El análisis estático hace morder el polvo a los hackers

Por Nikola Valerjev de Green Hills Software

www.ghs.com

Nikola Valerjev es Director de Ingeniería de Green Hills Software y Master of Engineering in Computer Science por la Cornell University.

Fundada en 1982, Green Hills Software, Inc. es líder en tecnología de sistemas operativos de tiempo real (RTOS) y en optimización de software para dispositivos (DSO) para sistemas integrados de 32 y 64 bits. Los compiladores INTEGRITY® RTOS, velOSity™ kernel, µ-velOSity™ microkernel, los entornos de desarrollo integral MULTI® y AdaMULTI™ y el depurador TimeMachine™, exentos de derechos, (todos ellos Productos de Green Hill Software) ofrecen una solución de desarrollo completa para aplicaciones de gran complejidad y extrema fiabilidad. Las oficinas centrales de Green Hills Software están ubicadas en Santa Barbara, CA, EE. UU., y las oficinas centrales para Europa están situadas en el Reino Unido.

La seguridad se ha convertido en un tema candente en el mundo entero. Los bancos se esfuerzan por salvaguardar la seguridad de sus servicios en línea. Las tarjetas de crédito llevan chips de seguridad incorporados. La prensa informa con regularidad sobre las vulnerabilidades de los ordenadores y sobre ataques que siembran el desastre entre los confiados usuarios. Aún más inquietantes son las noticias sobre hackers que consiguen penetrar en los sistemas altamente protegidos de instituciones financieras y gubernamentales. Hay una guerra permanente entre los "buenos", que intentan hacer realidad la promesa de la tecnología informática y de Internet, y los "malos", que la utilizan para fines nefastos.

Yo también he sufrido recientemente un ataque de phishing. Recibí un correo electrónico del Bank of America Military Bank Online (o eso parecía). El correo electrónico decía que se había producido una modificación insignificante en mi cuenta y que debía conectarme para confirmar el cambio. Exploré el sitio web; parecía auténtico. Sin embargo, como jamás había servido en las Fuerzas Armadas, informé al Bank of America sobre el incidente y me respondieron que ya habían recibido denuncias parecidas. El (auténtico) sitio web del Bank of America incluye una página de preguntas frecuentes que informa sobre ataques de este tipo, mostrando capturas de pantalla que permiten identificar el phishing con mayor facilidad. Curiosamente, el indicador más común son los errores de ortografía.

Aunque mi gran experiencia con el phishing no deja de ser una mera anécdota para ser comentada entre amigos, me recordó lo importante que es no bajar la guardia en ningún momento. La preocupación y la curiosidad me indujeron a investigar un poco. Pregunté en mi lugar de trabajo sobre experiencias parecidas y escuché numerosas historias de identidades electrónicas robadas, software malicioso, más intentos de phishing, etc. Consulté algunos sitios que detectan vulneraciones de seguridad y descubrí unas estadísticas alarmantes. Averigüé que más de 800 ataques de hackers al Departamento de Seguridad Nacional de Estados Unidos tuvieron éxito. De acuerdo con el CERT (Computer Emergency Response Team), en 2007 se registraron más de 7.200 vulneraciones de la red.

Muchas vulnerabilidades de este tipo se pueden prevenir, lo que dependerá en gran parte de la forma en que estén diseñados e implementados los sistemas informáticos. El problema consiste en que no es fácil desarrollar y divulgar sistemas altamente seguros, especialmente teniendo en cuenta nuestra dependencia de sistemas heredados que se han venido utilizando durante años y que no fueron diseñados pensando precisamente en la seguridad.

¿Pero no podríamos, en nuestra época de innovación y automatización, inventar algo que permitiera solucionar estos problemas? Si los ordenadores son capaces de hacer volar (y aterrizar) aviones, conducir coches y poner en marcha un horno de cocina a través de Internet, ¿por qué no usarlos para asegurar que los programas que desarrollemos sean seguros?

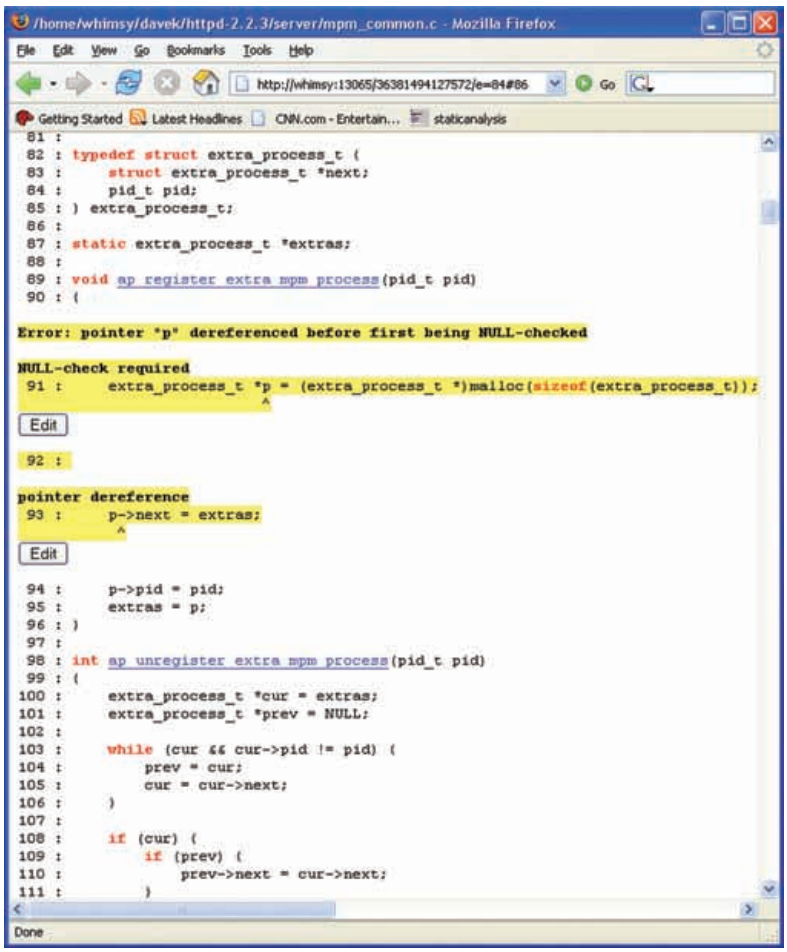
Y lo cierto es que pueden hacerlo

Una solución prometedora viene de una rama relativamente nueva de herramientas de desarrollo de software llamada análisis estático de código fuente. En pocas palabras, un analizador estático de código fuente es una herramienta que examina el código fuente de los programas y busca defectos que podrían resultar en lagunas para la seguridad.

Gran parte del software del mundo utiliza lenguajes de programación –tales como C, C++ y Java– que son un arma de doble filo: al tiempo que son potentes y flexibles, estos lenguajes pueden también hacer que los programadores se ahorquen con su propia soga. Muchos de mis colegas ingenieros tienen que luchar sistemáticamente con una sarta de trampas de la programación: "variables no inicializadas", "desbordamiento de la memoria intermedia" y "condiciones de ejecución", por mencionar sólo unas pocas. La buena noticia es que los analizadores estáticos pueden detectar estos fallos y notificarlos al desarrollador antes de desplegar el producto. Pues muy bien, se podría pensar, pero ¿cómo podría esto ayudar a hacer un ordenador más seguro? Primero veamos qué es lo que hace vulnerables los sistemas.

Manual de hacking para principiantes

Si creyéramos lo que vemos en las películas, podríamos pensar que los hackers son una panda de inconformistas de pelo verde adictos al ordenador que pasan gran parte de su tiempo en clubes de música tecno. Claro, y en su tiempo libre se meten en sistemas de alta seguridad para hacer de las suyas o para "pedir prestados" unos pocos dólares que les ayuden a seguir con su forma de vida poco convencional. La realidad es algo distinta. Aunque entre los hackers hay un buen número de rebeldes, muchos de ellos pertenecen a grupos bien organizados y financiados, que se pasan semanas enteras buscando la forma de penetrar en un determinado sistema. Gran parte del trabajo consiste en romperse la cabeza ante miles de líneas de código fuente o código máquina. Con el creciente uso de software de fuente abierta, gran parte del código queda a disposición de todo el mundo, lo que facilita considerablemente el trabajo de los hackers. Aunque el



trabajo de los hackers sea bastante menos glamuroso de lo que nos quieren hacer creer los estudios de Hollywood, su resultado puede ser mucho más siniestro de lo que se ve en la gran pantalla. No hacen simples bromas pesadas: los hackers consiguen penetrar en las redes de poder de la nación y hacerse con el sector financiero y los más importantes secretos militares.

Un atacante intentará varios de los siguientes trucos o todos a la vez

Encontrar una puerta trasera

El hacker buscará la forma de meterse en un sistema por medios poco tradicionales, como por ejemplo pretendiendo ser otro ordenador en vez de un usuario. Una vez que un hacker se ha metido en un sistema, podrá bajar un programa que cree una contraseña que se pueda usar más tarde. Durante el proceso, el hacker oculta las pruebas de su irrupción.

Conocer el sistema mejor que su desarrollador

Puesto que los grandes sistemas son implementados a menudo por cientos o incluso miles de desarrolladores, habrá partes de la implementación que se entenderán menos que otras (demasiados cocineros estropean la comida, por así decirlo). Un hacker no necesita entender el sistema entero, sólo sus partes más vulnerables.

Buscar caminos menos frecuentados

Toda aplicación o sistema tiene componentes que se utilizan más a menudo que otros, ejecutados con menor frecuencia. Normalmente es cierta la regla 80/20: durante el 80% del tiempo se ejecuta un 20% del código.

Esto quiere decir que un 20% del código es probablemente más fiable y seguro, en tanto que el 80% restante, al usarse menos, podría tener lagunas de seguridad que se tardará años en descubrir. En cuestiones de seguridad, el aforismo de que "una cadena es tan fuerte como su eslabón más débil" no podría ser más cierto: todo hacker necesita un camino de entrada y un camino poco frecuentado seguramente ofrecerá más posibilidades.

Buen cronometraje

El hacker que consiga entrar a menudo sólo tendrá una posibilidad de atacar antes de ser detectado. Por consiguiente, tendrá que medir cuidadosamente el tiempo que tiene para el ataque.

Los hackers encuentran la horma de su zapato

Para la ardua tarea de entender el código fuente y buscar vulnerabilidades, los hackers se sirven de una serie de trucos harto conocidos para exponer las vulnerabilidades.

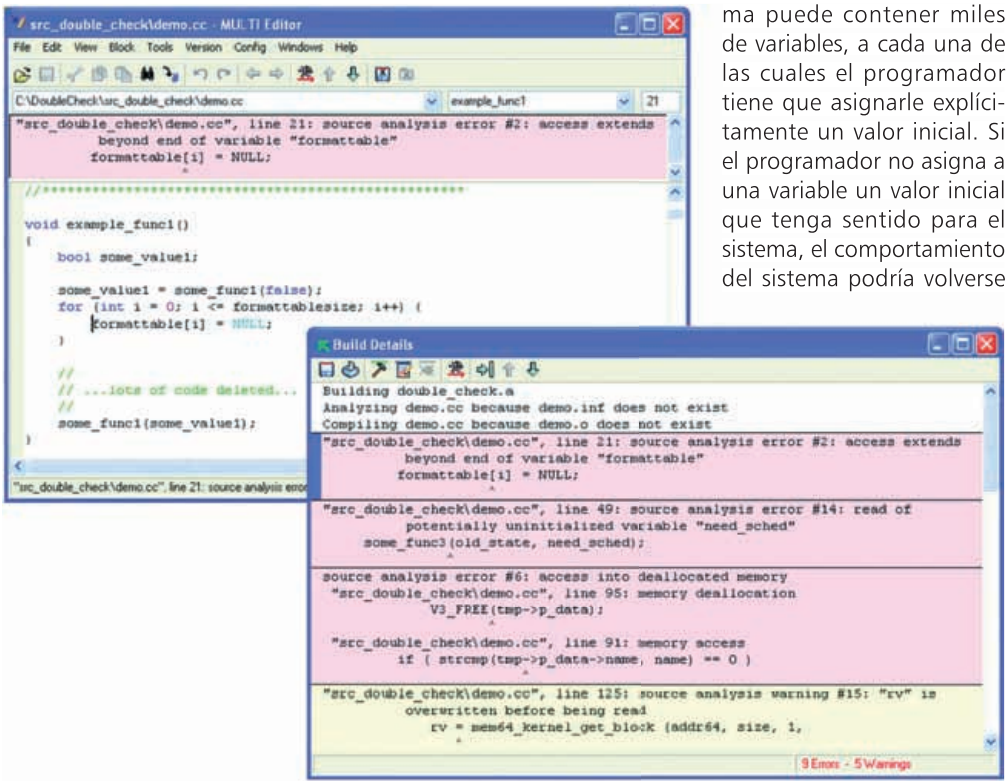
Los analizadores estáticos de código fuente están diseñados para detectar exactamente los escenarios propensos a causar vulnerabilidades y revelárselos al programador antes de liberar el sistema. Ejecutando el código a través de un analizador estático, los hackers pierden de golpe su ventaja. De hecho, una onza de prevención mediante análisis estático vale más que una libra de remedios para arreglar los estropicios causados por un ataque.

Éstos son algunos de los grandes trucos favoritos de los hackers y las formas en las que los combaten los analizadores estáticos:

- Desbordamiento de la memoria intermedia
- Problemas de variables (por ejemplo, variable sin inicializar)
- Desaprovechamiento de recursos
- Desbordamiento de la pila

Desbordamiento de la memoria intermedia

Siempre que un sistema necesita guardar datos, por ejemplo un nombre de usuario, tiene que asignar una parte de la memoria o una memoria intermedia. Según la memoria intermedia se va llenando con datos (por ejemplo, cuando un usuario teclea su nombre de usuario), si el programador no tiene cuidado, es fácil que el espacio de memoria previamente asignado resulte insuficiente, con lo que el sistema empezaría a sobreleer



o sobregrabar otros datos (por ej. los datos que indican al sistema qué código debe ejecutar a continuación). Así, un hacker habilidoso podrá insertar un programa suyo en el sistema y si su programa consigue acceder al sistema, podrá crear nuevas contraseñas y ocultar las pruebas de su irrupción en el proceso.

Solución

Los analizadores estáticos de código fuente son capaces de detectar con facilidad cualquier desbordamiento de memorias intermedias. Consultan la cantidad de memoria asignada y miran los accesos a la memoria intermedia. En cuanto detectan un acceso a la memoria intermedia que sobrepasa la memoria asignada inicialmente, los analizadores estáticos dan error, lo que permite subsanar el error totalmente antes de que el sistema quede liberado y expuesto a los ataques de los hackers.

Problemas de variables (por ejemplo, variable sin inicializar)

En lenguajes como C y C++, las variables son conceptos que permiten crear un estado que el sistema utiliza para tomar decisiones. Un programa puede contener miles de variables, a cada una de las cuales el programador tiene que asignarle explícitamente un valor inicial. Si el programador no asigna a una variable un valor inicial que tenga sentido para el sistema, el comportamiento del sistema podría volverse impredecible. Cuando el sistema se comporta de una forma impredecible y sin verificar, pueden abrirse rápidamente agujeros imprevistos en el sistema de seguridad.

Solución

Los analizadores estáticos toman nota de cuándo y con qué valores son inicializadas las variables. Los analizadores estáticos pueden distinguir también entre variables utilizadas exclusivamente por su valor propio (en ese caso el valor debe ser el esperado) y variables que se utilizan como puntero hacia un objeto que describe un estado algo más complejo (en ese caso el objeto que se señala y la variable deben tener un valor esperado). Los analizadores estáticos son capaces de localizar también los problemas de este tipo.

Desaprovechamiento de recursos

En el caso ideal, un sistema no deja de funcionar jamás. Una de las funciones fundamentales de un sistema consiste en asignar recursos a las aplicaciones que se ejecutan en éste. Recursos como la memoria, el acceso a disco, los accesos a puertos de medios, etc., son limitados, ya que dependen del hardware instalado. Cuando una aplicación deja de utilizar los recursos (por ejemplo cuando una aplicación deja libre un recurso o termina), el sistema tiene que "reciclarlos" para la siguiente aplicación que los solicite. Si el programador no instruye al sistema para que reclame estos recursos, un hacker puede crear un escenario vulnerable en el que, por ejemplo, se asigne más y más memoria hasta agotar por completo la memoria disponible. Esto lleva a menudo a ataques de "denegación de servicio", porque el sistema está ocupado en buscar recursos que simplemente no están disponibles.

Solución

Los analizadores estáticos se pueden personalizar para entender el tipo de recursos que el sistema puede ofrecer. Los sistemas tienen distintas formas de asignar, utilizar y reclamar recursos –lo que se conoce como API (application programming interface o interfaz de programación de aplicaciones)–. Un analizador estático puede asegurar que un programa sólo pueda utilizar un recurso del sistema después de haber asignado un recurso mediante un juego especial de instrucciones API. Sirviéndose de instrucciones API, un analizador estático puede asegurar también que el programa deje libre el recurso cuando ya no vaya a utilizarlo, evitando así el desaprovechamiento del recurso.

Desbordamiento de la pila

Esta vulnerabilidad es mucho más difícil de detectar, pudiendo ser especialmente peligrosa en los actuales sistemas multihilo. Una pila de funciones es un espacio de memoria local que se tiene que haber asignado previamente a un programa para que pueda ejecutarse correctamente. El mayor problema con las pilas de funciones es saber cuánta memoria debe asignar el programador en el momento de crear el nuevo hilo. Al programador le puede resultar difícil calcular esta cantidad con exactitud. Así que, al final, la mayoría de los programadores suelen calcular la memoria que se necesita a ojo (y a veces se pasan en varios kilobytes, para que no falte).

Si tenéis la impresión de que la programación es como el salvaje oeste, habéis acertado. Si los programadores se equivocan con este valor (es decir, si se quedan cortos), la pila de funciones de un hilo de ejecución podría corromper la pila de funciones de otra y causar, así, fallos sutiles, difíciles de reproducir y casi imposibles de detectar. Peor aún, a menudo se cuelan en el código definitivo pese a todas las pruebas y procesos de pregunta y respuesta.

Solución

Hasta hace poco, los analizadores estáticos no tenían forma de combatir este fallo. Lo que hacía imposible detectarlo era que el código fuente estaba bien, lo que fallaba era la interacción entre el compilador (herramienta que traduce el código fuente a código máquina) y la estructura de ejecución del programa. La solución viene en forma de un nuevo diseño de analizadores estáticos, que los integra en los compiladores. El resultado de esta sinergia entre analizadores estáticos y compiladores da como resultado la detección de toda una nueva serie de problemas.

Implementación de la seguridad

A menudo escuchamos que la seguridad es difícil de implementar. ¿Y eso por qué? La mejor forma de formularlo es decir que resulta muy difícil encontrar el equilibrio entre la manejabilidad y la seguridad de un sistema. Es muy fácil crear un sistema seguro pero imposible de manejar –por ejemplo uno que no esté conectado a una red externa o que esté desconectado.

¿Y por qué es tan difícil combinar la seguridad con la manejabilidad? La respuesta es sencilla: debido a la enorme cantidad de código fuente que se necesita para un sistema típico. Los sistemas actuales no son grandes sino gigantescos, y se componen de millones de líneas de código. Todas las demás razones no son más que corolarios de este hecho fundamental. Y esto nos lleva a la primera regla:

Regla 1: la seguridad es difícil de lograr porque los sistemas son gigantescos

Los sistemas se implementan a través del código fuente. Cada línea de código en una fuente base ordena al sistema ejecutar una serie de instrucciones. En otras palabras, cada línea de código podría hacer algo que comprometiera la seguridad y fiabilidad del sistema. Para implementar un sistema seguro, alguien o algo tiene que comprobar todas y cada una de las líneas de código fuente.

Pero aquí no termina la cosa. ¡Cada línea fuente depende e interactúa con todas las líneas que tiene alrededor! Un sistema no es simplemente una colección de líneas fuente –más bien podríamos decir que se trata de un tejido muy tupido en el que las líneas fuente son los hilos–. Así que no sólo necesitamos comprobar cada una de las líneas fuente, sino también las interacciones entre cada una de las distintas líneas fuente con el resto.

De acuerdo con Wikipedia, la fuente base de Microsoft Windows consiste en unos 50.000.000 de líneas de código fuente. Parece una cifra gigantesca, pero ¿qué significa exactamente? Me viene a la mente una novela larguísima –Guerra y paz. Esta obra maestra de León Tolstói, que describe la Rusia del siglo xix, sólo tiene unas 1.500 páginas, lo que equivale a unas 100.000 líneas de texto o a la unidad que me acabo de inventar: un GYP (Guerra y paz). Conforme a esto, la fuente base de Windows sería de unos 500 GYP, lo que supondría leer y estudiarse a fondo quinientos libros del tamaño y de la complejidad de Guerra y Paz. ¡Menuda pesadilla! (sin ánimo de ofender a Tolstoi).

Una aplicación más pequeña, como por ejemplo el servidor de web Apache, equivaldría a unos 1,5 GYP. La fuente base de uno de los sistemas operativos más populares, el Linux Debian, se llevaría la palma, con 2.000 GYP.

El problema de entender el código no aumenta en proporción al volumen de código. La solución no consiste en meter simplemente a más programadores a estudiar y comprobar el código, ya que la complejidad aumenta exponencialmente. En un determinado punto, la complejidad llega a un límite en el que deja de ser posible obtener un cierto nivel de seguridad y, según parece, los desarrolladores de Windows y Linux han abandonado la idea hace ya varios cientos de GYP. Así, no es de extrañar que mi equipo de Windows se baje cada pocos días nuevas actualizaciones críticas para la seguridad. Es imposible encontrar todas las vulnerabilidades.

¿Y cómo nos hemos metido en este lío? El mayor problema es que casi todos los sistemas son versiones evolucionadas que han surgido de elementos que fueron desarrollados hace décadas. Y cuando la gente añade cosas nuevas, rara vez quita lo viejo. ¿Y eso por qué? Porque nadie quiere correr el riesgo de quitar algo que podría ser importante para algo. Y así se crea un círculo vicioso: cuanto más código se añade, menos probable resulta que alguien entienda la complejidad del sistema y menos probable es que alguien se atreva a quitar algo viejo, pero en algún momento hay que añadir más código…, y así sucesivamente.

Cuando estudiaba en la universidad, los profesores nos animaban a participar en los proyectos de clase, porque ésa sería la única vez que escribiríamos un nuevo programa desde cero. Tenían razón.

Recapitulando, las dos reglas derivadas de la regla 1 son:

Regla 2: la seguridad es difícil de lograr, porque no importa lo viejo que sea o lo poco que se use: por lo general el código jamás se elimina.

Regla 3: la seguridad es difícil de lograr porque, debido a las interdependencias, la complejidad del código fuente crece exponencialmente.

Resulta que los analizadores estáticos de código fuente tienen soluciones también para eso. Analizar una línea fuente cada vez no es suficiente. Lo que se necesita realmente es analizar la interacción entre dos líneas fuente cualquiera que pudieran tener algo en común (como por ejemplo, vía de ejecución, acceso a los mismos datos, etc.). Y eso es exactamente lo que hacen los analizadores estáticos de código fuente, sin importar la distancia física que pueda haber entre dos líneas de código. Esto se consigue construyendo internamente estructuras de datos mayormente en base a técnicas de optimización de compiladores, tales como análisis del flujo de datos hacia adelante y hacia atrás, y gráficos de control de flujo. Puede sonar como un montón de chorradas técnicas, pero lo bueno es que los analizadores estáticos son capaces de localizar las interdependencias que incluso los programadores más expertos serían incapaces de detectar.

Por otra parte, como emplear a miles de programadores para leer y verificar código fuente manualmente

probablemente provocaría un motín, los analizadores estáticos analizan alegremente sistemas o aplicaciones de cualquier tamaño con gran profundidad y rigor. También se somete a inspección cualquier código heredado que apenas se usa (y puede contener lagunas de seguridad); los analizadores estáticos escrutan a fondo cualquier rincón y fisura hasta dejar toda la base limpia como una patena.

No es la panacea universal, pero...

Vale, admitamos que los analizadores estáticos son fenomenales. Ejecutémoslos en cualquier sistema, dejemos que arreglen todos los problemas que encuentren y ya está, ¿no?

Bueno, no exactamente. Aunque los analizadores estáticos encuentran las clases de problemas que podrían causar irrupciones en un sistema, hay otra clase de problemas que los analizadores estáticos no pueden solucionar ni hoy ni posiblemente mañana. De igual modo que no tenemos programas lo suficientemente inteligentes como para diseñar por sí solos un sistema completo desde la nada, tampoco tenemos analizadores estáticos lo suficientemente inteligentes como para entender un sistema de la forma en que lo puede entender un ser humano. Por fortuna para nosotros, los programadores, los seres humanos no son obsoletos.

Algunos cínicos ven en los analizadores estáticos de código fuente otra arma más que viene a engrosar el arsenal de los hackers. Probablemente se trate de los mismos cínicos que, hace unas décadas, cuando salieron, pensaban que los depuradores de líneas fuente eran una mala idea (actualmente resulta impensable embarcarse en cualquier proyecto de cierta importancia sin utilizar depuradores de líneas fuente).

Este punto de vista tan cínico es fácil de rebatir. En primer lugar, el análisis estático del código fuente sólo funciona si se tiene acceso al código fuente, y generalmente los hackers sólo tendrán acceso al código fuente de programas desarrollados a partir de fuentes abiertas. Según mis recientes comprobaciones, la mayoría de los sistemas utilizados actualmente son de propiedad, especialmente para entornos de alta seguridad. Así que no es tan fácil obtener el código fuente.

En segundo lugar, los hackers necesitan encontrar una sola vulnerabilidad para quebrantar la seguridad, así que para ellos es una simple cuestión de tiempo encontrar la forma de entrar –con analizador estático o sin–. Cuanto más complejo es el sistema, tanto más fácil es encontrar un camino. Por otra parte, los diseñadores de sistemas necesitan detectar y eliminar el mayor número posible de vulnerabilidades (con suerte todas). Cuanto más complejo es el sistema, tanto más difícil es detectar todas las vulnerabilidades. Por lo tanto, los principales beneficiarios del análisis estático son los diseñadores, ya que no sólo les ahorra tiempo, sino que les permite también diseñar sistemas más seguros.

Finalmente, los analizadores estáticos realizan un montón de trabajo que, realizado por seres humanos, tendría un coste prohibitivo. Lo más irónico es que podemos crear sistemas más complejos gracias a una mayor velocidad y capacidad del hardware y a la ayuda de herramientas de software y depuradores más eficientes. Actualmente, los analizadores estáticos, una nueva línea de herramientas de software, nos ayudan a superar el reto de tener que lidiar con una complejidad cada vez mayor.

Si hace unos años los analizadores estáticos nos parecían artículos de lujo, hoy lo único que podemos hacer es recomendarlos encarecidamente y, en un futuro próximo, serán una metodología indispensable para implementar y comprobar cualquier software.