UNMITIGATED RISK

un.mit.i.gat.ed: Adj. Not diminished or moderated in intensity or severity; unrelieved. risk: N. The possibiity of suffering harm or loss; danger.

MONTHLY ARCHIVES: NOVEMBER 2015

Graphene CLI

A few weeks ago we released Graphene, a PKCS #11 binding for NodeJS. Today we are releasing a CLI based on the same library.

Our goal was to make it easy for you to work with PKCS#11 devices in a vendor neutral way without the complexity of installing and configuring a bunch of miscellaneous packages.

Using the tool itself is pretty straight forward, load the PKCS#11 module and open a session to a slot then you are ready to go:

```
rmh@peculiar-02:/home/graphene# graphene

> module load -l /usr/safenet/lunaclient/lib/libCryptoki2_64.so -n test


Module info

============================

  Library: /usr/safenet/lunaclient/lib/libCryptoki2_64.so

  Name: test

  Description: Chrystoki

  Cryptoki version: 2.20


> slot open --slot 0 -p {YourPin}
```

```
Session is started

>
```

Once logged in you can always find help with the '?' command:

```
> ?

  Commands:

    ?         output usage information
    exit      exit from the application
    module    load and retrieve information from the PKCS#11 module
    slot      open a session to a slot and work with its contents
    object    manage objects on the device
    hash      compute a hash for a given file
    test      benchmark device performance for common algorithms

  Note:

    all commands require you to first load the PKCS #11 module

      > module load -l /path/to/pkcs11/lib/name.so -n LibName
```

Some things you can do with the tool:

- Enumerate the supported algorithms of your device
- Enumerate and manage the objects on your device
- Benchmark the performance of your device
- Hash a file utilizing your device

We recently benchmarked the capabilities of a SafeNet G5, you can see the results here.
We hope you find it useful.

This entry was posted in Programming, Security and tagged cli, open source, pkcs#11, tool on November 17, 2015 [https://unmitigatedrisk.com/?p=541] by rmhrisk.

Certificate based Encryption in PDFs

The PDF format is the most used file format on the internet but unfortunately, the specification that documents it leaves a lot to be desired when it comes to producing signed and encrypted documents.

PDF is still the only truly cross-platform “paper like” experience available to users. It also has a number great of features that many are not aware of, one of which is the ability to encrypt the PDFs so they are not readable without having access to the appropriate secrets.

It supports two approaches to this encryption, one based on passwords and one based on digital certificates. In a later post I will discuss the issues in password based encryption but here I want to talk about the second approach as it offers the potential for the most security.

When looking at our findings it’s important to keep in mind the history and timeline of the PDF format. It was released as a free to implement specification in 1993 and then was standardized by ISO in 2008. It has been left largely unchanged since then. This is important because much of the practices and approaches in the standard were considered state-of-the-art in the 90s but are no longer considered strong today.

NOTE: The below is based on our findings while reading a pre-release of ISO 32000-2 that was approximately one-year-old. It is notably more readable than its predecessor, 32000-1, in many areas but it seems little change has been made to how signing and encryption is handled. Unfortunately the ISO standardization process does not produce public “intermediate” documents and this was the freshest document we could find.

Message Format

The granddaddy of signature formats is something called PKCS #7. This was defined by RSA in the mid-90s and later handed off to the IETF when they republished in as Cryptographic Message Syntax (CMS) in 1999. CMS is now a superset of PKCS #7 where additional attributes and cryptographic algorithms are also supported. The PDF specification however still references signature format as being PKCS #7 but its references are to the at least one CMS RFC and not the PKCS#7 one. Most implementations, however, such as Adobe Acrobat, have added support for algorithms and options that are available in the latest CMS specifications so we can assume this is what they mean. This is not a security issue, but it does create a mess when it comes to interoperability.

This inconsistency in the specification makes it harder for an implementer to know which bits and pieces to implicitly pull in from different unreferenced specifications. This not only makes interoperability more challenging, but it also results in a lack of common capabilities across implementations.

Padding

The most popular asymmetric cryptographic algorithm of all time is clearly RSA, it serves as the foundation of most key management and distributions solutions in use today. When encrypting data with RSA you need to "pad" the data you are encrypting. In the 90's you would pad using a scheme called RSA PKCS v1.5. The problem is that in the late 90s it became clear this scheme was attackable.

- Chosen Ciphertext Attacks Against Protocols Based on the RSA Encryption Standard PKCS #1, 1998
- New Attacks on PKCS#1 v1.5 Encryption, 2000

These attacks are most relevant to systems that are online, for example, a website or API applying verifying electronic signatures or decrypting documents. As a result of these weaknesses, in 2001 the world started moving to something called OAEP that addressed the identified risks in PKCS v1.5. It takes a decade or more to depreciate a cryptographic technique that is broadly in use, and thankfully we now see cryptographic libraries (such as the new WebCrypto) deprecating these weaker constructs to prevent future scenarios from accidentally supporting them.

Unfortunately, it seems the PDF specification was never updated to use the more secure OAEP padding scheme. While there is nothing stopping a client that implements the standard from also using the more modern padding algorithm also (just as many have adopted features in the latest CMS specification even though not part of the PDF specification) it seems the most popular client, Adobe Acrobat, has not decided to do that. This means if people want to encrypt documents using the more secure approach they won't be able to work with Adobe Acrobat.

It is about time the standard incorporated OAEP and thankfully doing so is almost as easy as "search and replace".

Content Encryption

The PDF specification states that AES should be used in Cipher Block Chaining (CBC) mode. There is nothing wrong with CBC per-se, with that said it is "easy to get it wrong" and for this reason, most practitioners will tell you to use a mode called Galois Counter Mode (**GCM**). This mode of encryption is an "authenticated" mode where you can easily tell if a message has been modified when decrypting.

It is true that the PDF format includes an MD5 as an integrity mechanism, but unless special care is given one could still easily expose the associated attack vectors.

The ability to verify the integrity of an encrypted message before decryption is materially important to systems that handle lots of documents. Doing so as part of the encryption mechanism, as is done with GCM, ensures this happens. GCM has been around since 2005, three years before the ISO standard for PDF was published — It is about time the standard incorporated it.

Key Strength

In the certificate based PDF encryption scheme there are two “secrets” that need to be protected, the first is called a “seed” by the specification. This seed is used to derive the content encryption key (CEK) that is used to encrypt the actual PDF content.

This “seed” is a 20-byte random value. The 20-bytes was more than sufficient when working with shorter key lengths like those used in 3DES but with AES-256 it is not sufficient. When deriving the key in accordance with the specification this “seed” is mixed with a hash of portions of the document. It is possible the authors thought the use of the hash was sufficient to provide the additional entropy, but it is not. These inputs are not random, they may be unique to a given document, but all instances of that document would have the same values. As a result, there is insufficient entropy to get all the security benefits of AES-256.

The specification should be updated to indicate that for a 128-bit AES key you use at least 16-bytes, for a 256-bit AES key you use at least 32-bytes of entropy.

ECC Support

ECC represents an important cryptographic tool with lots of great security properties. Several PDF clients, including Acrobat Reader, support signing and verifying signatures based on ECC. There would probably be even more, but again, the PDF specification we have states the format is PKCS #7 even though it links to an older copy of the CMS standard.

Those clients that do this do support ECC do so via ECDSA (this was specified in 2002 for CMS and later updated in 2010).

It is also possible to encrypt with ECC. This would be done using ECDH and is also documented in the same RFCs. Adding support into the specification would make sense since many cryptographic guidelines across the world mandate the use of ECC based algorithms. Additionally signing and encryption go hand-in-hand and if one signs with ECC they surely would also like to encrypt the same document and today that’s not possible, at least in Adobe Reader, without also being enrolled for an RSA certificate.

Long story short, it is about time both ECC signing and encryption are supported by the PDF standard.

Implementation Guidance

The specification as written offers essentially no implementation guidance. There are numerous cases of this, but one of the more glaring implications comes up when we think about the key hierarchy used in encryption.

While the specifications use of two keys in the key hierarchy is a convoluted approach, if done correctly it can work fine. One of the larger issues here an implementor needs to be mindful of is the "effective key strength" they are offering the users of their products. For example, if you encrypt the "seed" using 3DES and the "content" with AES then the content is only as secure as the 3DES key. We have encountered at least one client that does exactly this. Another variant we have seen is a client that encrypts the "seed" with AES-128 and the content with "AES-256", of course, the client tells the users the file was protected with the larger key length.

The specification should be updated to make it clear that both keys need to be protected with algorithms that offer the same effective security, and in fact, should simply be protected using the same algorithm.

Backward Compatibility

Another example of missing implementation guidance is that of backwards compatibility. While I am sure there are examples outside of how documents are signed and encrypted what is directly obvious to us is that the specification includes support for many algorithms that are weak and/or broken (for example MD2, MD5, DES, and 3DES).

The text really should be updated to make it clear that no new documents should be created using these algorithms and recommend that clients warn when viewing documents that have been produced with them since the guarantees of privacy, integrity and authentication the user expects are likely not being met.

Conclusion

The PDF format uses cryptographic approaches from the 90s and implementers have pulled in, on an as needed basis of more modern cryptographic approaches. Today algorithms that would be used to build such a standard would be AES-GCM, ECDSA, ECDH, and RSA-OAEP almost all of which are not supported by PDF as specified.

Thankfully the ISO 32000-2 specification is not yet complete, it is my hope that it will be soon, and that the editors of this specification read this and update the draft to incorporate this feedback. If they do then we will all be better off.

Ryan & Yury

This entry was posted in Security, Standards and tagged Encryption, PDF, Signing on November 10, 2015 [https://unmitigatedrisk.com/?p=536] by rmhrisk.