**Problem**

OpenSSL before 0.9.8zd, 1.0.0 before 1.0.0p, and 1.0.1 before 1.0.1k does not enforce certain constraints on certificate data, which allows remote attackers to defeat a fingerprint-based certificate-blacklist protection mechanism by including crafted data within a certificate's unsigned portion, related to crypto/asn1/a\_verify.c, crypto/dsa/dsa\_asn1.c, crypto/ecdsa/ecs\_vrf.c, and crypto/x509/x\_all.c.

The Distinguished Encoding Rules (DER) format is **a defined standard which Bitcoin uses to encode ECDSA signatures**. A signature consists of a random number called an r value, and the signature itself, called the s value.

The Elliptic Curve Digital Signature Algorithm (ECDSA) is **a Digital Signature Algorithm (DSA) which uses keys derived from elliptic curve cryptography (ECC)**. It is a particularly efficient equation based on public key cryptography (PKC).

DER = Distinguished Encoding Rules

DSA = Digital Signature Algorithm

ECDSA = Elliptic Curve Digital Signature Algorithm

<https://github.com/openssl/openssl/commit/684400ce192dac51df3d3e92b61830a6ef90be3e>

\*) Fix various certificate fingerprint issues.

By using non-DER or invalid encodings outside the signed portion of a

certificate the fingerprint can be changed without breaking the signature.

Although no details of the signed portion of the certificate can be changed

this can cause problems with some applications: e.g. those using the

certificate fingerprint for blacklists.

1. Reject signatures with non zero unused bits.

If the BIT STRING containing the signature has non zero unused bits reject

the signature. All current signature algorithms require zero unused bits.

2. Check certificate algorithm consistency.

Check the AlgorithmIdentifier inside TBS matches the one in the

certificate signature. NB: this will result in signature failure

errors for some broken certificates.

Thanks to Konrad Kraszewski from Google for reporting this issue.

3. Check DSA/ECDSA signatures use DER.

Reencode DSA/ECDSA signatures and compare with the original received

signature. Return an error if there is a mismatch.

This will reject various cases including garbage after signature

(thanks to Antti Karjalainen and Tuomo Untinen from the Codenomicon CROSS

program for discovering this case) and use of BER or invalid ASN.1 INTEGERs

(negative or with leading zeroes).

<https://code.woboq.org/crypto/openssl/crypto/asn1/asn1.h.html> (pas important)

* /\* Function codes. \*/
* #define ASN1\_F\_ASN1\_VERIFY 137
* #define ASN1\_F\_ASN1\_ITEM\_VERIFY 197

<https://code.woboq.org/crypto/openssl/crypto/err/err.h.html#214> (pas important)

* #define ASN1err(f,r) ERR\_PUT\_error(ERR\_LIB\_ASN1,(f),(r),\_\_FILE\_\_,\_\_LINE\_\_)

<https://code.woboq.org/crypto/openssl/crypto/asn1/asn1t.h.html#877> (pas important)

<https://code.woboq.org/crypto/openssl/crypto/dsa/dsa_asn1.c.html#91> (pas important)

**buggy\_a\_verify.c**

if (signature->type == V\_ASN1\_BIT\_STRING && signature->flags & 0x7)

{

ASN1err(ASN1\_F\_**ASN1\_VERIFY**, ASN1\_R\_INVALID\_BIT\_STRING\_BITS\_LEFT);

return -1;

}

**dsa\_asn1.c**

int DSA\_verify(int type, const unsigned char \*dgst, int dgst\_len,

const unsigned char \*sigbuf, int siglen, DSA \*dsa)

{

DSA\_SIG \*s;

int ret=-1;

s = DSA\_SIG\_new();

if (s == NULL) return(ret);

if (d2i\_DSA\_SIG(&s,&**sigbuf**,siglen) == NULL) goto err;

ret=DSA\_do\_verify(dgst,dgst\_len,s,dsa);

err:

DSA\_SIG\_free(s);

return(ret);

}

**ecs\_vrf.c**

int ECDSA\_verify(int type, const unsigned char \*dgst, int dgst\_len,

const unsigned char \*sigbuf, int sig\_len, EC\_KEY \*eckey)

{

ECDSA\_SIG \*s;

int ret=-1;

s = ECDSA\_SIG\_new();

if (s == NULL) return(ret);

if (d2i\_ECDSA\_SIG(&s, &**sigbuf**, sig\_len) == NULL) goto err;

ret=ECDSA\_do\_verify(dgst, dgst\_len, s, eckey);

err:

ECDSA\_SIG\_free(s);

return(ret);

}

**x\_all.c**

int X509\_verify(X509 \*a, EVP\_PKEY \*r)

{

return(ASN1\_item\_verify(ASN1\_ITEM\_rptr(X509\_CINF),a->sig\_alg,

a->signature,a->cert\_info,r));

}

**Fix**

**a\_verify.c**

if (signature->type == V\_ASN1\_BIT\_STRING && signature->flags & 0x7)

{

ASN1err(ASN1\_F\_ASN1**\_ITEM**\_VERIFY, ASN1\_R\_INVALID\_BIT\_STRING\_BITS\_LEFT);

return -1;

}

**dsa\_asn1.c**

int DSA\_verify(int type, const unsigned char \*dgst, int dgst\_len,

const unsigned char \*sigbuf, int siglen, DSA \*dsa)

{

DSA\_SIG \*s;

**const unsigned char \*p = sigbuf;**

**unsigned char \*der = NULL;**

**int derlen = -1;**

int ret=-1;

s = DSA\_SIG\_new();

if (s == NULL) return(ret);

if (d2i\_DSA\_SIG(&s,&**p**,siglen) == NULL) goto err;

**/\* Ensure signature uses DER and doesn't have trailing garbage \*/**

**derlen = i2d\_DSA\_SIG(s, &der);**

**if (derlen != siglen || memcmp(sigbuf, der, derlen)) (\*\*Spécifiquement ici\*\*)**

**goto err;**

ret=DSA\_do\_verify(dgst,dgst\_len,s,dsa);

err:

**if (derlen > 0)**

**{**

**OPENSSL\_cleanse(der, derlen);**

**OPENSSL\_free(der);**

**}**

DSA\_SIG\_free(s);

return(ret);

}

**ecs\_vrf.c**

#include "cryptlib.h"

int ECDSA\_verify(int type, const unsigned char \*dgst, int dgst\_len,

const unsigned char \*sigbuf, int sig\_len, EC\_KEY \*eckey)

{

ECDSA\_SIG \*s;

**const unsigned char \*p = sigbuf;**

**unsigned char \*der = NULL;**

**int derlen = -1;**

int ret=-1;

s = ECDSA\_SIG\_new();

if (s == NULL) return(ret);

if (d2i\_ECDSA\_SIG(&s, &**p**, sig\_len) == NULL) goto err;

**/\* Ensure signature uses DER and doesn't have trailing garbage \*/**

**derlen = i2d\_ECDSA\_SIG(s, &der);**

**if (derlen != sig\_len || memcmp(sigbuf, der, derlen)) (\*\*Spécifiquement ici\*\*)**

**goto err;**

ret=ECDSA\_do\_verify(dgst, dgst\_len, s, eckey);

err:

**if (derlen > 0)**

**{**

**OPENSSL\_cleanse(der, derlen);**

**OPENSSL\_free(der);**

**}**

ECDSA\_SIG\_free(s);

return(ret);

}

**x\_all.c**

int X509\_verify(X509 \*a, EVP\_PKEY \*r)

{

**if (X509\_ALGOR\_cmp(a->sig\_alg, a->cert\_info->signature))**

**return 0;**

return(ASN1\_item\_verify(ASN1\_ITEM\_rptr(X509\_CINF),a->sig\_alg,

a->signature,a->cert\_info,r));

}

<https://www.openssl.org/docs/man1.1.1/man3/X509_ALGOR_cmp.html>

X509\_ALGOR\_cmp() compares a and b and returns 0 if they have identical encodings and nonzero otherwise.