Vulnerability CVE-2016-2108, aka the „Negative Zero“ issue
Research for Computer Security MTAT.03.134 by Alar Leemet

Supervisor: Meelis Roos

Tartu 2017
INTRODUCTION

OpenSSL is an open source software library used in applications that need secure communications over computer networks. It is used to protect network traffic content against eavesdropping or verification of the identity of connection parties. [1][2]

The vulnerability in research is a memory corruption problem within OpenSSL’s Abstract Syntax Notation One (ASN.1), which is a language independent formal data description notation. This issue is caused by two smaller bugs that pose no security impact on their own. [3][4]

The first of those issues, which is related to the mishandling of negative zero integers, was detected in April 2015 and patched in June 2015 without anyone recognizing its security impact at that time. This is also where the nickname “Negative Zero” issue comes from. The second bug, which is related to the mishandling of large universal tags, was reported in March 2016. In the same month, David Benjamin revealed that the two issues combined can result in an exploitable memory corruption vulnerability. An attacker could cause OpenSSL to crash or even execute arbitrary code. [5]

DETAILS AND EXPLOITING THE VULNERABILITY

With the issue reported in April 2015, ASN.1 encoding the value 0 (zero) represented as a negative integer can cause a buffer underflow with an out-of-bounds write in i2c_ASN1_INTEGER. This alone cannot be exploited by the attacker as the ASN.1 parser does not normally create “negative zeroes” when parsing ASN.1 input. [6]

The issue reported in March 2016 revealed that the ASN.1 parser (specifically, d2i_ASN1_TYPE) can misinterpret a large universal tag as a negative zero value. These tags are not used in any common ASN.1 structures (for example X509, which is the certificate format used for the public keys in SSL) but are accepted as part of content in ANY structures. [6]

These two issues combined can allow an attacker to trigger the out-of-bounds write described in the first issue. This happens when an application deserializes untrusted ASN.1 structures that contain an ANY field and later reserializes them. The resulting memory corruption can possibly be exploited with some memory allocation (malloc) implementations. Therefore, applications that specifically parse and re-encode X509 certificates are known to be vulnerable. [6]

The reserialization cannot be triggered by any certificate, though. The certificates need to have valid signatures for that. As OpenSSL verifies the entire X509 certificate chain from root to
leaf, the certificates, that trigger the bug, must have been issued by trusted Certification Authorities. If that occurs, the trusted Certification Authorities have been compromised, which could be an indicator of a far larger issue. [3][6]

There is no exploit currently (publicly) available. [7]

**AVOIDING THE VULNERABILITY**

As the first bug was patched in April 2015 with OpenSSL version 1.0.2c and 1.0.1o, the issue cannot be exploited the way as specified by David Benjamin in March 2016 in these and later versions of OpenSSL. This means that the best solution to avoid the issue is keeping OpenSSL up to date. [5]

For the first issue, 2 minor code alterations were needed to address the vulnerability of handling zeroes as negative integers in `i2c_ASN1_INTEGER`. Specifically, the statement

```
if (BN_is_negative(bn) && !BN_is_zero(bn))
```

now makes sure that zeroes aren’t given the type `V_ASN1_NEG_INTEGER`. The other highlighted edit assures that a buffer underflow can no longer occur. [8]
For the second issue, three additional minor code alterations were needed to negate the vulnerability. The idea is to treat ASN1_ANY types as integers only if it has the V_ASN1_INTEGER tag. As a consequence, V_ASN1_NEG_INTEGER and V_ASN1_NEG_ENUMERATED tags are discarded in the switch statements. This results in the tag V_ASN1_NEG_INTEGER being an internal only value and is never used on the wire encoding. [10]

```
diff --git a/crypto/asn1/a_type.c b/crypto/asn1/a_type.c
index af79530..bb166e8 100644 (file)
--- a/crypto/asn1/a_type.c
+++ b/crypto/asn1/a_type.c
@@ -126,7 +126,7 @@ int ASN1_TYPE_cmp(const ASN1_TYPE *a, const ASN1_TYPE *b)
        /* They do not have content. */
        break;
    case V_ASN1_INTEGER:
-    case V_ASN1_NEG_INTEGER:
-    case V_ASN1_ENUMERATED:
-    case V_ASN1_NEG_ENUMERATED:
-    case V_ASN1_BIT_STRING:
-    case V_ASN1_OCTET_STRING:
-    case V_ASN1_SEQUENCE:
    break;

diff --git a/crypto/asn1/tasn_dec.c b/crypto/asn1/tasn_dec.c
index 9256049..2a13388 100644 (file)
--- a/crypto/asn1/tasn_dec.c
+++ b/crypto/asn1/tasn_dec.c
@@ -903,7 +903,7 @@ int asn1_ex_c2i(ASN1_VALUE **pval, const unsigned char *cont, int len,
            break;
    case V_ASN1_INTEGER:
    case V_ASN1_NEG_INTEGER:
-    case V_ASN1_ENUMERATED:
-    case V_ASN1_NEG_ENUMERATED:
    tint = (ASN1_INTEGER **)pval;
    if (!c2i_ASN1_INTEGER(tint, &cont, len))
        goto err;

diff --git a/crypto/asn1/tasn_enc.c b/crypto/asn1/tasn_enc.c
index f04a689..f7f83e5 100644 (file)
--- a/crypto/asn1/tasn_enc.c
+++ b/crypto/asn1/tasn_enc.c
@@ -511,7 +511,7 @@ int asn1_ex_i2c(ASN1_VALUE **pval, unsigned char *cont, int *putype,
            break;
    case V_ASN1_INTEGER:
    case V_ASN1_NEG_INTEGER:
-    case V_ASN1_ENUMERATED:
-    case V_ASN1_NEG_ENUMERATED:
/*
   * These are all have the same content format as ASN1_INTEGER
 */

OpenSSL source code
```
REFERENCES

2. https://www.openssl.org/
7. https://vuldb.com/?id.83254
8. https://git.openssl.org/?p=openssl.git;a=commitdiff;h=a0eed48d37a4b7beea0e966caf09ad46f4a92a44
9. https://git.openssl.org/?p=openssl.git;a=commitdiff;h=f5da52e308a6aee6d5f3df98ce4da295d7e9cc27