

## 코드 리뷰 보고서

리뷰번호	010	작성자	곽채원	작성날짜	2023.06.03
관련규칙	49. Remove short-lived objects from long-lived container objects.				
중요도	상				
	Before				
대상코드	<pre style="background-color: #f0f0f0; padding: 10px;"> try(FileOutputStream fos = new FileOutputStream(ename + "_key.txt")) {     byte[] encrypted = c.doFinal(secretKey.getEncoded());     fos.write(encrypted);     System.out.println("The secretKey is successfully encrypted (using received key)");     return true; } </pre>				
	After				
보안약점/ 보안취약점 이유 및 해결법	<pre style="background-color: #f0f0f0; padding: 10px;"> try(FileOutputStream fos = new FileOutputStream(ename + "_key.txt")) {     byte[] encrypted = c.doFinal(secretKey.getEncoded());     fos.write(encrypted);     // 사용한 secretKey는 사용 후 지워주기 (가비지 컬렉터가 빨리 회수하게끔 null로)     secretKey = null;     System.out.println("The secretKey is successfully encrypted (using received key)");     return true; } </pre>				
	<p>secretKey와 같은 민감 정보를 담은 객체가 사용을 마친 후에도 계속 살아있는 경우 공격자에 의해 악용될 가능성이 크다.</p> <p>이를 해결하기 위해서는 사용 후에 바로 null로 처리해주면 된다. 이렇게 하면 secretKey 내용을 없앨 수도 있고, 객체 자체를 가비지 컬렉터에게 사용하지 않는다고 알려주기 때문에 null로 처리해주었다.</p>				

## 코드 리뷰 보고서

리뷰번호	011	작성자	곽채원	작성날짜	2023.06.03
관련규칙	22. Minimize the scope of variables.				
중요도	상				
	Before				
대상코드	<pre style="border: 1px solid black; padding: 10px;"> <b>static boolean vrfy(String rpkey, String Pkey, String ename) {</b>     Cipher c;     <b>try {</b>         // secretKey 해독에는 대칭암호 : RSA 알고리즘 사용         c = Cipher.getInstance(keyAlgorithm);         <b>if (c != null) {</b>             c.init(Cipher.DECRYPT_MODE, kmanager.readKey(rpkey));              // key 읽어오기             Key secretKey = <b>null</b>; </pre>				
	After				
보안약점/ 보안취약점 이유 및 해결법	<pre style="border: 1px solid black; padding: 10px;"> <b>static boolean vrfy(String rpkey, String Pkey, String ename) {</b>     <b>try {</b>         // secretKey 해독에는 대칭암호 : RSA 알고리즘 사용         Cipher c = Cipher.getInstance(keyAlgorithm);         <b>if (c != null) {</b>             c.init(Cipher.DECRYPT_MODE, kmanager.readKey(rpkey)); </pre>				
	<p>Before 코드에서는 함수 반환 값에 대해 고려하려다 Cipher 객체에 대한 선언을 try문 밖에 해주었다. 이럴 경우 Cipher 객체 c는 try문 안의 동작이 끝난 뒤에도, 즉 오류가 발생해서 try문을 빠져나온 뒤에도 공격자가 접근할 수 있게 된다.</p> <p>이를 해결하기 위해 try문 안에 Cipher 객체를 선언하여 변수의 영역범위를 최소화하였다.</p>				

## 코드 리뷰 보고서

리뷰번호	012	작성자	곽채원	작성날짜	2023.06.03
관련규칙	26. Always provide feedback about the resulting value of a method.				
중요도	중				
	Before				
대상코드	<pre> public static void generateKey(String fname) { .. } public static void generateKeyPair(String fname1, String fname2) { .. } public static void sign(String msg, String pkey, String skey, String rPkey, String ename) {..} public static void vrfy(String rpkey, String Pkey, String ename) {..} </pre>				
	After				
보안약점/ 보안취약점  이유 및 해결법	<pre> static boolean generateKey(String fname) { .. } static boolean generateKeyPair(String fname1, String fname2) { .. } static boolean sign(String msg, String pkey, String skey, String rPkey, String ename) {..} static boolean vrfy(String rpkey, String Pkey, String ename) {..} </pre>				
	<p>void 메소드로 작성하게 되면 메소드를 호출했을 때 성공적으로 메소드 안의 동작이 끝났는지 여부를 확인하기 어렵다.</p> <p>이를 해결하고자 프로그램의 가독성 및 유지보수를 향상하기 위해 boolean으로 메소드 동작의 성공, 실패 여부를 반환해주었다. 이를 활용할 수 있도록 main에서도 반환값을 체크해서 동작하도록 조건문을 추가하였다.</p>				

## 코드 리뷰 보고서

리뷰번호	013	작성자	곽채원	작성날짜	2023.06.04
관련규칙	38. Do not declare more than one variable per declaration.				
중요도	중				
	Before				
대상코드	<pre>private static int keyLen = 1024;    private static int keySize = 256;</pre>				
	After				
보안약점/ 보안취약점 이유 및 해결법	<pre>private static int keyLen = 1024; // RSA key size private static int keySize = 256; // AES key size</pre>				
	<p>Before 코드처럼 한 줄에 변수를 두 개 이상 선언하면 추후 개발자가 변수를 수정하거나 할 때 오류가 발생할 가능성이 있다. 이는 프로그램의 가독성과 유지보수성을 해친다.</p> <p>이를 해결하고자 변수를 한 줄에 하나씩 선언하고, 주석을 통해 사용 목적을 명확히 기술하였다.</p>				

## 코드 리뷰 보고서

리뷰번호	014	작성자	곽채원	작성날짜	2023.06.04
관련규칙	50. Be careful using visually misleading identifiers and literals.				
중요도	중				
대상코드	Before				
	<pre> 31     private static int keyLen = 1024;    // RSA key size 32     private static int keySize = 256;    // AES key size 33 34     private static KeyManage kmanager = new KeyManage(); 35     private static SigManage smanager = new SigManage(); 36     private static CipherManage cmanager = new CipherManage(); 37 38     public static void generateKey(String fname) { 39         Key secretKey = kmanager.generateKey(cipherAlgorithm, keySize); 40 41         if (secretKey != null) { 42             kmanager.writeKey(fname, secretKey); 43 44         } 45         System.out.println("A secretKey is successfully generated."); 46     } 47 48     public static void generateKeyPair(String fname1, String fname2) { 49         KeyPair keypair = kmanager.generateKeyPair(keyAlgorithm, keyLen); 50 </pre>				
보안약점/ 보안취약점 이유 및 해결법	After				
	<pre> 31     private static int RSA_keysize = 1024;    // RSA key size 32     private static int AES_keysize = 256;    // AES key size 33 34     private static KeyManage kmanager = new KeyManage(); 35     private static SigManage smanager = new SigManage(); 36     private static CipherManage cmanager = new CipherManage(); 37 38     public static void generateKey(String fname) { 39         Key secretKey = kmanager.generateKey(cipherAlgorithm, AES_keysize); 40 41         if (secretKey != null) { 42             kmanager.writeKey(fname, secretKey); 43 44         } 45         System.out.println("A secretKey is successfully generated."); 46     } 47 48     public static void generateKeyPair(String fname1, String fname2) { 49         KeyPair keypair = kmanager.generateKeyPair(keyAlgorithm, RSA_keysize); 50 </pre>				
	<p>Before 코드에서 keyLen과 keySize는 둘 다 키의 길이를 나타내는 변수명이다. 그러나 변수 이름에 명시적으로 용도를 적어놓은 것이 아니라서 39번과 49번 line 코드에서 각각 어떤 의미인지 한 눈에 파악하기 힘들다. 이렇게 코드를 모호하게 작성하게 되면 수정을 위해서 코드 전체를 꼼꼼히 살펴봐야 하기 때문에 유지보수성이 떨어진다. 이를 해결하기 위해 각각을 RSA_keysize, AES_keysize로 변경하여 알고리즘과 keysizes를 한눈에 구분하도록 변경하였다.</p>				

## 코드 리뷰 보고서

리뷰번호	015	작성자	곽채원	작성날짜	2023.06.04
관련규칙	24. Minimize the accessibility of classes and their members.				
중요도	중				
	Before				
	<pre> public class sign_verify {     private static String signAlgorithm = "SHA256withRSA";     private static String cipherAlgorithm = "AES";     private static String keyAlgorithm = "RSA";      private static int keyLen = 1024;    // RSA key size     private static int keySize = 256;    // AES key size      private static KeyManage kmanager = new KeyManage();     private static SigManage smanager = new SigManage();     private static CipherManage cmanager = new CipherManage();      public static void generateKey(String fname) {    █         public static void generateKeyPair(String fname1, String fname2) {    █             public static void sign(String msg, String pkey, String skey, String rPkey, String ename) {█                 public static void vrfy(String rpkey, String Pkey, String ename) {█ } </pre>				
대상코드	After				
	<pre> final class sign_verify {     private static String signAlgorithm = "SHA256withRSA";     private static String cipherAlgorithm = "AES";     private static String keyAlgorithm = "RSA";      private static int keyLen = 1024;    // RSA key size     private static int keySize = 256;    // AES key size      private static KeyManage kmanager = new KeyManage();     private static SigManage smanager = new SigManage();     private static CipherManage cmanager = new CipherManage();      static boolean generateKey(String fname) {    █         static boolean generateKeyPair(String fname1, String fname2) {    █             static boolean sign(String msg, String pkey, String skey, String rPkey, String ename) {█                 static boolean vrfy(String rpkey, String Pkey, String ename) {█ } </pre>				
보안약점/ 보안취약점  이유 및 해결법	<p>이 코드는 main과 같은 패키지에 있는 sign_verify 클래스이다. Before 코드에서 변수는 private으로 설정해주었으나 메소드가 전부 public이어서 다른 패키지에서도 얼마든지 메소드를 호출할 수 있게 된다. 또한 해당 클래스를 상속받아 메소드를 오버라이딩한다면 공격자의 입맛에 맞게 코드를 수정할 수 있다.</p> <p>이를 해결하고자 클래스의 경우 상속이 불가하도록 final로 설정해주어 공격자가 상속 후 메소드 오버라이딩을 할 수 없게끔 하였으며, 안에 위치한 함수들은 같은 패키지 내에서만 사용할 수 있도록 default로 접근 제한을 두었다.</p>				

## 코드 리뷰 보고서

리뷰번호	016	작성자	곽채원	작성날짜	2023.06.05
관련규칙	43. Use a try-with-resources statement to safely handle closeable resources.				
중요도	상				
	Before				
대상코드	<pre> public boolean writeCipher(byte[] input, String output, Cipher c) {     try {         FileOutputStream fos = new FileOutputStream(output);         CipherOutputStream cos = new CipherOutputStream(fos,c);         cos.write(input);         cos.flush();         cos.close();         return true;     } catch (IOException e) {         e.printStackTrace();     }     return false; } </pre>				
	After				
보안약점/ 보안취약점 이유 및 해결법	<pre> public boolean writeCipher(byte[] input, String output, Cipher c) {     try (FileOutputStream fos = new FileOutputStream(output);          CipherOutputStream cos = new CipherOutputStream(fos,c)) {         cos.write(input);         cos.flush();         return true;     } catch (IOException e) {         e.printStackTrace();     }     return false; } </pre>				
	<p>Before 코드는 FileOutputStream과 CipherOutputStream이라는 두 개의 스트림을 활용하고 있다. 다만 닫혀야 하는 자원이 제대로 닫히지 않는 코드이다. 우선, FileOutputStream은 닫는 구문조차 없고, CipherOutputStream은 try문 안에서 예러가 발생할 시 닫히지 않는다. 자원이 제대로 닫히지 않고 코드를 떠돌아다니게 되면 공격자가 악용할 가능성이 있다.</p> <p>이를 해결하기 위해서는 finally블록을 사용해도 좋지만 보다 안전하게, 무조건 try문 안에서 사용한 자원을 닫아주기 위해서 try-with-resource 구문을 After 코드처럼 활용하였다.</p>				

## 코드 리뷰 보고서

리뷰번호	017	작성자	곽채원	작성날짜	2023.06.07
관련규칙	63. Detect and remove superfluous code and values.				
중요도	중				
	Before				
대상코드	<pre> c.init(Cipher.DECRYPT_MODE, secretKey);  byte[] signature = cmanager.readCipher_sig(ename + "_sig.txt", c);     </pre> <pre> public byte[] readSign(String fname) {     try (FileInputStream fis = new FileInputStream(fname)) {         return fis.readAllBytes();     } catch (IOException e) {         e.printStackTrace();     }     byte[] empty = {};     return empty; }     </pre>				
	After				
보안약점/ 보안취약점 이유 및 해결법	<pre> package Manager;  import java.io.FileInputStream;..   public final class SigManage {     public SigManage() { }      public Signature generate(String algorithm) {..}      public byte[] makeSign(String input, Signature sig) {..}      public boolean verifySign(byte[] signature, Signature sig) {..} }     </pre> <p>파일에 암호화 과정 없이 저장된 서명을 읽어오는 readSign메소드는 서명을 암호화하며 저장하기 시작하면서 더 이상 쓰이지 않으나, 별도의 클래스로 저장되어 있어 unused code 경고 메시지가 뜨지 않았다. 코드 수정이 거듭되면서 충분히 벌어질 수 있는 일이며, 이렇게 안 쓰는 메소드를 남겨두는 것은 공격자가 악용할 여지를 주며 코드를 쓸데없이 복잡하게 만든다.</p> <p>이를 해결하기 위해 해당 메소드는 지워주었다.</p>				