Acquiring Private Keys for Packet Analysis

Author: Robert Bullen
Revision: 8
Date: 2013/12/11

Introduction

In order to decrypt SSL packet captures for application layer analysis, analysts must ask the information security team to deliver private keys belonging to the SSL servers of interest. This document explains popular key file formats and, of those, which will be accepted by popular packet analyzers so that the analysts and security personnel can agree on the correct format. If the delivered key files are incompatible with the analyst's packet analyzer and must be converted, this document covers the process and tools for that, as well.

Definitions

In the encryption/decryption space, there are a few different cryptographic key file formats and terminology is often used pretty loosely. Therefore it is important that the analyst and the security team have a common understanding of these file types.

It is unusual for a private key to be stored or passed around in the raw. Usually it exists within a password-protected container file, which may also contain an associated certificate chain that describes the private key's issuance and authenticity. Discussed herein are three such container file formats.

Container File Format	File Extensions*	Description
PEM		PEM stands for Privacy-enhanced Electronic Mail, defined in RFCs 1421–1424. The impression I get is that the PEM container file format is a branch of those RFCs and the only piece that became widespread.
	pem	If a PEM file contains both an encrypted private key and its certificate chain, convention dictates that the *.pem* extension is used.
	key	If it contains only the unencrypted private key in RSA format, the de facto extension is *.key* (hereafter referred to as KEY files).
	cer, cert	If it contains only the certificate chain then *.cer* or *.cert* is commonly used for the file extension.**
PKCS#12	pfx, p12, pkcs12	PKCS #12 is a member of the family of standards called Public-Key Cryptography Standards (PKCS) published by RSA Laboratories. See the Wikipedia article.

Container File Format	File Extensions*	Description
Java Keystore	jks, keystore	Java Keystores are the preferred container format in the Java domain, most notably used by Oracle WebLogic. See the Wikipedia article.

**It should be noted that extensions for these files can be used just as loosely as the terminology surrounding them. This table includes conventions used when naming container files, but a file's extension is not an absolute indicator of its format or contents.*

***An interesting feature of PEM, and in fact all three of these file formats, is that they are not required to contain a private key at all. They could hold only certificates, in which case they are useless for decryption.*

Packet Analyzer Compatibility

Packet analyzers expect private keys in PEM format. Some analyzers support password-protected PEM files, but nearly all will take a KEY file (see the container file format above for PEM and KEY file definitions). To determine which, consult the following compatibility matrix.

Packet Analyzer	Supports PEM?	Supports KEY?
Wireshark	Yes	Yes
Network Instruments Observer	No	Yes
WildPackets OmniPeek	Yes	No
Compuware AMD	No	Yes

TODO: This matrix needs more tools, more research, and more validation!

If the analyst has been given a PEM file (and its password) and his tool of choice supports PEM, then manually extracting the key is not necessary. If, however, a raw KEY file is needed, continue reading.

Private Key Extraction

How the private key is extracted depends upon the starting format of the container file:

Container File Format	Private Key Extraction Tool(s)
PEM	OpenSSL
PKCS#12	OpenSSL
Java Keystore	keytool and jksExportKey

PEM and PKCS#12 Private Key Extraction using OpenSSL

To extract a private key from its PEM or PKCS#12 container file, use *OpenSSL*—an open source command line utility widely available for all operating systems. Once it is installed, execute it with one of the following commands:

Format	Extraction Command Line
PEM	`openssl rsa -in <file.pem> -out <file.key> [-passin pass:<password>]`
PKCS#12	`openssl pkcs12 -in <file.pfx> -out <file.key> [-passin pass:<password>] -nodes -nocerts`

The relevant command arguments are:

Argument	Description
`rsa` or `pkcs12`	Specifies the mode in which OpenSSL should operate. As mentioned previously, there are lots of different encryption/decryption algorithms and container file formats, and OpenSSL needs this piece of information to correctly handle the input or output format.
`-in <file.ext>`	Indicates the input file. Replace `<file.ext>` with the actual PEM or PKCS#12 file you were given.
`-out <file.key>`	Specifies the output file in which to place the extracted key. Replace `<file.key>` with the desired file name. This key file is ultimately what will be used by the packet analyzer.
`-passin pass:<password>`	Specifies the container file's password. Replace **`<password>`** with the container file's actual password. If these arguments are not supplied on the command line, OpenSSL will prompt for the password.
`-nodes`	For PKCS#12 files only. "No DES" tells OpenSSL to not encrypt the KEY file being output.
`-nocerts`	For PKCS#12 files only. Prevents the certificate chain from being included in the output file.

Java Keystore Private Key Extraction with keytool and jksExportKey

OpenSSL doesn't work with Java Keystores. You must turn to a couple Java-based tools and invoke them in a two-step process.

Step 1: Discover the Private Key Alias Using keytool

The first tool is (appropriately) named *keytool*. It is included in Java installations and can be used to output a summary of the Java Keystore of interest. JKS files may contain multiple entries and each has its own *alias*. The private key is one such entry and if its alias is not known a priori, keytool can be used to find it.

Step	Command Line
Discover the private key alias	`keytool -list [-v|-rfc] -keystore <file.jks> [-storepass <password>]`

The relevant command line arguments are:

Argument	Description
`-list`	Tells keytool to print a summary of the contents of the keystore.
`-v` or `-rfc`	Optionally modifies the output of **`-list`**; **`-v`** forces the summary to be in human readable format and **`-rfc`** emulates the contents of PEM files. Neither of these is necessary as the alias you are looking for will be included regardless.
`-keystore` ***`<file.jks>`***	Indicates the input file. Replace *`<file.jks>`* with the actual JKS file.
`-storepass` ***`<password>`***	Specifies the container file's password. Replace ***`<password>`*** with the JKS file's actual password. If these arguments are not supplied on the command line, keytool will prompt for the password.

The output will include a header followed by one or more entry summaries, each starting with its alias. That is the text to look for. An example of using keytool on a JKS file named "my_keystore.jks" to discover that the alias for the private key entry is "my_alias" is at right. (Coloring added for readability.)

```
> keytool.exe -list -keystore my_keystore.jks
Enter keystore password: ******

Keystore type: JKS
Keystore provider: SUN

Your keystore contains 1 entry

my_alias, Nov 13, 2008, PrivateKeyEntry,
Certificate fingerprint (MD5):
B0:B6:00:D2:84:20:42:A1:C5:03:9A:F4:CA:F0:A5:24
```

Step 2: Extract the Private Key Using jksExportKey

Unfortunately keytool doesn't do the critical task of extracting private keys from JKS containers. For that another piece of code is needed that is not part of the Java runtime environment. It is (also appropriately) named *jksExportKey*. As a JAR file made available by miteff.com, it must be downloaded before it can be used. It is invoked using Java as follows:

Step	Command Line
Extract the private key	`java -jar jksExportKey-1.0.jar <file.jks> <alias> <password> > <file.key>`

The relevant command line arguments are:

Argument	Description
`-jar jksExportKey-1.0.jar`	Tells Java to execute the code in the JAR file `jksExportKey-1.0.jar`. The full path to jksExportKey (including quotes if the path contains spaces) may be necessary if it isn't placed it in a location where it can be found implicitly.
`<file.jks>`	Indicates the input file. Replace *`<file.jks>`* with the actual JKS file.
`<alias>`	Tells jksExportKey the alias of private key entry to extract.
`<password>`	Specifies the container file's password. Replace ***`<password>`*** with the JKS file's actual password. Unlike OpenSSL and keytool, the password must be supplied as an argument.
`<file.key>`	Specifies the output file in which to place the extracted key. Replace *`<file.key>`* with the desired file name. This key file is ultimately what will be used by the packet analyzer.

References

Besides the links embedded in the body, the following links were useful when putting together this document:

- http://serverfault.com/questions/9708/what-is-a-pem-file-and-how-does-it-differ-from-other-openssl-generated-key-file
- http://how2ssl.com/articles/working_with_pem_files/
- http://www.sslshopper.com/article-most-common-openssl-commands.html
- https://www.sslshopper.com/article-most-common-java-keytool-keystore-commands.html
- http://www.startux.de/index.php/java/44-dealing-with-java-keystoresyvComment44
- http://sharkfest.wireshark.org/sharkfest.09/AU2_Blok_SSL_Troubleshooting_with_Wireshark_and_Tshark.pps