis. Estos resultados pueden ser consultados para cada uno de los plugins de forma rápida y eficaz mediante el acceso a una tabla Hash.

```
struct kb_item {
    char *name;
    BOOL type;
    union {
        char * v_str;
        int v_int;
    } value;
    struct kb_item *next;
} **kb;
```

La incorporación de una Knowledge Base compatible con nessus en el desarrollo de una nueva herramienta de análisis intrusivo de seguridad permitiría, como se comenta anteriormente, verificar de forma automática la viabilidad de explotación de los agujeros de seguridad en una plataforma.

Mediante la carga de los resultados de una auditoria con Nessus, podemos utilizar la información para evitar los falsos positivos y ajustar, las especificaciones de nuestros exploits a la plataforma de destino.

2.1 MODULARIZACIÓN DE LA HERRAMIENTA DE ANÁLISIS.

La mejor opción para conseguir que una herramienta de estas características pueda ser actualizada periódicamente es contar con un diseño modular. Estos módulos se dividen en diferentes categorías: En primer lugar una Gestión de los plugins de los que se compone la herramienta; En segundo lugar gestión de sesiones, de la que depende la administración de los plugins, manejo de una Knowledge Base reactiva así como la E/S de datos en la interfaz que nos permite el acceso a los sistemas; Por último, la capa de Red, que será la encargada de asegurar el mejor rendimiento para el intercambio de información con los sistemas analizados.

Todas estas capas conforman un Framework muy poderoso para el desarrollo de exploits y de nuevos plugins de auditoría.

2.1.1 PROCEDIMIENTO DE CARGA DE PLUGINS.

El funcionamiento interno del escáner esta basado en la carga dinámica de módulos. Todo modulo, nombre con el que nos referiremos a las librerías cargas en la herramienta, es categorizado según su actividad (portscanning, Information Gathering y attack) y es cargado en una lista dinámica en memoria inicializando y cargando las funciones exportadas por estas librerías

```
typedef struct plugin {  
    ATAQUE *ataque;  
    char DllPath[256];  
    HANDLE libreria;  
    LANZAATAQUE LanzaAtaque;  
    GIVEMEINFORMATION  
    GiveMeInformation;  
    } PLUGIN;
```

Durante la ejecución de la aplicación, se generará, dependiendo de los requisitos de los plugins (consultados en la Knowledge base), una lista de puertos a analizar junto con aquellos puertos predefinidos por los propios módulos de portscanning.

Está primera fase de análisis, al igual que los resultados de otros muchos plugins podrán ser obviados en el caso de que exista una Knowledge Base que haya sido cargada previamente. En caso de no contar con la Kb de Nessus, nuestros plugins deben de ser capaces de realizar las tareas de information Gathering de forma independiente.

2.1.2 FUNCIONAMIENTO INTERNO DE LOS MÓDULOS.

Para asegurar que la comunicación entre el motor de la aplicación y los módulos sea bidireccional y, para garantizar el acceso a todas las funciones de Networking, cada módulo y el propio Engine exportan una serie de funciones que serán llamadas por la herramienta en su ejecución.

Cada modulo (exploit/info Gathering) consiste en una librería Dll que exporta una serie de funciones llamadas el proceso de análisis

Algunas de estas funciones son:

-**DllMain**: punto de entrada que tiene como objetivo inicializar el módulo y cargar las funciones exportadas por el scanner. Estas funciones serán utilizadas para acceder al resto de los módulos de Networking, gestión de sesiones y Kb.

-**GetVersion**: Utilizado para verificar la compatibilidad de cada uno de los módulos.

- **GiveMeInformation**: Devuelve una estructura que contiene la información de puertos y Servicios necesarios para el lanzamiento del plugin Así como el resultado del mismo (Informativo, shell)

- **LaunchAttack**: Proporciona al módulo toda la información disponible del sistema.

```

DLL_EXP ATAQUE* WINAPI GiveMeInformation(int totaldlls) {
    Char descripcion[]="MS05-029 - Plug and Play";
    ATAQUE *ataque=InitAtaque(descripcion,NULL,BINDSHELL);
    AddAttackPort(ataque,445,TCP,0,SERVICE_UNKNOWN);
    return(ataque);
}

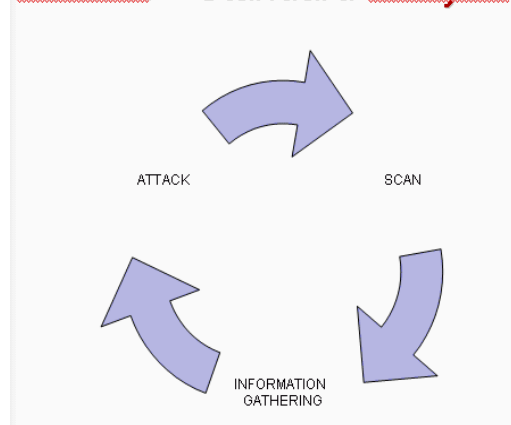
```

2.1.3 MANEJO DE LA KB EN ATAQUE.

Para minimizar el impacto sobre la plataforma así como el tiempo de análisis, se intenta emular desde el principio la metodología de un test de intrusión utilizando para ello una Knowledge Base reactiva. Ante cada resultado insertado en la Knowledge Base se verificara, por orden de prioridad, los requisitos de cada uno de los modulos cargados al inicio. Cada uno de ellos devolverá una nueva información que será a su vez utilizada para retroalimentar nuevos modulos.

Cada vez que se encuentra una concordancia entre información de la KB y la devuelta por la inicialización de los módulos se llamara a LaunchAttack() enviando como parámetros la Knowledge Base de esa dirección IP. La utilización de esa información permitirá al exploit ajustar offsets para que el desbordamiento de buffer se ejecute con éxito. Entre esta información debería encontrarse el sistema operativo, La versión del Service Pack, la presencia o no de un Firewall (por ejemplo mediante peticiones PORT en un FTP), ...

Pentest – Standard analysis



2.1.4 CAPA DE ATAQUE.

Las facilidades ofrecidas a los plúgins para la generación de ataques están integradas en parte en la capa de Networking.

Esta capa ofrece facilidad de generación automática de shellcodes al igual que realizan proyectos como Metasploit. Además, integra una capa de red con funciones encargadas del manejo de proxys para tunelizar las conexiones a otras redes.

El resultado del uso de estas Apis será, en último lugar, la correcta ejecución del exploit en el sistema remoto y el acceso al mismo mediante una shell manejada internamente por el programa (Modulo de consola).

2.1.5 CAPA DE E/S DE DATOS.

El acceso concurrente a múltiples sistemas es posible mediante la emulación de *Ttys* virtuales. Cada *tty* virtual, es accedida mediante la pulsación de una tecla y es detectada mediante la integración de un *keylogger* en la aplicación. Simulando la ejecución de un *screen* en un entorno Linux.

Cada una de estas terminales virtuales garantiza el acceso a los sistemas analizados.

Este parcheado dinámico de las funciones de networking permitirá a FIVS el uso de estos hosts como proxys para saltar a nuevas redes y la ejecución de herramientas como *wget* o nuestro *exploit/backdoor* favorito en el sistema remoto todo esto con la ventaja de que la ejecución es un nuevo *thread* de la aplicación atacada por lo que no será visible ninguna nueva aplicación en el sistema y dificultando las tareas de análisis forense.

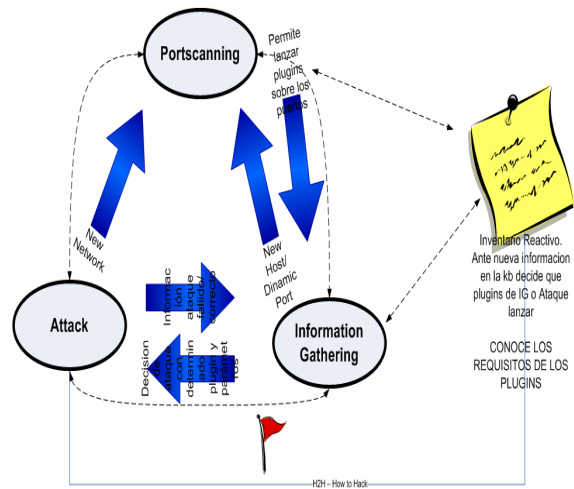
2.1.6 ENCAPSULACIÓN DE CONEXIONES.

Básicamente el principal problema durante un test de intrusión es que, una vez accedido al sistema remoto a través de una shell ,es necesario poder conectar contra nuestro equipo Base y transferir nuestras herramientas para poder continuar con la intrusión. Dependiendo de los privilegios de los que dispongamos, aplicaciones como *vbs*, *tftp*, *ftp*, creación de paginas *ASP*, *netbios*.. No estarán disponibles para facilitarnos nuestra tarea.

La integración de un *API Proxy* en la capa de generación de *Shellcodes* y en la de *Networking* nos permitirá encapsular estas llamadas a través de un *Proxy* de forma totalmente transparente para cada uno de los módulos.

Un pequeño *Lóader* incluido en esta herramienta nos permitirá ejecutar remotamente cualquier aplicación en el sistema remoto incluyendo la ansiada *Shell* remota.

Todas estas ventajas nos permiten reescribir el grafico de una metodología de análisis clásico añadiendo funciones nunca vistas hasta ahora y desarrollando una metodología *H2H How To Hack* en la que el tiempo necesario para ganar acceso a cada host es mínimo.



2.3 CONCLUSIONES.

El uso de esta metodología permitirá maximizar la automatización de tareas de análisis de seguridad consiguiendo unos resultados mucho más correctos.