SYNACKTIV
DIGITAL SECURITY

Bypassing HMAC validation in OWASP ESAPI symmetric encryption

Security advisory

10/12/2013

Renaud Dubourguais & Renaud Feil

1. Vulnerability description

1.1. About OWASP ESAPI

From the ESAPI website (http://code.google.com/p/owasp-esapi-java/):

"OWASP ESAPI (The OWASP Enterprise Security API) is a free, open source, web application security control library that makes it easier for programmers to write lower-risk applications. The ESAPI for Java library is designed to make it easier for programmers to retrofit security into existing applications. ESAPI for Java also serves as a solid foundation for new development."

1.2. The issue

On August the 21st 2013, Philippe Arteau posts a vulnerability on the *esapi-dev* mailing list: if a *Ciphertext* structure generated with ESAPI is tampered to contains a HMAC that is null, the HMAC validation is bypassed. By default, ESAPI uses symmetric encryption in CBC mode, which is vulnerable to padding oracle attacks once the HMAC is bypassed.

On September the 2nd 2013, version 2.1.0 is published and corrects the issue. This issue is tracked as CVE-2013-5679.

During a penetration test, Synacktiv discovered that the security fix for the CVE-2013-5679 vulnerability ("MAC Bypass in ESAPI Symmetric Encryption") does not prevent a malicious user to bypass the HMAC validation on a serialized cipher text. Indeed, the *Ciphertext* can be tampered to use a 1-byte only HMAC value, which can be brute-forced by an attacker.

The serialized cipher text generated by ESAPI has the following structure:

Entry	Description
< kdfInfo / kdfPrf : int >	identification of the key derivation function
< timestamp : long >	time when the cipher text is generated
< cipherlen : short >	size of the "cipher" string
< cipher : str >	algorithm for encryption and decryption
< keysize : short >	size of the HMAC validation key (in bits)
< blocksize : short >	size of the encryption blocks
< ivlen : short >	size of the IV (in bytes)
< iv : str >	Initialization Vector
< ciphertextlen : int >	size of the cipher text in bytes
< ciphertext : str >	encrypted value
< maclen : short >	size of the HMAC in bytes
< mac : str >	HMAC of the IV and the cipher text

The *keysize* field, under the control of a malicious user, is used to compute the HMAC encryption key. The ESAPI library checks for a minimal key size of 56 bytes using an assertion. However, if the application is not started with the *-ea* Java option (which is usually not the case in production environments), assertions are turned off. By setting the *keysize* field in a serialized *CipherText* to a value between 1 and 8 (bits), ESAPI will compute the validation HMAC using a 1 byte key, which can be trivially brute-forced to produce a valid HMAC.

After contacting the ESAPI developers on December the 2nd 2013, it appears that further checks have been implemented in a development branch (*kww-crypto-2.1.1*). Specifically, the change r1908 replaces *assert* with *if()* and corrects the issue. However, as of December 10th 2013, this change hasn't been pushed into the main ESAPI release or even in the main SVN snapshot for the future version 2.1.1 and ESAPI users are vulnerable.

1.3. Affected versions

The following versions are affected:

- 2.0.1
- 2.1.0
- 2.1.1-SNAPSHOT (from SVN)

1.4. Mitigation

While waiting for the 2.1.1 version to be published, ESAPI users are advised not to trust the ESAPI HMAC validation. It is advised to replace the default CBC mode used in ESAPI with an authenticated encryption algorithm mode such as CCM (Counter with CBC-MAC) or GCM (Galois/Counter Mode). Even if the ESAPI HMAC is broken, the underlying encryption algorithm will have it's own MAC which will prevent padding oracle attacks.

2. Technical description and proof-of-concept

2.1. From the code

If *keysize* is set to [1-8], the *CryptoHelper.computeDerivedKey()* method used to compute the HMAC encryption key will return an *authKey* of 1 byte. The following extract from the file *org/owasp/esapi/crypto/CryptoHelper.java* shows were the *computeDerivedKey()* is called:

```
public static boolean isCipherTextMACvalid(SecretKey sk, CipherText ct)
{
   if ( CryptoHelper.isMACRequired( ct ) ) {
      try {
         SecretKey authKey = CryptoHelper.computeDerivedKey(sk, ct.getKeySize(),
"authenticity");
         // VULN: if ct.getKeySize() is 1, authKey.length will be 1 byte
         boolean validMAC = ct.validateMAC( authKey );
         return validMAC;
      } catch (Exception ex) {
         logger.warning(Logger.SECURITY_FAILURE, "Unable to validate MAC for ciphertext " +
ct, ex);
         return false;
      }
   }
   return true;
}
```

The *computeDerivedKey()* method is found in the file *org/owasp/esapi/crypto/KeyDerivationFunction.java*:

```
public SecretKey computeDerivedKey(SecretKey keyDerivationKey, int keySize, String purpose)
   throws NoSuchAlgorithmException, InvalidKeyException, EncryptionException
{
   [...]
   assert keySize >= 56 : "Key has size of " + keySize + ", which is less than minimum of
56-bits.";
   // VULN: assert is not triggered in production
   [...]
   keySize = calcKeySize( keySize ); // if keySize is 1, calcKeySize() returns 8 bits
   [...]
   do {
      mac.update( ByteConversionUtil.fromInt( ctr++ ) );
      mac.update(label);
      mac.update((byte) '\0');
      mac.update(context);
      tmpKey = mac.doFinal(ByteConversionUtil.fromInt( keySize ) );
      if ( tmpKey.length >= keySize ) {
         len = keySize;
      } else {
         len = Math.min(tmpKey.length, keySize - totalCopied);
      }
      System.arraycopy(tmpKey, 0, derivedKey, destPos, len);
      label = tmpKey;
      totalCopied += tmpKey.length;
      destPos += len;
   } while( totalCopied < keySize );
   [...]
   return tmpkey; // VULN: tmpkey will only contain 1 byte
}
```

If *CryptoHelper.computeDerivedKey()* returns an *authKey* containing only 1 byte, *ct.validateMac(authKey)* will compute a HMAC using a 1-byte key. It will then compare the calculated HMAC with the HMAC in the *CipherText* sent by the attacker. The *validateMAC()* method is in the file *org/owasp/esapi/crypto/CipherText.java*:

```
public boolean validateMAC(SecretKey authKey) {
   boolean requiresMAC = ESAPI.securityConfiguration().useMACforCipherText();

   if (  requiresMAC && macComputed() ) {
      byte[] mac = computeMAC(authKey); // VULN: authKey.length is 1 byte
      assert mac.length == separate_mac_.length : "MACs are of different lengths. Should
both be the same.";
      return CryptoHelper.arrayCompare(mac, separate_mac_);
      [...]
```

It means that a malicious user can set the *keysize* field to a value between 1 and 8 in the *CipherText* structure, recompute all possible HMAC in the *CipherText* using a 1-byte key, until *CipherText.computeMAC(key)* matches the forged HMAC. Brute forcing 1 byte takes only 256 iterations in the worst case. When the right value is found, *CipherText.validateMAC(key)* will return *true* and the HMAC check will be bypassed.

2.2. Impact

Once the MAC is bypassed, the attacker will be in a good position to attack the encryption layer, by using for example padding oracle attacks on the default CBC mode used by ESAPI. Other cryptographic attacks are also possible depending on the encryption algorithm and mode used.

2.3. Proof of concept

A sample application using the ESAPI library and a proof of concept of the attack can be found here: http://www.synacktiv.fr/ressources/owasp_esapi_hmac_bypass_poc.tgz .

To decrypt a token, use:

```
$ java EsapiVictim d <token>
```

To generate a token, use:

```
$ java EsapiVictim g key=value
```

To use the exploit trying to bypass the HMAC validation of *EsapiVictim.java*, the Java *classpath* should be adjusted on your system. Then use the exploit:

```
$ python esapi_brute.py <token>
```

If *esapi_brute.py* bypasses the MAC validation, a padding exception will be thrown by the *EsapiVictim* application:

```
"Caused by: javax.crypto.BadPaddingException: Given final block not properly padded"
```