ZIMPERIUM

Guía práctica de "hardening" para aplicaciones

¿Qué es el "hardening" para aplicaciones?

El hardening para aplicaciones, también denominado como protección para aplicaciones o protección integrada en las aplicaciones, es el proceso de modificar una aplicación ya existente para hacerla más resistente a los ataques más comunes de los hackers como un uso indebido, uso de ingeniería inversa o la adquisición del control de la aplicación.

Con este proceso se garantiza la protección tanto de la propia aplicación como de los datos que se usan en ella. Es un método de seguridad para aplicaciones obligatorio que se rige por diferentes normas y estándares de seguridad. Hay diferentes técnicas de hardening: las empresas pueden optar por todas ellas o por solo unas cuantas, dependiendo de su nivel de exposición y tolerancia al riesgo y requisitos de rendimiento.

Por qué es una cuestión importante

Las aplicaciones tienen vulnerabilidades de las que los hackers se pueden aprovechar. Uno de los puntos de vulnerabilidad más frecuentes son los errores de codificación. Lo normal en este sector es que haya entre 15 o 50 errores por cada 1000 líneas de código. Como las aplicaciones modernas tienen entre decenas y miles de millones de líneas de código, hay mucha probabilidad de que existan errores potenciales de diferentes tipos de gravedad en cualquier aplicación.

Los hackers pueden aprovecharse de estas vulnerabilidades para robar datos sensibles y de propiedad intelectual, averiguar claves criptográficas o hackear la aplicación con fines maliciosos.

El hardening en las aplicaciones sirve como una especie de escudo ante estos ataques. Dependiendo del nivel de hardening usado, puede proteger de forma parcial o total la aplicación contra el análisis estadístico de su código fuente, el análisis dinámico de la aplicación en tiempo real, los ataques a los controles de la aplicación o al propio sistema, e incluso contra la manipulación del código.

Mejores prácticas para proteger las aplicaciones

Como en cualquier otro aspecto del desarrollo de aplicaciones, hay que evaluar el riesgo, el coste y los beneficios. Si la IP de su aplicación no tiene casi valor (o ninguno), no almacena ni da acceso a datos sensibles, no se conecta con otros sistemas y no usa claves criptográficas, entonces puede que con un método de hardening sencillo le sea suficiente.

Si desea protegerse contra la piratería, el robo de datos o el uso indebido de los códigos, o en caso de que su aplicación se use como punto de entrada y pueda comprometer otros sistemas... entonces sí que sería necesario implementar un hardening más sofisticado.

1 Protecciones básicas para las aplicaciones

Cambio de nombre

Esta es una forma bastante rudimentaria de ofuscación del código que sirve para cambiar la variable identificable y los nombres de segmentos como “user_name” o “grant_access” por segmentos con caracteres aleatorios sin sentido para que el hacker no los entienda y se sienta confuso. Sin embargo, la ejecución del programa en sí no ve alterada. Hay que tener en cuenta que esta estrategia de cambio de nombre no es eficaz contra los desofuscadores, depuradores o herramientas de control.

Insertar código ficticio

Con este enfoque lo que se hace es añadir código a la aplicación de manera que no afecte a la ejecución o a la lógica del programa pero que de alguna forma haga que el código de ingeniería inversa se torne mucho más difícil de analizar. Muchos de los ofuscadores gratuitos que cambian el nombre de los segmentos también siguen este método.

Eliminación de metadatos y códigos no utilizados

Al eliminar los códigos muertos, depurar la información y los metadatos no esenciales de las aplicaciones, se reduce la cantidad de información que un hacker puede conseguir y utilizar.

2 Ofuscación avanzada

La ofuscación del código es una técnica contra la ingeniería inversa. El objetivo es hacer que el código sea lo más confuso y difícil de entender posible pero sin dejar de ser funcional. Al contrario que con los métodos de ofuscación del código que hemos descrito más arriba, este método de ofuscación es más difícil de corromper, sobre todo si se combina con otras técnicas de hardening más avanzadas.

Ofuscación del flujo de control

En la ofuscación del flujo de control se modifica la estructura básica del nombre de las subrutinas para hacer que el código sea más difícil de rastrear. De esta manera, se impide que los descompiladores reconstruyan el código fuente eliminando los patrones que suelen buscar. Por ejemplo, algunas funciones integradas (como sustituir las acciones por el cuerpo del método en sí), sustituir las acciones para las subrutinas por saltos calculados y convertir las construcciones tipo árbol en segmentos planos. Este último método, llamado aplanamiento del flujo de control, utiliza un despachador para controlar el flujo en vez de apelar a rutinas directamente desde otras rutinas, tal y como se puede ver en la Figura 1.

Ofuscación de metadatos y mensaje de la llamada en el Objetivo C.

En el Objetivo C, los mensajes a las instancias de los objetos se resuelven solo en tiempo real, lo que significa que los mensajes de las llamadas se almacenan en el código binario de forma simple. Los hackers pueden usar esto como ataque para manipular la lógica de la ejecución. El código del objetivo C se puede proteger mediante llamadas con texto plano para ofuscar el código fuente y así conseguir que no sea ni editable ni fácil de leer. También es importante cifrar los metadatos del Objetivo C, como los nombres de las clases, protocolos de los métodos o argumentos y tipos, para ocultar esta información que resulta muy útil para las herramientas de análisis estático. Estos datos encriptados solo se pueden descifrar en tiempo real al cargar la aplicación ofuscada.

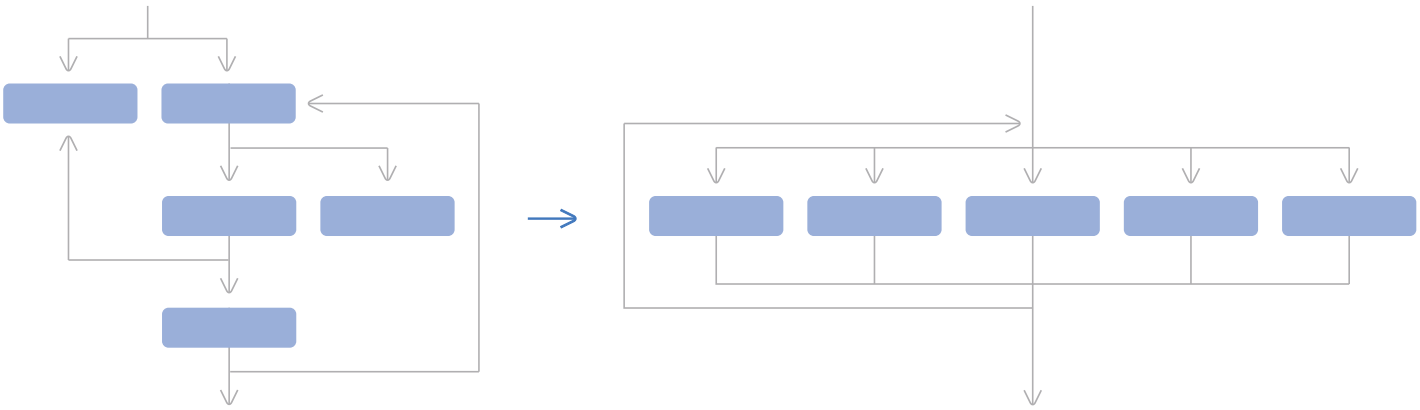


Figura 1.

Aplanamiento del flujo de control para impedir ingeniería inversa.

3 Protección contra

Los ingenieros de software usan depuradores de forma legítima para encontrar errores de codificación, pero en manos de los hackers son una poderosa herramienta para hacer ingeniería inversa del código. Con ellos pueden saber muchas cosas del estado de la aplicación, extraer información sobre las funciones en uso, sobre los valores de las variables y otros datos que aparecen en ubicaciones de la memoria arbitrarias. Por lo general, lo que hacen es establecer puntos de interrupción en la aplicación: cuando alcanzan ese punto, el programa detiene la ejecución y el usuario comprueba lo que ocurre en ese punto. Algunos depuradores también se encargan de desmontar y desarticular.

Un método de antidepuración muy básico es insertar llamadas de la API para saber más sobre el proceso y el sistema y comprobar si hay algún depurador activo. Por ejemplo, se obtendrán datos como IsDebuggerPresent, CheckRemoteDebuggerPresent, o se harán comprobaciones de la separación del depurador y CloseHandle. Estos datos son bastante sencillos de añadir a un código ya existente en una aplicación. Al mismo tiempo, para un hacker es bastante sencillo acceder a ellos.

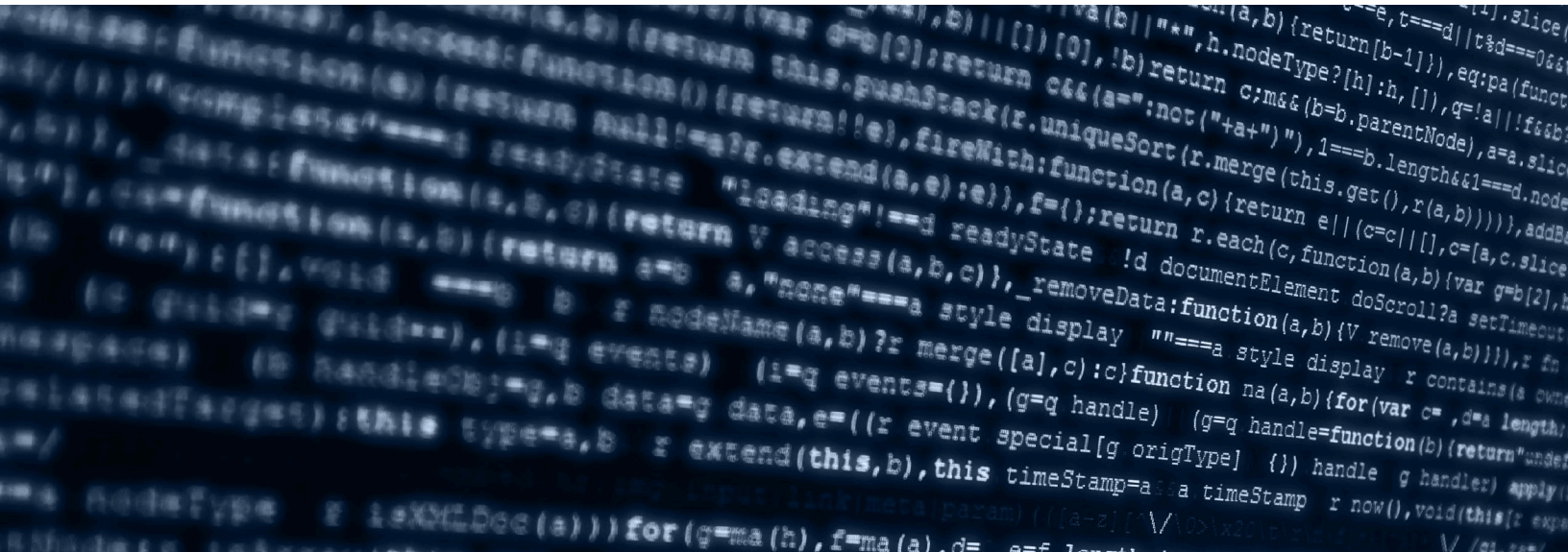
Uno de los métodos más efectivos para evitar la depuración se denomina "código modificado contra la depuración", mediante el cual se inserta código en la aplicación que detecta la presencia de un depurador e implementa las medidas de defensa necesarias.

4 Sistema binario

Normalmente, analiza el proceso de la aplicación y lo compara con el patrón esperado para ver si hay puntos de interrupción en los que se haya insertado un depurador. Se puede usar la función syscalls del modo Kernel para eludir los bloqueos del modo de usuario insertados por los hackers, lo que obligará al hacker a cambiar su kernel y a mejorar mucho su forma de atacar el sistema si quiere tener éxito.

Estos compiladores encriptan y comprimen el código e incluyen un stub que lo descifra en tiempo real. Los hackers pueden usar este truco para ocultar el malware de los antivirus ya que el código compilado no es reconocible.

Si se compila el código de las aplicaciones es casi imposible que los hackers consigan hacer ingeniería inversa, ya que no podrán obtener el código ni descomponiéndolo. La única alternativa sería que hicieran una captura del código ya descifrado y la guardasen en la memoria y luego, hicieran un proceso de ingeniería inversa. Con esto básicamente se evitan todos los análisis de las estadísticas, haciendo que los atacantes se vean obligados a aplicar¬técnicas de análisis dinámicos que les llevarán mucho más tiempo.



5 Diversificación

Cuando un hacker consigue entrar en una instancia en una aplicación, es bastante probable que cree una herramienta automatizada para entrar en otra instancia en la misma aplicación. Esto se suele llamar ruptura de clase o ruptura puntual y es un problema que se expande (BORE).

La diversificación altera el código para crear instancias del mismo software que sean idénticas a nivel funcional pero cuya superficie del código no lo sea, ni en forma como ni en estructura.

Esto sirve para frustrar los intentos de los hackers de hacer uso de la información obtenida y extrapolar esto a otras ubicaciones. Es mucho más difícil desarrollar un esquema universal de hackeo en instancias de software diversificadas. A consecuencia de ello, deberían hackear cada instancia de un software de forma individual.

La diversificación se puede llevar a cabo a través de una población de aplicaciones, para que cada instancia de la aplicación sea diferente. Pero es una técnica muy útil para la diversificación de versiones del código. Por ejemplo, si un hacker consigue hacer una ingeniería inversa de la Versión 2.3 de una aplicación, tendrá que empezar de nuevo todo el proceso cuando se lance la versión 2.4.

Y no se olvide de la protección mediante las claves encriptadas

Una parte fundamental para proteger sus aplicaciones reside en las claves encriptadas. Lo que suele pasar es que las claves estén codificadas en la aplicación, desde donde los hackers pueden extraerlas fácilmente, o están expuestas en la memoria mientras se utilizan en operaciones criptográficas. Si las claves se han visto afectadas, está igual de indefenso que si no tiene nada encriptado.

Seguridad basada en hardware

Las protecciones basadas en hardware, como los módulos de seguridad de hardware (HSM), los módulos de plataforma segura (TPM) y los entornos d e ejecución confiable (TEE), pueden ofrecer una protección muy sólida de las claves criptográficas, pero son muy difíciles de implementar en las diferentes plataformas y pueden ser vulnerables si el propietario ha obtenido privilegios de administrador, por ejemplo, con un teléfono desbloqueado. También suelen ser susceptibles en algunos tipos de ataques a canales laterales.

Keystores

Muchas plataformas y OSes ofrecen keystore, un sistema de almacenamiento de claves con claves criptográficas (Android Keystore, Java Keystore, Apple Secure Enclave, Windows Keystore). Esto debería ser suficiente para las aplicaciones que no tienen mucho valor o que no contienen información sensible. Aunque los algoritmos criptográficos compatibles y las operaciones suelen ser limitadas y las operaciones criptográficas se tienen que volver a implementar en cada plataforma.

Criptografía de caja blanca

Con la criptografía de caja blanca, las claves se ocultan siempre que están en uso, se muevan o almacenen. En general, la criptografía de caja blanca no es considerada tan segura como el hardware de seguridad integrado y los cálculos suelen ser más lentos. Sin embargo, los algoritmos de software de caja blanca pueden implantarse en dispositivos que carecen de soporte de hardware y funcionan de manera uniforme en todas las plataformas. Además, las claves permanecen seguras incluso si alguien consigue tener privilegios de administrador al dispositivo. Algunas bibliotecas con criptografía de caja blanca ofrecen una protección excelente contra los ataques de canal lateral.

6 Protecciones contra un uso indebido

A menudo, los ataques se hacen para modificar el código de la aplicación y robarlo para poder usarlo con fines propios. Estos hackers pueden instalar rootkits y entradas secretas, desactivar el control de seguridad, trastocar la autenticación e incluir código malicioso que registre los toques al teclado, robe datos, usar los privilegios del usuario o realizar otras acciones maliciosas. Las protecciones contra un uso indebido, también denominadas protección de aplicaciones en tiempo real (RASP), detectan y evitan los ataques que alteran su aplicación.

Comprobación de la integridad

La comprobación de la integridad refuerza las aplicaciones mediante la inserción de miles de pequeñas piezas de código superpuestas, denominadas comprobadores, como se muestra en la figura 2. En el momento en el que se ejecuta una aplicación, los comprobadores verifican un segmento concreto para ver si ha sido manipulado o no. En caso afirmativo, se ponen en marcha medidas de protección para preservar la integridad de la aplicación, como notificar al usuario, activar una respuesta personalizada, generar un mensaje de registro o incluso cerrar el programa.

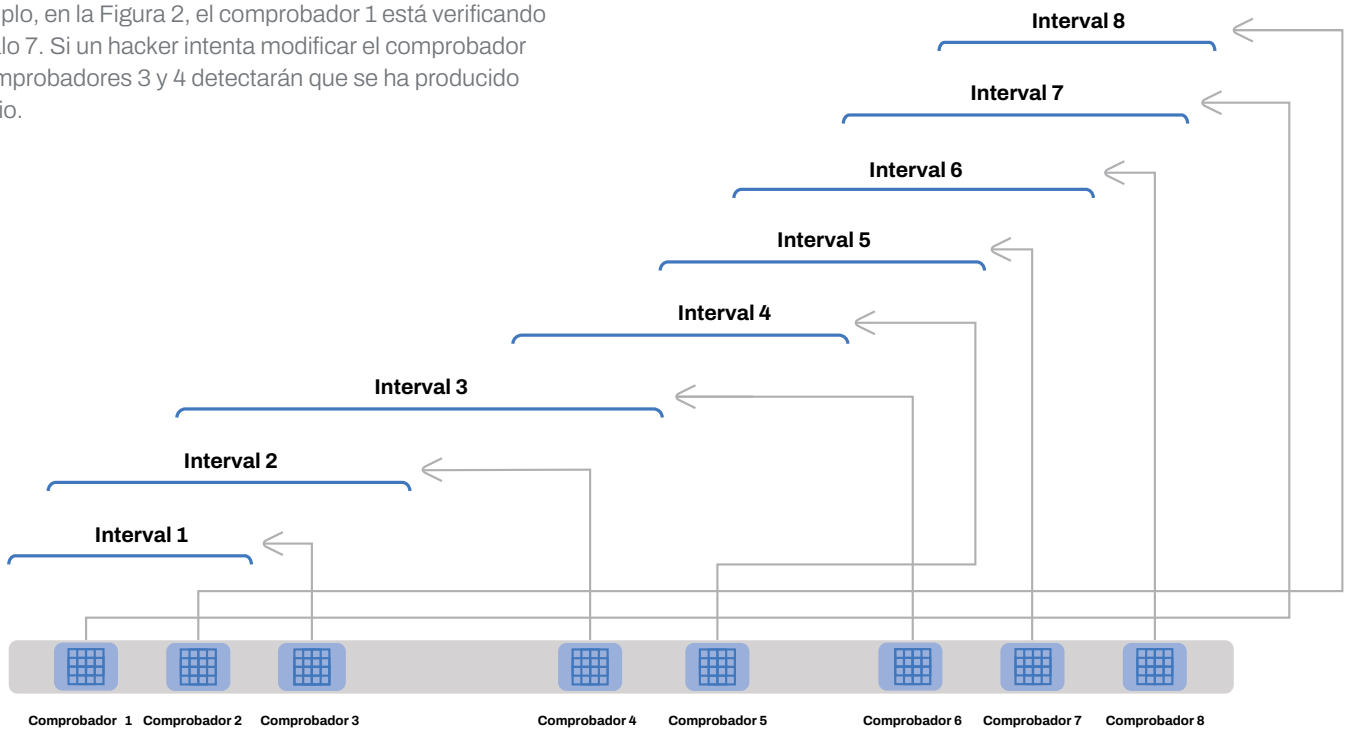
Por ejemplo, en la Figura 2, el comprobador 1 está verificando el intervalo 7. Si un hacker intenta modificar el comprobador 1, los comprobadores 3 y 4 detectarán que se ha producido un cambio.

Protección contra el swizzling

El método swizzling es una función del lenguaje Objetivo C, muy usado en los dispositivos Apple, que ha sido cooptada por los hackers para atacar y cambiar el comportamiento de las aplicaciones. Este método modifica la aplicación mediante la asignación de un nombre de método de clase a una implementación de método diferente, cambiando su comportamiento en tiempo real. Se usa de forma legítima para sustituir o ampliar los métodos de las clases en los binarios cuyo código fuente no está disponible. Pero en las manos de un hacker, este lo puede usar para redirigir los datos de una tarjeta de crédito a otro servidor, para extraer credenciales, guardar información del cliente o para otros fines.

Para intentar reducir al máximo esta posibilidad, hay que intentar trabajar todo el tiempo con C++ y usar lo menos posible las clases ObjC. Las soluciones de escudo para las aplicaciones, como zShield de Zimperium, suelen incluir mecanismos para detectar el swizzling y ejecutar acciones de defensa.

Figura 2.



Detección de jailbreak de iOS y rooting de Android

El desbloqueo de un dispositivo iOS consiste en eliminar las limitaciones de software y hardware establecidas por Apple y obtener acceso como administrador al dispositivo. De forma parecida, el rooting proporciona un acceso privilegiado al sistema operativo en los dispositivos Android, anulando las limitaciones establecidas por las operadoras de telefonía y los fabricantes de hardware. Una vez que el dispositivo ha sido desbloqueado o rooteado, los controles de seguridad instalados se violan y puede ser que una aplicación falsa acceda a su aplicación, datos, credenciales y claves. Una parte fundamental de la protección de las aplicaciones es la capacidad para detectar un dispositivo desbloqueado o rooteado y tomar las medidas correctivas necesarias.

Comprobaciones extra contra usos indebidos

Cuantos más métodos de uso indebido se detecten, más potente será su escudo. Las mejores prácticas para su aplicación incluyen mecanismos de respuesta y detección contra ataques específicos. A continuación, presentamos algunas de las protecciones más comunes.

Comprobación de la función de la persona que llama

Un archivo de aplicación ejecutable contiene una serie de funciones. En general, se trata de una lógica predeterminada sobre cómo y cuándo se activan estas funciones en tiempo real. Sin embargo, un hacker con experiencia puede analizar el código binario, encontrar puntos vulnerables y alterar el flujo original del programa activando algunas funciones de una forma poco usual, por ejemplo en Windows, usando un DLL.

Para protegerse de dicha manipulación de las llamadas, se puede crear una lista blanca de módulos (*.dll or *.exe files) permitidos para activar ciertas funciones del código de la aplicación. Guarde las firmas de estos módulos autorizados dentro del binario de la aplicación y úselas en tiempo real para verificar los módulos que activan las funciones.

Comprobación cruzada de la biblioteca compartida

Una de las formas que usan los hackers para reemplazar o modificar las bibliotecas compartidas es a través de una aplicación. Para disminuir el riesgo de este tipo de ataques, hay que limitar el uso de bibliotecas compartidas siempre que sea posible. Si su aplicación usa bibliotecas compartidas, implemente un control cruzado para detectar cualquier uso indebido de la biblioteca. Por ejemplo, puede crear firmas criptográficas para cada biblioteca o hacer actualizaciones aleatorias en tiempo real para garantizar que coincidan.

Verificación de firmas binarias Mach-O

Todas las aplicaciones macOS, iOS y tvOS que se distribuyen a través de App Store se firman con una clave privada de Apple, lo que sirve para evitar la piratería y la distribución no autorizada. Sin embargo, los miembros del programa de desarrolladores de Apple pueden volver a firmar cualquier aplicación con una clave propia incluida en el certificado de desarrollo y permitir que la aplicación se ejecute en los dispositivos de desarrollo correspondientes. Muchos servicios de internet vuelven a firmar de forma ilegal aplicaciones para ofrecer aplicaciones de pago de forma gratuita. Puede insertar protecciones en su aplicación para protegerla contra esto y contra la distribución no autorizada de aplicaciones usando un archivo con formato Mach-O (utilizado por las aplicaciones de macOS, iOS y tvOS).

Protección de las licencias de Google

La piratería de las aplicaciones sigue siendo una de las mayores preocupaciones de los desarrolladores de Android. Aunque Android dispone de una biblioteca antipiratería para comprobar y hacer cumplir las licencias en tiempo real, esta biblioteca basada en Java y puede ser fácilmente hackeada. Una forma efectiva de protegerse contra la piratería es reemplazar la biblioteca de verificación de licencias de Google Play por una implementación reforzada que sea más difícil de modificar mediante ingeniería inversa.

7 Acciones de defensa dentro de las aplicaciones

Cuando su aplicación identifique una amenaza, se debería activar una respuesta de defensa. Lo más sencillo es cerrar la aplicación. Sin embargo, algunos ataques no son tan peligrosos y no debería ser necesaria dicha acción. O puede que sea una buena estrategia hacerle creer al hacker que ha tenido éxito y que así deje de atacar la aplicación. Por ejemplo, si desea proteger un juego, puede corromper secretamente el mapa del juego en lugar de detener la aplicación, para que al hacker le dé la impresión de que está craqueada pero en realidad solo se haya transformado y no se pueda jugar al juego.

Las acciones de defensa más comunes incluyen:

- Bloquear el acceso a la cuenta
- Generar un mensaje de registro que se enviará al administrador
- Detener la ejecución de comandos
- Corromper elementos de la aplicación para que los hackers crean que han tenido éxito. En realidad, solo tendrán un acceso mínimo
- Eliminar los datos sensibles
- Cerrar completamente la aplicación

Las acciones de defensa automáticas en tiempo real suponen una clara ventaja. Son una alternativa mejor que la de generar alertas que tendrán que ser inspeccionadas y resueltas de forma manual. Este tipo de detección es compleja y la respuesta suele requerir una solución de protección más avanzada e integrada en la aplicación.

Protección en las aplicaciones con el escudo de Zimperium

El escudo de Zimperium **zShield** es una gran herramienta de defensa para su aplicación. Evita un uso indebido, la ingeniería inversa y otras de las técnicas usadas por los hackers para acceder a datos sensibles y al código de la aplicación. zShield usa varios métodos de defensa como la ofuscación de códigos, el aplanamiento de códigos y la detección de la intrusión en tiempo real para reforzar la seguridad de cualquier aplicación en tan solo unos minutos.

zShield es uno de los componentes de la Mobile Application Protection Suite (MAPS) de Zimperium, la única solución unificada que combina una protección integral dentro de las aplicaciones con una visibilidad centralizada de las amenazas. MAPS también incluye:

zKeyBox: una herramienta de criptografía de caja blanca de última generación que garantiza el secretismo de las claves criptográficas dentro del código de la aplicación, incluso en el entorno de ejecución.

zDefend: una protección integrada muy avanzada que permite que las aplicaciones móviles determinen de forma inmediata si el dispositivo de un usuario está en riesgo, si hay ataques a la red o si se han instalado aplicaciones maliciosas. Los desarrolladores de las aplicaciones pueden configurar métodos correctivos en caso de que se detecte una amenaza.

zScan: una solución para testar las aplicaciones móviles e identificar riesgos de privacidad, seguridad y cumplimiento en la fase de desarrollo de la aplicación y antes de que se lance al mercado.

Para obtener más información, visite zimperium.com.