SecNeo AppShield Technology

Taking mobile application security to new levels

Table of Contents

1 Overview

1.1 Security Threats to Android Applications

With the rapid development in the mobile Internet industry, and the immense popularity of mobile applications, the number of Android application developers has increased tenfold to hundreds of thousands, and there are millions of applications available now. The number of new application development has also experienced a geometric rise every day. As at 2015, the number of Android applications Android has reached 1.4 million in total and the cumulative number of Android applications that have been downloaded and installed has exceeded 65 billion.

At the same time, due to the open source nature of the Android system, Android applications are gradually replacing PCs as the main target of attacks by hackers. More than 97% of Android applications have been pirated, and infected with Trojans and viruses. Rogue software and phishing applications are also raging everywhere. These severely affect the revenue for developers and compromise the security at the client-end and user experience.

1.2 Security Requirements for Android Applications

- Android applications are written in the Java language with limited scope for obfuscation;
- The applications can be easily decompiled to decipher the core service logic and algorithm;
- The applications can also be easily repackaged with malicious code such as viruses, trojans, and rogue advertisements inserted into the applications;
- Android devices generally face severe root and dynamic injection threats;
- There are more and more varied attacks that target Android applications;
- Android applications have penetrated into all aspects of life.

1.2 Android Applications Security Protection

What kind of protection does the application need?

- The Java code needs to be shielded and encrypted;
- Prevents debugging and dump for memory data;
- Protects configuration parameters from being tampered;
- Prevents resource files from being added or replaced;
- Prevents shared objects library files from being decompiled or replicated for use in other applications;
- Data files and certificate files need to be encrypted to prevent theft.

2 AppShield Technologies

2.1 Anti-Decompiler (1st Generation)

The Android platform is developed with the Java language which can be easily decompiled. At the same time, the decompiled code is often very close to the source code, and very easy to read, such that all the client and server logic will be revealed like the communication method deployed at the server-end, the encryption and decryption algorithms and crypto-keys, account transfers and service processes, and how the software keyboard is implemented. Hence, there is a need to encrypt and protect the Java code.

SecNeo encryption technology offers protection in the following attack situations:

➢ **Protects main code from reverse engineering**

With the source code encryption technology in AppShield, the user can be assured that the attacker will not be able to reverse engineer and analyze the code used to implement the core services. Based on the analysis SecNeo did on mobile clients' susceptibility to security attacks, source code analysis is often the essential first step for many malicious attacks. By reverse engineering the installation packages, the attacker can carry out dynamic injections or hijacking attacks with precision. When all the source code at the client's end has been encrypted for protection, the threshold against attacks can increase significantly, and the first line of defense for the mobile client can be established.

➢ **No bypassing the client's existing security logic**

Mobile clients often implement security features like signature verification and certificate validation, or embed some secure SDK (such as a soft keyboard component). With source code encryption, the attacker cannot lock onto the main code, and cannot tamper with the code to bypass the security logic.

➢ **No breach on embedded encryption algorithm and crypto-key**

With regards to the symmetric encryption logic implemented locally in the application, source code encryption can protect the encryption algorithm and the embedded crypto-key to make sure that the attacker cannot obtain such information from reverse engineering.

2.1.1 Technology Overview

The Android platform uses Java as a native language for development. In the final installation package, all the Java code is compiled and packaged into the classes.dex file in the APK, and this classes.dex is the protection target for Java code.

The first generation encryption technology in AppShield is implemented with the class loading method, and the basic principle is to perform encryption and shielding on the classes.dex file, and store that within the APK resources. The encrypted classes.dex file will be decrypted in the memory at runtime, where the Dalvik virtual machine will dynamically load and execute the file.

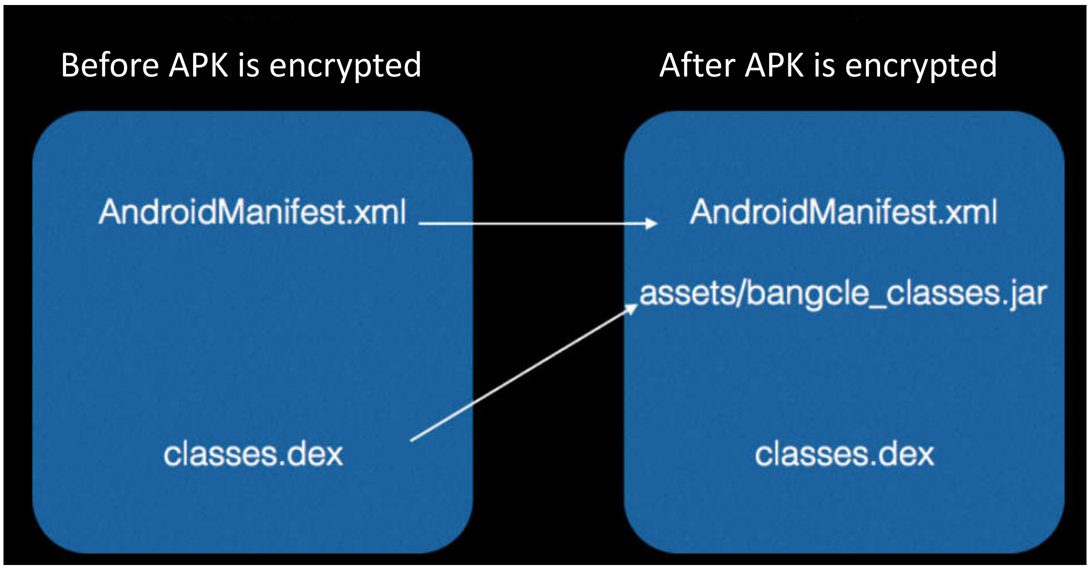


Figure 1: Comparison before & after the first generation encryption technology applied

The execution process for the first generation encryption technology is as follows:

1) The classes.dex file in the original installation package is extracted and encrypted during packing;
2) The encrypted classes.dex file is stored in the resources;
3) The program entry point for the original installation package is modified to make sure that non-shielded code will be executed first;
4) At runtime, the non-shielded code will first load the encrypted classes.dex file from the resources into the memory;
5) This file will be decrypted in the memory, and the decrypted code will then be dynamically loaded using the Dalvik class loading mechanism;
6) Once loading is completed, the control of the program is handed over to the decrypted code, and the decrypted code will start to execute the relevant operations.

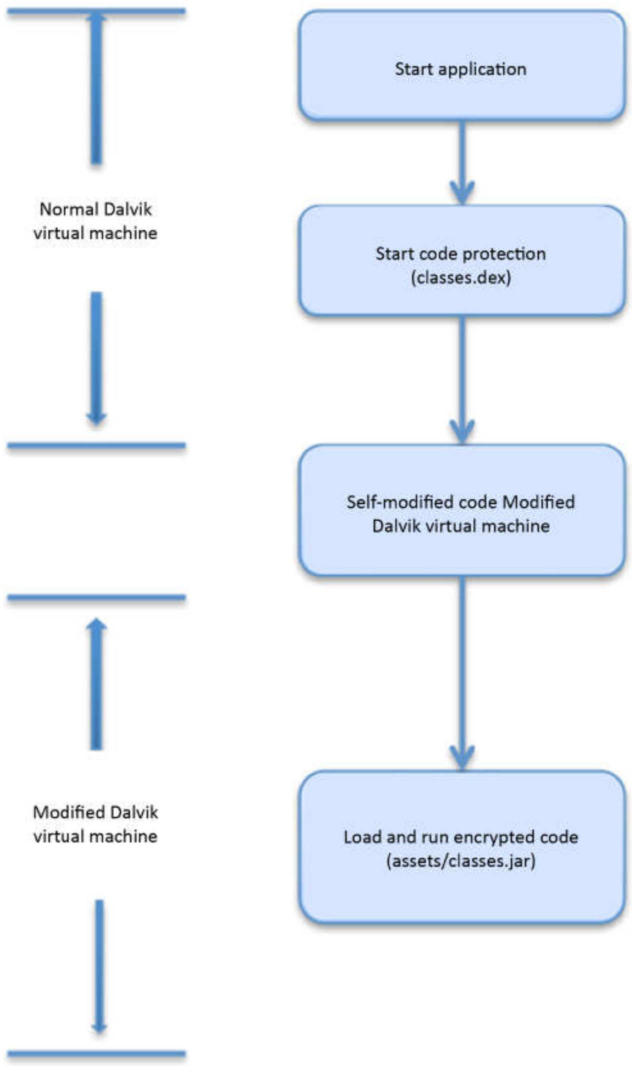

Normal Dalvik virtual machine
Modified Dalvik virtual machine
Start application
Start code protection (classes.dex)
Self-modified code Modified Dalvik virtual machine
Load and run encrypted code (assets/classes.jar)

Figure 2: Execution process for the first generation encryption technology

2.1.2 Solution Advantages

- The entire dex is encrypted and shielded, and this can effectively protect against all kinds of decompilers
- Protection targets include Java code, HTML, JavaScript scripts and others
- Effectively protects APK from static decompilers.

2.2 Anti-Decompiler (3rd Generation)

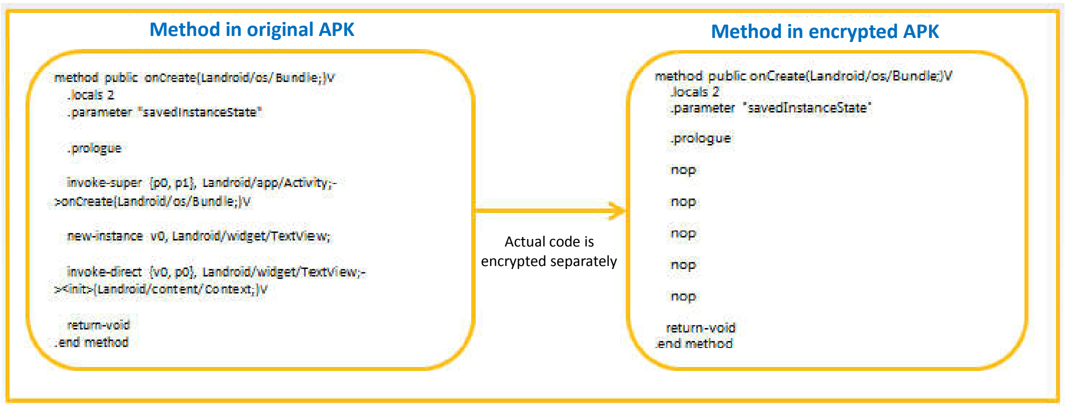
As the first generation of encryption technology carries out the encryption and decryption on the entire classes.dex file, the completed decrypted code is contiguously stored inside the memory at runtime, which means that the attacker can modify the Dalvik virtual machine and access the memory dump to obtain the decrypted code. Although we can patch to increase the difficulty in deciphering the data, for example, erase or obfuscate information in the dex file header or footer once the class loading is completed, these methods are palliative measures and cannot solve the fundamental issue related to the memory dump.

Because the first generation encryption technology cannot deal with the threat of dynamic analysis, it cannot prevent the attacker from using the memory dump to crack the APK, so there is an urgent need to prevent dynamic analysis of the APK with regards to decompilers.

2.2.1 Technology Overview

The third generation encryption technology from AppShield improves on the weakness related to memory dump when compared to the first generation. It uses a replacement method to implement the security enhancement that significantly increases the application security.

Its basic principle is to use the Java virtual machine execution mechanism to dynamically decrypt the specific method only just before the method is to be executed, and that method will not be stored contiguously in the memory, while other methods which are not executed will not be decrypted into the memory. This encryption method stop attackers from extracting the source code in the dex files stored in the memory dump as every method in the code is individually encrypted and only decrypted when the method is required for execution.



```
method public onCreate(Landroid/os/Bundle;)V
  .locals 2
  .parameter "savedInstanceState"

  .prologue

  invoke-super {p0, p1}, Landroid/app/Activity;-
>onCreate(Landroid/os/Bundle;)V

  new-instance v0, Landroid/widget/TextView;

  invoke-direct {v0, p0}, Landroid/widget/TextView;-
><init>(Landroid/content/Context;)V

  return-void
.end method
```

```
method public onCreate(Landroid/os/Bundle;)V
  .locals 2
  .parameter "savedInstanceState"

  .prologue

  nop

  nop

  nop

  nop

  nop

  return-void
.end method
```

Figure 3: Comparison before & after the 3rd generation encryption technology applied

The execution process for the DEX function extraction and encryption technology is as follows:

1) The code for the main method in the original APK is extracted, and individually encrypted to a single file during packing;
2) At runtime, the encryption engine will dynamically modify the Dalvik virtual machine for the current process, and initialize the virtual machine adaptation layer;
3) The encryption engine will decrypt the code only when the Dalvik virtual machine executes the specific method, and the decrypted code will be handed over to the virtual machine to be executed by the engine.

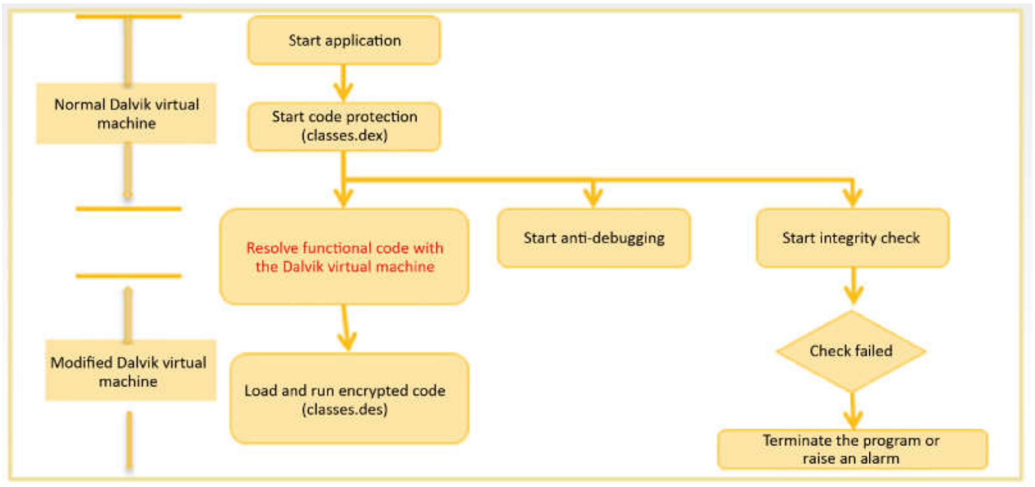

Normal Dalvik virtual machine
Modified Dalvik virtual machine
Start application
Start code protection (classes.dex)
Resolve functional code with the Dalvik virtual machine
Load and run encrypted code (classes.des)
Start anti-debugging
Start integrity check
Check failed
Terminate the program or raise an alarm

Figure 4: Execution process for the third generation encryption technology

2.2.2 Solution Advantages

- The encryption granularity is reduced, which is an improvement from encryption at the file level to encryption at the method and function levels
- On-demand encryption that can be flexibly configured and adapted to suit customers' needs and the unique characteristics of each APK
- Dynamic decryption where the method is decrypted only when it is required, and there is no way the entire source code can be retrieved from the memory dump
- The memory is not stored contiguously, as the decrypted code is divided and dispersed to be stored in separate memory segments to make sure that the code is secure after decryption.

2.3 Anti-Decompiler (5th Generation)

Traditional code protection technology mainly involves two techniques: code obfuscation and shielding. Code obfuscation refers to translating the computer program code into a functionally equivalent but difficult to read and understand format. Code obfuscation does not really prevent reverse engineering, but it increases the difficulty of understanding the code. Shielding is another software protection technology that is very common. The so-called "shield" is a layer of code used to wrap around the program code. This layer of code is executed before the protected code. It will execute the decrypted code and anti-debugging operations, and only hand over the execution rights to the target code after all these operations have been completed. Shielding can prevent static analysis effectively, but not dynamic analysis. This is because the final decrypted code will eventually be executed in memory, and once the attacker finds the decrypted code address in the memory, it is not difficult to remove the shield.

Basically speaking, the first generation of dex overall shielding technology and the third generation dex class extraction and encryption technology are similar to the encryption technology used in PCs, which is to hide the code, but the final code still have to be executed with the Dalvik/ART virtual machine. Therefore, to decipher the code, the attacker can build a modified virtual machine to remove the shield on the protected solution. We need a more powerful and secure encryption technology to make sure that the APK is secure.

2.3.1 Technology Overview

SecNeo fifth generation encryption technology (VMP) is based on the virtual system protection technology. Virtual protection technology uses a brand-new language to translate the original code, where only customized virtual machine engines by SecNeo can understand this language and without any available reference, it is extremely challenging for hackers to master this language. Therefore, the software protection method based on virtual machine is recognized in the industry as the ultimate protection method that is very difficult to crack. It not only increases the difficulty in reverse engineering, but it also greatly increases the difficulty in restoring the source code.

SecNeo VMP technology converts the original executable code into a secure bytecode customized by SecNeo and the customized bytecode can only be interpreted and executed by a SecNeo customized virtual machine.

The unshielded methods in SecNeo VMP technology can continue to be executed by a Dalvik virtual machine as normal Dalvik bytecode, while the shielded methods will be executed by a customized virtual machine. A hacker who is trying to crack the system, even if the hacker manages to obtain the customized bytecode, the hackers still needs to analyze and understand the customized bytecode format, hence the hacker needs to spend more time to reverse engineer the customized virtual machine interpretation engine. At the same time, the VMP technology uses virtualization to create the shield, and many different virtual interpretation engines can be built, where different methods can use different virtual execution engines. With this methodology, the security level is further elevated.

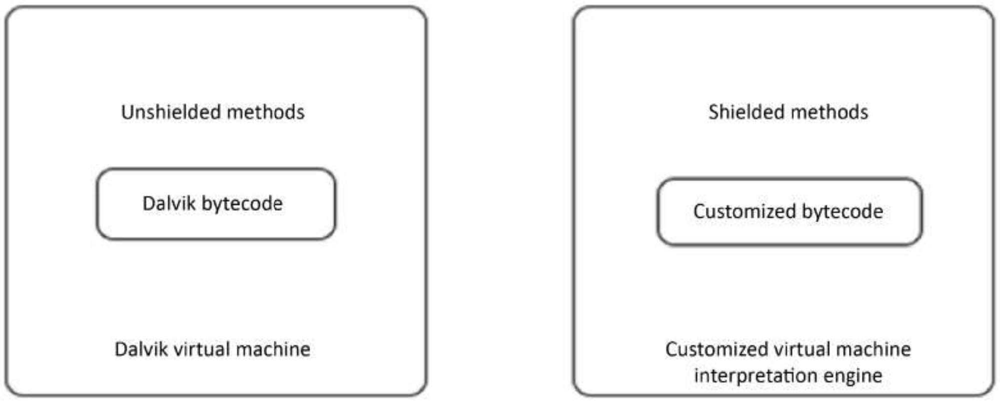


Figure 5: Fifth generation encryption technology schematic

The execution process for the VMP encryption technology is as follows:

1) The Java function that is to be shielded is converted into a native function;
2) The bytecode in the shielded Java function is randomly converted by the virtual instructions into a new bytecode format;
3) To implement the native function related to this function, the purpose of this function is to read the new bytecode that have been converted by the virtual instructions in order to interpret and execute these bytecode.

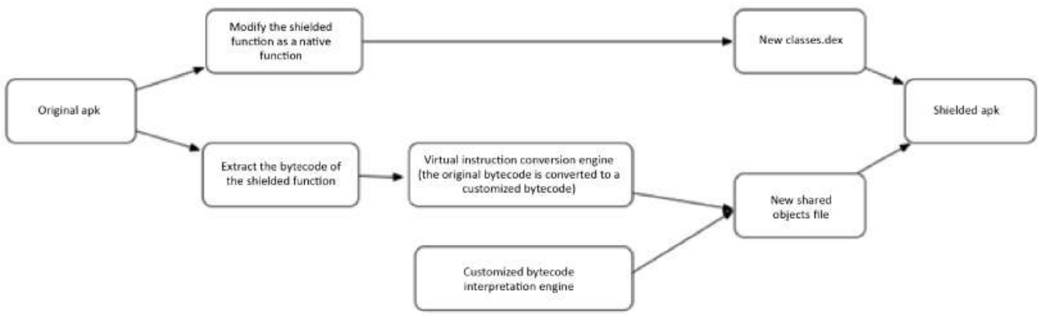

Original apk
Modify the shielded function as a native function
Extract the bytecode of the shielded function
Virtual instruction conversion engine (the original bytecode is converted to a customized bytecode)
Customized bytecode interpretation engine
New classes.dex
New shared objects file
Shielded apk

Figure 6: Execution process for the fifth generation encryption technology

2.3.2 Solution Advantages

- Completely alters the presentation format of the APK code, and converts the assembly code into bytecode;
- Customized virtual machine instructions make it impossible for any attacker to execute;
- Virtual instruction conversion is randomized, even the author cannot remove the shield easily;
- Backward compatibility to ensure that future Android versions continue to be compatible.

2.4 HTML Development Framework Encryption

With the development of mobile technologies, frameworks that support application development using HTML, JavaScript and others have surfaced. Examples include PhoneGap and IBM Worklight. This type of development frameworks solve the portability issue in mobile application development and improve the development efficiency.

However, these frameworks also create more severe security issues, script files for HTML, JavaScript and others require the local data to be stored as plaintext in the resource files of the package or in the local data at runtime, hence the attacker can directly obtain the source code and does not even need to perform any reverse-engineering. Therefore, HTML, JavaScript and other files in these development frameworks must be encrypted and protected.

AppShield technology protects script files created by HTML or JavaScript with a combination of resource and transparent data encryption technologies.

2.4.1 Technology Overview

The principle of resource encryption technology is to intercept the function that reads in the resources and automatically decrypt the resources required for the operations. The resource encryption technology helps to ensure that the scripts in the installation package exist in an encrypted format. In frameworks like PhoneGap and IBM Worklight, it is often necessary to release the scripts to the local storage in the phone, and then load the scripts from the local storage. At this point, scripts stored locally have to be stored as plaintext. Therefore, it is necessary to support a transparent data encryption technology such that the scripts on the mobile phone are also encrypted.

The principle of transparent data encryption is to intercept all the I/O operations at runtime, where if the I/O operating target requires the transparent encrypted files, then it will automatically perform the corresponding decryption and encryption operations to support the read-write operations. Specifically, if the I/O operating target requires the transparent encrypted files, then the decryption operation will be carried out automatically during the file reading process, and the decrypted data will be handed over to the current process. If there are write operations in the current process, then the encryption operation will be carried out automatically, and the encrypted data will be written to the phone's local storage. By combining the resource encryption technology with transparent data encryption technology, AppShield is able to ensure that scripts from HTML, JavaScript and the like are stored in encrypted formats either in the package or local storage so as to effectively secure the scripts.

The HTML5 script encryption solution in AppShield includes the following important segments:

1) Encryption for H5 scripts from the server that are stored locally;
2) Dynamic decryption before H5 scripts are loaded to the local client.

AppShield provides an Android HTML5 script encryption solution that makes sure the H5 scripts are secure from the server to the client to runtime, and eliminates the situation where information about the service logic, security policies and communication protocols may be compromised because the H5 scripts are stored as plaintext.

2.4.2 Solution Advantages

- ➢ Transparent encryption that does not affect the user experience;
- ➢ The file encryption can effectively deal with all types of hacking behavior.
- ➢ Compatible with all HTML5 development frameworks

2.5 Shared Objects (SO) Library Encryption

The shared objects library is a dynamic library developed using the C/C++ language. To reverse engineer the shared objects library, the attacker must have a certain foundation in assembly languages, and the difficulty in reverse engineering is higher when compared to Java code decompilation. However, reverse engineering can still be carried out on the shared objects library, so there is a need to protect this library.

Why the shared objects library should be protected:

➢ **Protects the shared objects library code from reverse engineering**

The shared objects encryption technology from SecNeo AppShield will ensure the assembly code of the shared objects library cannot be reverse engineered and analyzed by attackers using disassemblers like IDA.

➢ **No breach on embedded encryption algorithm and crypto-key for the shared objects library**

With regards to the symmetric encryption logic implemented in the shared objects library, shared objects encryption can protect the encryption algorithm and the embedded crypto-key to make sure that the attacker cannot obtain such information from reverse engineering using disassemblers.

2.5.1 Technology Overview

The shared objects encryption technology is similar to the encryption technology used in the PC workstation. Encryption technology refers to the use of special algorithms to modify the executable program files or dynamically linked library files in order to encrypt the program code, and to prevent reverse engineering using disassemblers such as IDA. The main technology principle in SecNeo shared objects library encryption technology:

1) Assembly code compression and encryption

At the code level, the assembly code in the shared objects library is compressed and encrypted to prevent the code from being reverse engineered

2) Protects ELF data information in the shared objects library

The shared objects library is essentially a file in ELF format. In the ELF format, a large number of auxiliary data information will be defined such as dynamic positioning, function address and symbol table. SecNeo shared objects encryption technology uses methods like mapping entries removal to shield and hide the data so that hackers cannot recover the data information, and thus significantly elevates the security level.

3) Decrypted code will be removed dynamically

SecNeo shared objects protection technology also uses a dynamic technique to remove the decrypted code to further enhance the security of the shared objects library. The so-called dynamic removal of decrypted code refers to removing the functional code from the memory when the function has been executed, and the function will be decrypted again for implementation during the next execution. With this technology, the complete decrypted code does not exist in the memory at runtime, and this greatly increases the difficulty of trying to crack the application using the memory dump method.

4) Binds shared objects library with application

The technology principle in SecNeo shared objects library binding technology is to place the decrypted crypto-key of the library in the shared object where the Java code has been encrypted, so that it is separated from the encrypted APK, and this shared object cannot run as it is unable to obtain the crypto-key. In addition, installation for encrypted shared objects is replaced. So when the shared objects in the encrypted Java code tries to decrypt the shared objects library, decryption will fail and the program cannot run.

2.5.2 Solution Advantages

- ➢ Extremely strong and entrenched encryption and encryption to effectively prevent static decompile by all kinds of tools;
- ➢ ELF dynamic removal effectively prevents the debugging of memory data;
- ➢ Binding the application with the shared objects helps to make sure that the shared object files cannot be replicated for use in other applications.

2.6 Local Data Encryption

Mobile clients often need to use local files to store certain configuration information, user information and others. In a non-root situation, other programs are not able to read the data due to the sandbox mechanism inherent in Android. However, in the case of the mobile root, malicious programs can read these files directly to steal the user data. Therefore, this type of local data should be stored in encrypted form.

Under normal circumstances, developers using APIs can only encrypt certain segments of the local files. For example, when SQLite is used for data storage, developers can only encrypt the database domain and not the entire data file format. AppShield local data encryption can, however, encrypt the entire data file. So in SQLite, for example, the entire table structure can be encrypted, and when the SQLite viewing tool is used, the file cannot be opened for viewing. AppShield local data encryption can significantly improve the security of local data storage.

Advantages of AppShield local data encryption:

- **Ubiquitous local transparent data protection**

  With the transparent local file encryption solution from AppShield, encryption can be performed on any local file or database storage so that the local information is effectively protected and the situation where sensitive information may be stored as plaintext due to man-made defects in the development can be eliminated.

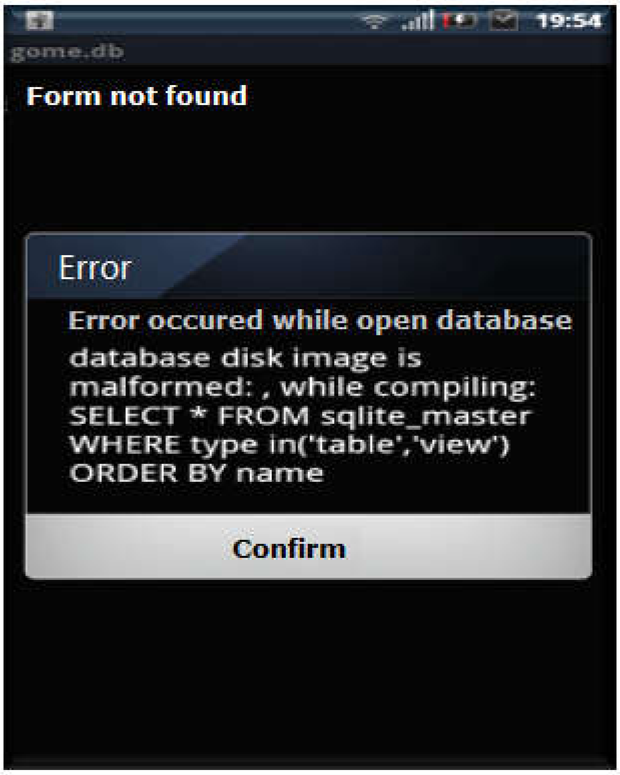
- **Easy, simple, and secure file storage**

  Developers who use AppShield to encrypt transparent local files do not need to worry about writing the code for data encryption and decryption, as the file functions at the system layer will automatically encrypt and decrypt all data, and this makes the development of secure file operations simple and convenient.

2.6.1 Technology Overview

AppShield uses the Android file system layer to implement the transparent data encryption mechanism, and this effectively protects the read and writes operations for all files, including database files and XML configuration files. Local data encryption adopts the "transparent data encryption" technology. In the PC domain, the so-called "transparent encryption technology" is a type of file encryption technology which is designed to satisfy enterprise file security requirements. The so-called transparency means that the process is transparent to the user. When the user opens or edits the specific file, the system will automatically encrypt the unencrypted file and decrypt the encrypted file. The file is stored as ciphertext in the hard disk, and as plaintext in the memory. Once outside its use environment, the application cannot start as the automatic decrypted service is not available, hence the file contents are effectively secured.

Like the transparent data encryption technologies used in PCs, the basic principle of the transparent encryption technology in AppShield is to intercept all the I/O operations at runtime, where if the I/O operating target requires the transparent encrypted files, then it will automatically perform the corresponding decryption and encryption operations to support the read-write operations. Specifically, if the I/O operating target requires the transparent encrypted files, then the decryption operation will be carried out automatically during the file reading process, and the decrypted data will be handed over to the current process. If there are write operations in the current process, then the encryption operation will be carried out automatically, and the encrypted data will be written to the phone's local storage.



Form not found

Error

Error occured while open database database disk image is malformed: , while compiling: SELECT * FROM sqlite_master WHERE type in('table','view') ORDER BY name

Figure 7: Database file with transparent file encryption

The main technology features of the transparent file system encryption solution in AppShield are as follows:

1. Supports data encryption for all file formats
2. Provides optimized encryption for all video, audio and image file formats
3. Supports database encryption
4. Supports data encryption and device binding
5. The encryption and decryption process is transparent to the application user
6. Encryption and decryption process is transparent to developers

2.6.2 Solution Advantages

- ➢ Static encryption and dynamic decryption to make sure that the application data file cannot be maliciously modified or replaced
- ➢ Transparent encryption where the developers do not need to modify the code

- Flexible file encryption solution where the encryption strategy can be customized according to the user's requirements

2.7 App Anti-Tamper Protection

Application tampering refers to acts that add to the code or modify the code, replace the resource files, modify the configuration, change the icons or insert illegal code within the application program, and then repackage the tampered application to generate all kinds of pirated and phishing applications. Especially for finance type of applications, the addition of viral code and advertising SDK may cause sensitive information such as the user's login account, payment password and SMS verification code to be compromised and stolen, as well as criminal behavior like altering the transfer account numbers and transfer amounts.

The attacker can decompile the application using reverse engineering tools, and after adding all kinds of malicious code to the application, repackage and launch the modified application to some applications markets where the audit and verification is not as stringent. These applications pose a major security threat to users, and protecting the applications from being tampered is an imminent issue.

AppShield anti-tamper protection can achieve the following:

- **Lets users use the official app version**

  AppShield uses a comprehensive verification technology to stop any tampered or repackaged application from starting, and this makes sure that the user has installed and is using an application that has been officially verified and launched.

  On the contrary, any application that has been tampered or repackaged will not be able to start because its integrity is questionable.

- **Developers no longer need to worry about securing the integrity of the app**

  Many client developers are concerned about how they can protect the integrity of the application, and the relevant mobile application security standards also mentioned the need to verify the client's signature. Developers often use local signature verification, local hash checksum or other online verification methods to check for the application integrity, but these schemes are flawed, and also add to the development and maintenance costs. With the integrity protection solution in AppShield, the integrity of the client can be basically guaranteed.

2.7.1 Technology Overview

When the client end is encrypted and protected, any modification made to the encrypted installation package, including code, resource and configuration files, and any client that has been tampered with illegally will be detected and the operation terminated at runtime. The technology principle for anti-tampering is based on integrity verification technologies which verify the installation package. Once the verification fails, the client is regarded as an illegal client.

The main technology principle is to perform a cross-validation on all the file contents in the application installation package, as well as encrypt the verified data and code for protection. The official application version can be detected so that the non-official versions cannot start and run.

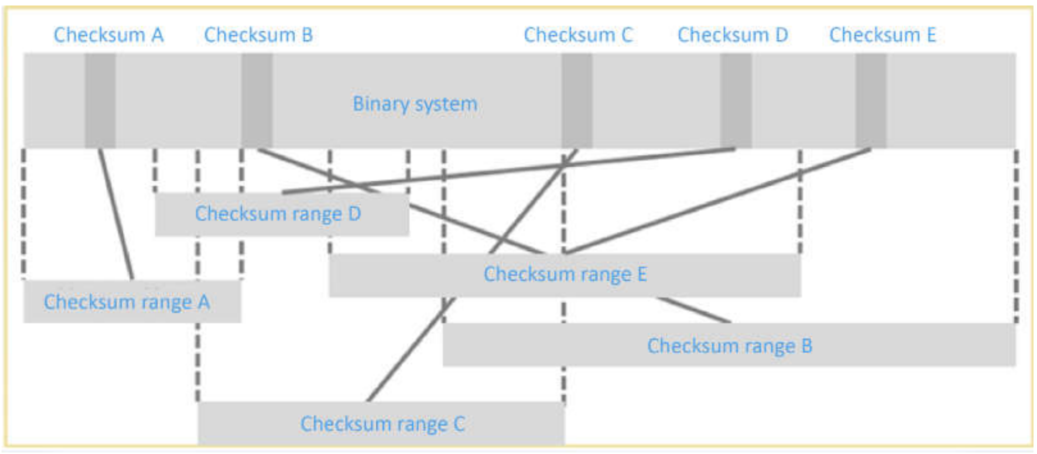


Figure 8: APP integrity cross-validation technology

2.7.2 Solution Advantages

- Full file verification to prohibit the addition of any malicious content to the application;
- Effectively prevents the generation and dissemination of pirated applications;
- Increases developers' revenue;
- Maintains brand image and reputation of application developer.

2.8 App Anti-Debugging Protection

Dynamic debugging attack refers to the use of a debugger to track the target program when it runs, view and modify the code and data in the memory, analyze the program logic, carry out an attack and cracking the code. Take a financial sector application as an example, when the application service is operating, dynamic debugging can modify data like the account number and amount.

As the Android platform does not prohibit the use of ptrace for system debugging, malicious programs that have gained access to the root permissions can use this API to modify the memory and registry of the target process in order to execute the shellcode to inject malicious module. Once the malicious code is injected, the malicious code can dynamically access the various sensitive information in the memory, such as the username and password.

Besides the ptrace system function call, the proc file system in the Android platform not only reveals a lot of process information, but it can also be used to implement read and write operations to the memory. Therefore, it is necessary for the program to be protected against debugging.

SecNeo anti-debugging technology offers protection in the following attack scenarios:

- **Prevents theft of account information**

  Protects key information such as account number and password entered by the user by encrypting the key information inputs and memory transfers. This stops the attacker from stealing account information with methods like code injection into memory.

- **Protects transaction from being hijacked**

  Protects the integrity of the transaction information by monitoring the key functions that handle such transactions to prevent any intermediate attacks that may be a result of injection or hijacking.

- **Protects memory information from being read**

  Carries out strict access controls to information in the memory using anti-debugging technology in order to protect sensitive information from being read when the information is temporarily stored in the memory. At the same time, dynamic debugging and analysis of the program data at runtime is terminated.

2.8.1 Technology Overview

The dynamic defense technology in SecNeo AppShield is based on anti-debugging protection. At the same time, it monitors key functions and linkages, and uses methods like polling and proactive detection to shield the mobile app from attacks at runtime.

SecNeo AppShield anti-debugging technologies include:

1) Terminated ptrace

   Uses a bidirectional ptrace protection technology to prevent other processes from performing a ptrace operation on the current process

2) Process debug status check

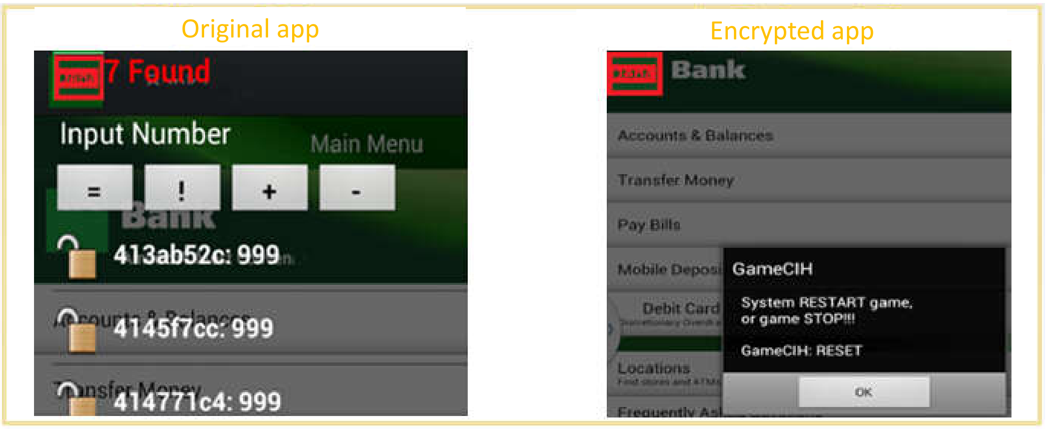
   Uses polling to check if the process status is in the debugging state

3) Signal processing mechanism

   Uses the Linux signal mechanism to check if the process status is in the debugging state

4) Monitoring of the proc file system

   Uses the inotify interface to monitor the proc file system to check if the current process is using the proc file system to perform read and write operations to the memory



GameCIH

System RESTART game, or game STOP!!!

GameCIH: RESET

Figure 9: Comparison of memory access control in app before and after encryption

The main technical characteristics of the dynamic defense feature in SecNeo AppShield:

1. Uses proactive detection mechanism like ptrace to prevent the attacker from debugging the application
2. Uses polling to perform anti-hook check on key functions
3. Monitors the processes and key features of mainstream injection tools to prevent dynamic injection attacks
4. Replaces and hides the key service logic functions to prevent them from being hooked

2.8.2 Solution Advantages

- ➢ Effectively prevents various debuggers from performing any dynamic debugging on applications;
- ➢ System overhead is small with no impact on the application and user experience.

3 Value Proposition

The main goal for SecNeo AppShield is to help customers to improve the security and protection capabilities of the customers' applications in all aspects.

The values that SecNeo AppShield brings:

- ✓ Protects applications from static analysis and dynamic debugging;
- ✓ Secures sensitive data stored in the applications;
- ✓ Prevents applications from process injection;
- ✓ Prevents applications from being repackaged;
- ✓ Protects intellectual property rights;
- ✓ Protects developers' revenue;
- ✓ Protects brand image and reputation.

4 System Support

- ✓ Supports ART, Dalvik modes in Android
- ✓ Supports Android's latest operating system, Android 7.0 Nougat (Android N)
- ✓ Supports Android 2.1 and higher operating systems
- ✓ Compatible with ARM-32 bit CPU
- ✓ Compatible with ARM-64 bit CPU
- ✓ Compatible with x86-32 bit CPU
- ✓ Compatible with x86-64 bit CPU
- ✓ Compatible with customized ROMs from major mobile phone manufacturers.