ZIMPERIUM®

Guide pratique du durcissement d'application

Qu'est-ce que le durcissement d'application ?

Le durcissement d'application, également connu sous le nom de blindage d'application ou de protection « in-app » ou intégré, est le processus de modification d'une application existante pour la rendre plus résistante aux tentatives de piratage telles que l'ingénierie inverse, la violation et la surveillance.

Le durcissement d'application protège à la fois l'application elle-même et les données qu'elle utilise. Cette méthode de sécurité d'applications est exigée par différentes réglementations et normes. De nombreuses techniques de durcissement d'application existent : les entreprises peuvent choisir de les utiliser toutes ou seulement un sous-ensemble en fonction de leur exposition, de leur tolérance au risque et de leurs exigences de performance.

Pourquoi est-ce important ?

Tant qu'il y aura des applications, il y aura des vulnérabilités applicatives exploitables par des pirates. Les erreurs de codage ordinaires sont l'un des moyens les plus courants d'introduire des vulnérabilités. La norme industrielle se situe entre 15 et 50 erreurs pour 1 000 lignes de code. Les applications modernes contiennent des dizaines de milliers, voire des millions de lignes de code. Des milliers de failles potentielles de différents niveaux de gravité peuvent donc exister dans une application.

Les pirates peuvent exploiter ces vulnérabilités pour voler la propriété intellectuelle et les données sensibles, obtenir des clés cryptographiques ou détourner l'application à des fins malveillantes.

Le durcissement d'applications protège ces vulnérabilités contre les attaques. Le degré de durcissement utilisé permet une protection partielle ou totale contre l'analyse statique de votre code source, l'analyse dynamique de votre application lors de l'exécution, les attaques visant à contourner les contrôles de l'application ou du système, et même la violation du code.

Pratiques conseillées pour durcir les applications

Comme pour tout élément du développement d'une application, vous devez évaluer les risques, les coûts et les avantages. Si votre application n'a pas ou a peu de valeur IP, ne stocke d'informations sensibles ou n'y accède pas, ne se connecte pas à d'autres systèmes et n'utilise pas de clés cryptographiques, les méthodes de durcissement de base de l'application peuvent suffire.

Si vous souhaitez protéger votre application contre le piratage, le vol de données ou la violation du code, ou si elle peut être utilisée comme point d'entrée pour compromettre d'autres systèmes, le choix d'un durcissement plus sophistiqué est recommandé.

1 Protections de base d'une application

Renommer

Cette forme rudimentaire de brouillage du code modifie les noms de variables et de méthodes identifiables, tels que « user_name » ou « grant_access », en chaînes de caractères dénuées de sens afin de les rendre confus pour un pirate. Le programme continue de s'exécuter à l'identique. Le renommage n'est toutefois pas efficace contre les désobscurcisseurs, les débogueurs ou les outils de surveillance.

Insérer un code fictif

Cette approche permet d'ajouter un code étranger à l'application, sans affecter l'exécution ou la logique du programme, mais en rendant le code obtenu par rétro-ingénierie un peu plus difficile à analyser. De nombreux brouilleurs gratuits modifient le nom insèrent également un code fictif.

Supprimer le code inutilisé et les métadonnées

La suppression du code mort, des informations de débogage et des métadonnées non ¬essentielles des applications, réduit le volume des données susceptibles d'être utilisées par un pirate.

2 Brouillage avancé

Le brouillage du code est une technique essentielle de lutte contre la rétro-ingénierie. L'objectif est de rendre le code aussi confus que possible tout en conservant les mêmes fonctionnalités. Contrairement aux types plus simples de brouillage de code décrits précédemment, le brouillage avancé est plus difficile à casser, surtout s'il est combiné à d'autres techniques de durcissement sophistiquées.

Brouiller le flux de contrôle

Le brouillage du flux de contrôle modifie la structure de base d'appel des sous-programmes afin de rendre le code plus difficile à tracer. Il empêche les décompilateurs de reconstituer le code source en supprimant les modèles qu'ils recherchent. Par exemple, des fonctions peuvent être intégrées (en remplaçant des appels de méthode par le corps de méthode réel), des appels de sous-routines peuvent être remplacés par des sauts calculés et des constructions conditionnelles des arborescentes peuvent être converties en instructions de commutation plates. Cette dernière méthode, l'aplatissement du flux de contrôle, utilise un répartiteur pour contrôler le flux au lieu d'appeler des routines directement à partir d'autres routines, comme montré dans l'illustration 1.

Brouiller des appels de messages et des métadonnées via Objective-C

En Objective-C, les messages envoyés aux instances d'objets ne sont résolus qu'à l'exécution, ce qui signifie que les appels de messages sont stockés dans le code binaire sous une forme brute. Les pirates peuvent s'en servir comme vecteur d'attaque pour manipuler la logique d'exécution. Le code Objective-C peut être protégé en brouillant les appels de messages en texte brut contenus dans le code source afin de les rendre difficilement lisibles et modifiables. Le chiffrement des métadonnées Objective-C, notamment les noms de classes, de méthodes, de protocoles, ainsi que les arguments des méthodes et leurs types, est également important pour dissimuler ces informations utiles aux programmes d'analyse statique. Les données chiffrées sont uniquement déchiffrées à l'exécution lors du chargement de l'application brouillée.

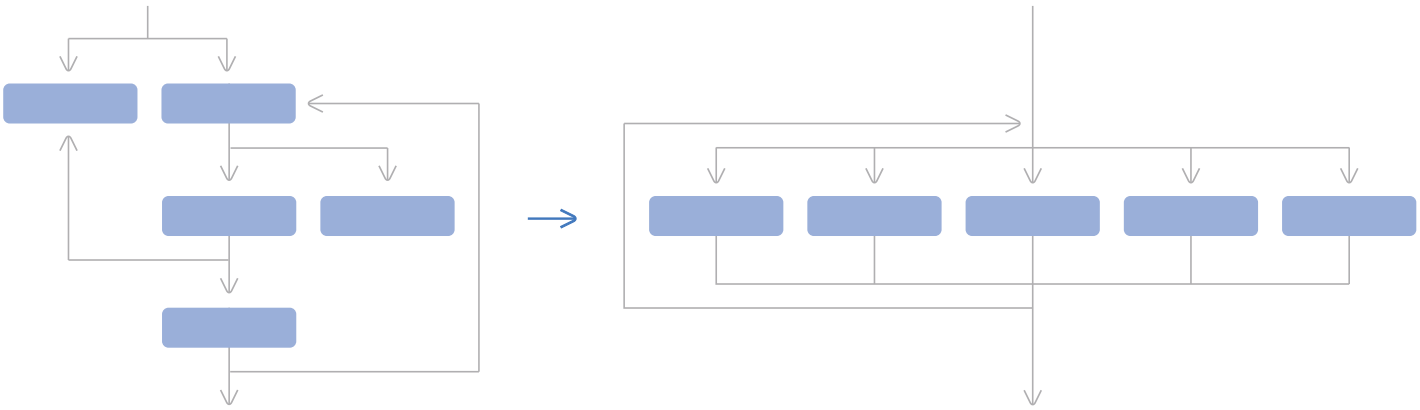


Illustration 1.

Aplatir le flux de contrôle pour empêcher la rétro-ingénierie

3 Anti-débogage

Les débogueurs sont utilisés par les ingénieurs logiciels légitimes pour trouver des erreurs de codage mais, entre les mains des pirates, ils constituent un outil puissant de rétro-ingénierie du code. Ils vérifient l'état d'une application, en extrayant des informations telles que les fonctions appelées, les valeurs de variables et les données contenues dans des emplacements de mémoire arbitraires. En général, ils agissent en fixant des points d'arrêt dans l'application - lorsque ce point est atteint, le programme arrête l'exécution et l'utilisateur inspecte l'état à ce moment-là. Certains débogueurs effectuent également le désassemblage et la décompilation.

Une méthode anti-débogage très basique consiste à insérer des appels API pour interroger les informations sur le processus et le système afin de vérifier l'existence ou le fonctionnement d'un débogueur. Par exemple, IsDebuggerPresent, CheckRemoteDebuggerPresent, ou en utilisant le détachement du débogueur et les contrôles CloseHandle. Ils sont assez faciles à ajouter au code d'une application existante. Cependant, un pirate informatique peut également les contourner assez facilement.

L'une des méthodes les plus efficaces pour empêcher le débogage, appelée « anti-débogage de code modifié », consiste à insérer du code dans l'application qui va détecter la présence d'un débogueur et prendra les mesures défensives appropriées lorsqu'il est présent. En général, il analyse le processus de l'application, en le comparant à la norme de référence pour découvrir des points d'arrêt insérés par le débogueur. Les appels système du noyau peuvent être utilisés pour contourner les hameçons en mode utilisateur insérés par des pirates, ce qui les oblige à modifier le noyau, augmentant ainsi les compétences et les efforts requis pour une attaque réussie.

4 Paquet binaire

Les compacteurs chiffrent et compressent le code, en ajoutant un élément qui le décompose à l'exécution. Les pirates utilisent cette astuce pour masquer les logiciels malveillants aux antivirus, car le code compacté ne contient pas de structure identifiable par la détection.

L'empaquetage du code d'application compilé complique l'ingénierie inverse pour les pirates, car ils ne peuvent pas exécuter le code à l'aide d'un désassembleur ou d'un décompilateur. Au lieu de cela, ils doivent capturer le code après son déchiffrement en mémoire, et seulement ensuite le reconstituer par rétro-ingénierie. Les analyses statiques sont donc bloquées, ce qui oblige le pirate à appliquer des techniques d'analyse ¬dynamique plus complexes et plus longues.

5 Diversification

Une fois que les pirates ont réussi à casser une instance d'application, ils peuvent potentiellement créer un programme automatisé pour casser ses autres instances. Ce phénomène est connu comme une rupture de classe ou un problème de « break once, run everywhere » (BORE).

La diversification modifie le code pour créer des instances du même logiciel au fonctionnement identique, mais avec une surface différente du code uniquement dans sa forme et sa structure.

Cela déjoue efficacement les tentatives d'exploitation d'informations obtenues par un pirate, à partir d'un déploiement pour en compromettre d'autres. Le développement d'un schéma de craquage universel est beaucoup plus difficile pour des instances logicielles diversifiées. Au lieu de cela, chaque instance logicielle doit être individuellement craquée.

La diversification peut être effectuée sur une même catégorie d'applications, chaque instance d'application pouvant ainsi être différente. Mais cette technique est aussi utile pour diversifier les versions du code. Par exemple, si un pirate réussit une rétro-ingénierie sur la version 2.3 d'une application, il devra tout recommencer sur la version suivante, la 2.4.

N'oubliez pas de protéger les clés cryptographiques

Une partie essentielle du blindage des applications consiste à protéger les clés cryptographiques. Trop souvent, les clés sont codées en dur dans l'application, donc facilement extractibles par les pirates, ou bien elles sont exposées en mémoire en cours d'utilisation dans des opérations cryptographiques. Dès que vos clés sont compromises, vous n'avez plus de chiffrement.

Sécurité matérielle

Les protections matérielles, telles que les modules matériels de sécurité (HSM), les modules de plateforme de confiance (TPM) et les environnements d'exécution de confiance (TEE), peuvent être très efficaces pour les clés cryptographiques, mais elles sont complexes à mettre en œuvre sur toutes les plateformes d'appareils et peuvent devenir vulnérables si le propriétaire a obtenu des privilèges d'accès racine au périphérique, par exemple sur un téléphone débridé. Elles sont également sensibles à certains types d'attaques.

Magasins de clés (keystores)

De nombreuses plateformes et systèmes d'exploitation proposent des fichiers keystore (magasins de clés) pour stocker et utiliser en toute sécurité des clés cryptographiques (Android Keystore, Java Keystore, Apple Secure Enclave, Windows Keystore). Ils devraient être suffisants pour les applications sans informations de grande valeur ou sensibles. Toutefois, les algorithmes et les opérations cryptographiques pris en charge sont souvent limités et les opérations cryptographiques doivent être réinitialisées sur chaque plateforme.

Cryptographie en boîte blanche

La cryptographie en boîte blanche, permet de masquer systématiquement les clés, qu'elles soient utilisées, en transit ou stockées. La cryptographie en boîte blanche n'est généralement pas considérée comme étant aussi sécurisée que le matériel de sécurité spécialement conçu, car les calculs sont plus lents. Cependant, les algorithmes logiciels en boîte blanche peuvent être déployés sur des appareils qui ne bénéficient pas de prise en charge matérielle et ils fonctionnent de manière uniforme sur toutes les plateformes. De plus, les clés restent protégées même si un adversaire obtient un accès racine à l'appareil. Certaines bibliothèques de cryptographie en boîte blanche offrent une excellente protection contre les attaques par canal latéral.

6 Protections anti-violation

Les attaques visent souvent à modifier le code de votre application pour la détourner aux fins des pirates. Ils peuvent installer un rootkit et une porte dérobée, désactiver les contrôles de sécurité, contourner l'authentification et injecter un code malveillant qui enregistre les frappes au clavier, vole des données, augmente les privilèges des utilisateurs ou effectue d'autres actions malveillantes. Les protections anti-violation, également appelées autoprotection des applications d'exécution (RASP), détectent et empêchent les attaques qui altèrent votre application.

Contrôle d'intégrité

Le contrôle d'intégrité renforce les applications en insérant des milliers de petits fragments de code qui se chevauchent, appelés contrôleurs, comme le montre l'illustration 2. Pendant l'exécution, chacun de ces vérificateurs vérifie si un segment particulier de l'exécutable a été altéré. En cas de violation, des actions peuvent être déclenchées pour protéger l'intégrité de l'application, par exemple en avertissant l'utilisateur, en appelant une fonction de réponse personnalisée, en générant un message de journal, voire en arrêtant le programme.

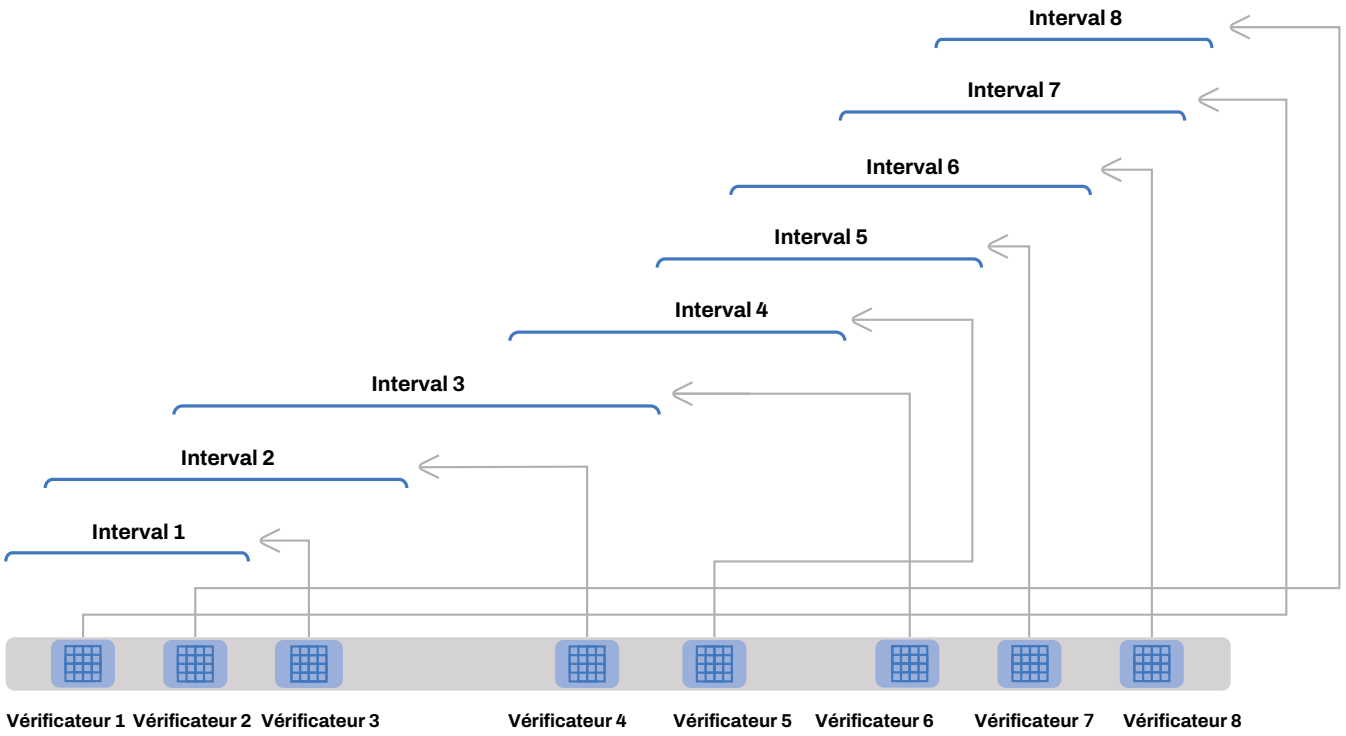
Par exemple, dans l'illustration 2, le vérificateur 1 effectue une somme de contrôle de l'intervalle 7. Si un pirate essaie de modifier le vérificateur 1, les vérificateurs 3 et 4 détecteront le changement effectué.

Swizzling anti-méthodes

Le swizzling de méthode est une fonctionnalité du langage Objective-C - couramment utilisée sur les plateformes Apple - qui a été cooptée par des pirates pour attaquer et modifier le comportement des applications. Le swizzling de méthode modifie le fichier exécutable en cartographiant un nom de méthode de classe sur une implémentation de méthode différente, changeant ainsi son comportement au moment de l'exécution. Il est légitimement utilisé pour remplacer ou étendre des méthodes de classes dans les binaires dont le code source n'est pas disponible. Entre les mains d'un pirate, il peut être utilisé pour rediriger le stockage des informations relatives aux cartes de crédit vers un autre serveur, extraire des informations d'identification, capturer des informations sur les clients ou servir d'autres mauvaises intentions.

Pour minimiser les manipulations de méthodes, travaillez principalement en C++ et utilisez le moins possible les classes ObjC. Les solutions de durcissement des applications, telles que zShield de Zimperium, comportent souvent des mécanismes de détection de « swizzling » et d'exécution d'actions défensives.

Illustration 1.



Détection de débridage d'iOS et d'enracinement d'Android

Débrider un appareil iOS consiste à supprimer les limitations logicielles et matérielles établies par Apple en obtenant un accès racine à l'appareil. L'enracinement donne également un accès privilégié au système d'exploitation sur les appareils Android, en passant outre les limitations définies par les opérateurs et les fabricants de matériel. Dès qu'un appareil a été débridé ou enraciné, les contrôles de sécurité installés sont violés et une application malveillante peut accéder à votre application, à ses données, à ses informations d'identification et à ses clés. Le durcissement des applications mobiles consiste essentiellement à permettre à votre application de détecter un appareil débridé ou enraciné et de prendre des mesures défensives appropriées.

Détections de tentatives de violation supplémentaires

Plus vous détectez de méthodes de violation, plus votre durcissement est solide. D'après les pratiques conseillées, votre application doit intégrer des mécanismes de détection et de réponse aux attaques spécifiques pertinentes. Vous trouverez ci-dessous quelques protections couramment utilisées.

Vérification de l'appelant de la fonction

Un fichier d'application exécutable contient plusieurs fonctions. Généralement, une logique prédéfinie permet de connaître le mode et le moment de l'appel de ces fonctions lors de l'exécution. Toutefois, un pirate habile peut analyser le code binaire, trouver des vulnérabilités dans la logique d'exécution et modifier le flux original du programme en appelant certaines fonctions de manière inattendue, par exemple sous Windows, en utilisant l'injection de DLL.

Pour éviter une telle manipulation des appels de fonctions, créez une liste blanche de modules (fichiers *.dll ou *.exe) autorisés à appeler certaines fonctions sensibles du code de l'application. Stockez les signatures de ces modules autorisés dans le fichier binaire de l'application et utilisez-les lors de l'exécution pour vérifier les modules appelant des fonctions.

Vérification croisée des bibliothèques partagées

L'une des stratégies d'attaque utilisées par les pirates consiste à remplacer ou à modifier les bibliothèques partagées appelées par une application. Pour réduire le risque de ce type d'attaque, limitez autant que possible l'utilisation des bibliothèques partagées. Mettez en place des vérifications croisées afin de détecter la violation d'une bibliothèque partagée utilisée par votre application. Par exemple, vous pouvez créer des signatures cryptographiques pour chaque bibliothèque et vérifier de façon aléatoire, pendant l'exécution, si elles correspondent.

Vérification de signature binaire Mach-O

Toutes les applications macOS, iOS et tvOS distribuées via l'App Store sont signées avec la clé privée d'Apple, ce qui empêche le piratage et la diffusion non autorisée. Toutefois, les membres du programme de développement d'Apple peuvent re-signer une application avec leur propre clé privée incluse dans le certificat de développement, ce qui permet d'exécuter l'application sur les appareils de développement correspondants. Plusieurs services sur Internet re-signent illégalement des applications pour distribuer gratuitement des applications payantes. Vous pouvez insérer des garanties dans votre application pour vous protéger contre la re-signature et la distribution non autorisées d'applications au format de fichier Mach-O (utilisé par les applications macOS, iOS et tvOS).

Protection des licences Google Play

Le piratage d'applications reste une préoccupation majeure pour les développeurs Android. Même la bibliothèque anti-piratage d'Android permettant de vérifier et de faire respecter les licences lors de l'exécution, en étant basée sur Java, peut facilement être piratée. Une protection efficace contre le piratage consiste à remplacer la bibliothèque de vérification des licences de Google Play par une implémentation renforcée, plus difficile à décompiler et à modifier.

7 Actions défensives intégrées à l'application

Dès que votre application identifie une tentative de violation, elle doit déclencher une réponse défensive. Il suffit de fermer l'application. Toutefois, certains types d'attaques présentent moins de risques et ne justifient pas forcément une action aussi extrême. Vous pouvez aussi faire croire au pirate qu'il a réussi et faire en sorte qu'il cesse d'attaquer votre application. Par exemple, si vous protégez un jeu, vous pouvez secrètement corrompre la carte de celui-ci au lieu de planter l'application, afin qu'elle semble craquée pour le pirate alors qu'elle a en réalité été rendue injouable.

Les actions de défense courantes comprennent :

- Bloquer l'accès au compte
- Générer un message de journal à envoyer à l'administrateur
- Arrêt d'exécution des commandes
- Corrompre des éléments de l'application pour qu'un pirate pense avoir réussi, alors qu'il n'a qu'un accès minimal.
- Suppression des données sensibles
- Fermer complètement l'application

Lancer des actions de défense automatiques en temps réel offrant un net avantage par rapport à la génération d'alertes à examiner et a corriger manuellement. Ce type de détection et de réponse complexes nécessite généralement une solution de protection intégrée et avancée.

Protection intégrée avec zShield de Zimperium

zShield de Zimperium injecte des capacités d'autodéfense dans votre application. Il empêche la violation, la rétro-ingénierie et d'autres techniques utilisées par les cybercriminels pour accéder aux informations sensibles et au code de votre application. zShield utilise plusieurs méthodes de défense, notamment l'obscurcissement et l'aplatissement du code, ainsi que la détection des intrusions en temps réel, pour renforcer la sécurité de votre application en quelques minutes.

zShield est l'un des composants de la seule solution unifiée, Mobile Application Protection Suite (MAPS) de Zimperium, qui combine une protection intégrée complète à une visibilité centralisée des menaces. Elle comprend également :

zKeyBox : cet outil de cryptographie en boîte blanche à la pointe de la technologie permet de garder les clés cryptographiques secrètes bien cachées dans le code de l'application, même lors de l'exécution.

zDefend : cette solution de protection avancée intégrée à l'application permet aux applications mobiles de déterminer immédiatement le moment où l'appareil d'un utilisateur est compromis, l'existence d'attaques réseau et même l'installation d'applications malveillantes. Les développeurs d'applications peuvent configurer des actions correctives appropriées lorsqu'une menace donnée est détectée.

zScan : cette solution de test des applications mobiles identifie les risques liés à la confidentialité, à la sécurité et à la conformité au cours du processus de développement, avant leur sortie.

Pour en savoir plus, consultez le site zimperium.com.