Paragon Initiative Enterprises

lcobucci/jwt

Source Code Audit
December 2015

I. Introduction

This document describes the security audit of Luís Cobucci's JWT library by Paragon Initiative Enterprises. This library was chosen by several members of the PHP community in response to an invitation to nominate libraries for a complementary source code review.

Our audit targeted git commit `1ae3426aba41b20fa67d13f61aaf4ec74f68edf3`, which was committed on December 20, 2015.

This report was prepared by Scott Arciszewski, CDO, and reviewed by Robyn Terjesen, CEO.

Audit Results Summary

We did not find any security vulnerabilities in the JWT library itself; however, we did find a previously undiscovered cryptographic vulnerability in one of its dependencies.

II. Audit Scope

We have limited the scope of this audit to focus specifically on the contents of the JWT source code library on Github with special attention to the usage of its phpecc/phpecc dependency, whose readme file states:

> **WARNING** Though this library is tested for compliance to standards, it is subject to at least one documented vulnerability in public-key derivation, which can potentially allow attackers to grab your private keys. **USE AT YOUR OWN RISK**. You've been warned.

III. Issues

1. PHPECC: ECDSA Verification is not Constant-Time

Users of this library can elect to use ECSDA signatures on their JSON Web Tokens, which is facilitated by a library called PHPECC. Whenever you write code to perform any sort of authentication in a cryptography context, it must be done in constant-time or else you leak information that could allow attackers to forge their own invalid tokens. Depending on what your application does with this data after signature verification, the damage could be severe.

What happens when you attempt to verify an ECDSA signature with PHPECC? The story starts in `Mdanter\Ecc\Crypto\Signature`:

```
return $math->cmp($v, $r) == 0;
```

The `$math` refers to an implementation of `MathAdapterInterface`, for which the only provided implementation is GMP. The verify method looks like this:

```
public function cmp($first, $other)
{
    return gmp_cmp(gmp_init($first, 10), gmp_init($other, 10));
}
```

If PHP's `gmp_cmp()` function is constant-time (and therefore suitable for signature verification), then this usage is perfectly safe. So let's next look at what `gmp_cmp()` does internally:

```
if (use_si) {
      res = mpz_cmp_si(gmpnum_a, Z_LVAL_P(b_arg));
} else {
      res = mpz_cmp(gmpnum_a, gmpnum_b);
}
```

This is defined in the GNU MP library, which invokes a compiler directive which finally references a macro for the actual comparison operation:

```
#define __GMPN_CMP(result, xp, yp, size)                                \
  do {                                                                  \
    mp_size_t  __gmp_i;                                                 \
    mp_limb_t  __gmp_x, __gmp_y;                                        \
                                                                        \
    /* ASSERT ((size) >= 0); */                                         \
                                                                        \
    (result) = 0;                                                       \
    __gmp_i = (size);                                                   \
    while (--__gmp_i >= 0)                                              \
      {                                                                 \
        __gmp_x = (xp)[__gmp_i];                                        \
        __gmp_y = (yp)[__gmp_i];                                        \
        if (__gmp_x != __gmp_y)                                         \
          {                                                             \
            /* Cannot use __gmp_x - __gmp_y, may overflow an "int" */   \
            (result) = (__gmp_x > __gmp_y ? 1 : -1);                    \
            break;                                                      \
          }                                                             \
      }                                                                 \
  } while (0)
```

As you can see by the explicit use of a loop and a `break` statement, this exits early as soon as a single value differs. This means the `gmp_cmp()` function is not constant-time and an alternative function should be used instead (such as `hash_equals()`).

We sent pull request #114 to the PHPECC project to remedy this defect.

V. Other Findings

Note: The findings in this section are not necessarily vulnerabilities.

1. Suggestion: Document the Risks of Forged Headers

As Tim McLean demonstrated in a guest post for Auth0, several JWT libraries are vulnerable to an attack where you change the algorithm from, e.g. RS256 to HS256, then use the RSA public key as an HMAC secret key to obtain a valid forged signature.

Although we did not find the same vulnerability, the documentation in the README file covers token signatures but does explicitly state that users **must not** automatically choose the `Signer` object based on the data contained within an attacker-controllable JWT header.

We sent pull request #64 to add this information to the README to discourage misuse. In addition, we sent pull request #63 to add a regression test to prevent this vulnerability from cropping up in future iterations of the library.

VI. Conclusion

In addition to being fully RFC 7519 compliant, the JWT library by Luís Cobucci does not itself contain any fatal design flaws that we were able to exploit.

Although it depends on an ECDSA implementation that we were able to identify an issue with, ECDSA is not mandatory.

We advise against the use of ECDSA signatures with this JWT library until the PHP ECC project can remedy the issue we identified (as well as other vulnerabilities identified by others before us).