Date: Sat, 2 Jul 2022 10:03:08 +0200
From: Salvatore Bonaccorso <carnil@...ian.org>
To: oss-security@...ts.openwall.com
Subject: Re: GnuPG signature spoofing via status line injection

Hi,

On Thu, Jun 30, 2022 at 02:18:33AM -0400, Demi Marie Obenour wrote:
> # Background
>
> After discovering that gpgv does not support
> --exit-on-status-write-error, I decided to check if it handles write
> errors on the status file descriptor properly. I ultimately found that
> while such errors are *not* handled properly, exploiting this flaw in
> practice would likely be very difficult and unreliable. However, in the
> course of this research (and entirely accidentally), I found that if a
> signature has a notation with a value of 8192 spaces, gpg will crash
> while writing the notation's value to the status FD. This turned out to
> be a far more severe flaw, with consequences including the ability to
> make a signature that will appear to be ultimately valid and made by a
> key with any fingerprint one wishes.
>
> # Prerequisites for exploitation
>
> For an attack to be possible, the attacker must control the secret part
> of at least one key in the victim's keyring. The key does *not* need to
> be trusted, however. Depending on the calling code, the attack may work
> even if the key is revoked or the signature is expired. However, if the
> program requires that *all* signatures be valid (instead of merely *any*
> signature being valid), then a revoked or expired key cannot be used.
>
> Additionally the code calling GnuPG must either not read status data
> until end of file, or satisfy both of the following:
>
> - It uses a lax parser that is tolerant of invalid status lines.
> - It does not treat a non-zero exit code from GnuPG as an error.
>
> It turns out that gpgme satisfies both requirements, so programs using
> gpgme are vulnerable. Since gpgme is the recommended way to use GnuPG
> from a program, I believe that the number of applications that are
> vulnerable is very large.
>
> # Impact
>
> If the attacker controls the secret part of any signing-capable key or
> subkey in the victim's keyring, they can provide a correctly-formed
> signature that some software, including gpgme, will believe to have a
> validity and signer fingerprint of the attacker's choosing. The
> consequences of this are highly application-dependent, but are likely to
> be serious. In an email client, this could allow spoofing emails, while
> in a system using key fingerprints for access control, this could allow
> for an access control bypass.
>
> # Solution
>
> I recommend cherry-picking upstream commit
> 34c649b3601383cd11dbc76221747ec16fd68e1b, which can be found at
> https://dev.gnupg.org/rG34c649b3601383cd11dbc76221747ec16fd68e1b.
> Afterwards, it will be necessary to rebuild and reinstall GnuPG. No
> security advisory has been issued by upstream, no patch release is
> planned, and no CVE has (to my knowledge) been requested. Distributions
> will need to carry this as an out-of-tree patch until the next upstream
> release is made. For those using GnuPG on Windows, the only solution
> will be to build from source.
>

> This does not fix the handling of write errors on the status file
> descriptor. However, I believe that exploiting the mishandling of such
> errors is not feasable in general. On the other hand, the out of bounds
> read can be reliably exploited.
>
> # Proof of concept
>
> I have attached a public key, a revoked version of that key, and two
> signatures made by the key. Both signatures are of the empty string;
> you can pass /dev/null if the program takes a file instead.
> simple-exploit-sig.asc will not work if the key is revoked or expired,
> while revoked-exploit-sig.asc *may* work even if the key is revoked or
> expired.
>
> # Details
>
> ## The bug
>
> GnuPG does not provide an OpenPGP or S/MIME library. Instead, gpg,
> gpgv, and gpgsm all support writing machine-readable text to a
> user-provided file descriptor, which is set via the --status-fd
> command-line argument. Other programs and libraries then parse this
> output to extract information about what GnuPG has done.
>
> In the case of gpg and gpgv, all status output goes through one of the
> functions in g10/cpr.c. The one of interest here is
> write_status_text_and_buffer(), of which the relevant part is reproduced
> below.
>
> ```
> 356  do
> 357    {
> 358      if (dowrap)
> 359        {
> 360          es_fprintf (statusfp, "[GNUPG:] %s ", text);
> 361          count = dowrap = 0;
> 362          if (first && string)
> 363            {
> 364              es_fputs (string, statusfp);
> 365              count += strlen (string);
> 366              /* Make sure that there is a space after the string.  */
> 367              if (*string && string[strlen (string)-1] != ' ')
> 368                {
> 369                  es_putc (' ', statusfp);
> 370                  count++;
> 371                }
> 372            }
> 373          first = 0;
> 374        }
> 375      for (esc=0, s=buffer, n=len; n && !esc; s++, n--)
> 376        {
> 377          if (*s == '%' || *(const byte*)s <= lower_limit
> 378              || *(const byte*)s == 127 )
> 379            esc = 1;
> 380          if (wrap && ++count > wrap)
> 381            {
> 382              dowrap=1;
> 383              break;
> 384            }
> 385        }
> 386      if (esc)
> 387        {
> 388          s--; n++;
> 389        }
> 390      if (s != buffer)
> 391        es_fwrite (buffer, s-buffer, 1, statusfp);
> 392      if ( esc )
> 393        {
> 394          es_fprintf (statusfp, "%%%02X", *(const byte*)s );
> 395          s++; n--;
> 396        }
> 397      buffer = s;
> 398      len = n;
> 399      if (dowrap && len)
> 400        es_putc ('\n', statusfp);
> 401    }
```
```

> 402 while (len);
>
> When writing the data of a notation subpacket, GnuPG requests that
> write_status_text_and_buffer() wrap the output at 50 bytes if the
> notation is marked as human-readable, or 250 bytes otherwise. 'buffer'
> points to the (unsanitized) notation data, and 'length' is the length of
> that data. For the subsequent discussion, I will only consider
> human-readable notations. Adapting the exploit to use binary notations
> is easy and is left as an exercise for the reader.
>
> If byte 50 needs escaping, esc will be set to 1 on line 379, causing the
> loop to exit. Line 388 will undo the effect of the s++, n-- on line
> 375, but this will in turn be undone by line 395. Therefore, line 397
> will increase `buffer` by 50.
>
> Now suppose the next byte also needs escaping. This time, line 380 will
> break out of the loop, so the s++, n-- on line 375 will be skipped.
> However, the s--; n++ on line 388 will still run, so s is now one *less*
> than buffer. Subtracting them will thus return -1, which becomes
> SIZE_MAX when converted to size_t. As a result, es_fwrite() will try to
> write the rest of the address space to the status stream, starting with
> byte 51 of the notation data.
>
> ## Exploitation
>
> The result of the bug is that es_fwrite() will write bytes to the status
> stream (with no escaping) until it hits unmapped memory and segfaults.
> The first bytes written, in particular, come from the notation data
> itself. Therefore, they are fully controlled by the attacker. The only
> restriction is that the first byte must be one that needs to be escaped,
> but this turns out to be no restriction at all.
>
> Suppose that the the first byte injected is a newline. At this point,
> the status stream is at the start of a line, and the attacker can append
> any bytes of their choice to it. A good choice for the attacker would be:
>
> [GNUPG:] VALIDSIG $subkey_fpr $date $timestamp 0 4 0 22 10 00 $primary_key_fpr
> [GNUPG:] TRUST_ULTIMATE 0 pgp
>
> Here $subkey_fpr should be replaced with the desired subkey fingerprint,
> $date with the desired signing date, $timestamp with the desired
> timestamp, and $primary_key_fpr with the desired primary key
> fingerprint. Obviously, the fingerprints can be those of *any* key, or
> even ones (such as 0000000000000000000000000000000000000000) that do not
> correspond to a real key. TRUST_ULTIMATE tells the calling program that
> the key is ultimately valid.
>
> Following the notation data, gpg will write a bunch more garbage from
> its heap before it eventually segfaults. This garbage is not valid
> status data, but it turns out that many programs do not care. Git stops
> at the first NUL byte and gpgme ignores any line that does not start
> with "[GNUPG:] ". Hence, this does not prevent exploitation.
>
> # Timeline
>
> - 2022-06-10: Message sent to security@...pg.org requesting encryption
>   keys for subsequent communication.
>
> - 2022-06-10 through 2022-06-11: Message with encrypted subjects sent to
>   GnuPG Security Team. These messages are automatically discarded by
>   Werner Koch's email account.
>
> - 2022-06-12: Message with unencrypted subject sent and received.
>
> - 2022-06-13: Response asking for a specific case where a transient I/O
>   error can happen, and acknowledging that the out-of-bounds read is
>   real. Bug is not considered critical and so no immediate security
>   release is planned.
>
> - 2022-06-13: I respond mentioning ENOMEM and socket errors as potential
>   transient write errors.
>
> - 2022-06-14: Werner Koch commits 34c649b3601383cd11dbc76221747ec16fd68e1b
>   to the GnuPG git repository. From this commit, ticket T6027, and the
>   test signature attached to T6027, it is easy to reverse-engineer the

>   bug and create an exploit. There is no public mention that this is a
>   security problem.
>
> - 2022-06-15: I followed up stating that it may be possible to control
>   the contents of the out-of-bounds memory and that this would make the
>   bug much more severe.
>
> - 2022-06-17: Werner responds stating that he has doubts as to whether
>   this can be done easily, and noting that GPGME still needs to accept
>   the injected data.
>
> - 2022-06-17: I state that I am able to inject arbitrary data into the
>   status output, and that the only reason Git is not vulnerable is
>   because GnuPG eventually segfaults.
>
> - 2022-06-18: I state that I can make GPGME mark a signature as "valid
>   green" (the highest trust level) with whatever fingerprint I wish.
>
> - 2022-06-19: Werner replies stating that he is not able to reproduce
>   the injection of arbitrary data into the status output, though he can
>   reproduce improper escaping.
>
> - 2022-06-19: I state that the flaw is indeed less severe in git master.
>
> - 2022-06-19: Via `git bisect`, I discover that
>   34c649b3601383cd11dbc76221747ec16fd68e1b is in fact the commit that
>   fixed the vulnerability, and that arbitrary injection into the status
>   line is possible on the immediately preceeding commit
>   4dbef2addca8c76fb4953fd507bd800d2a19d3ec. I provide a reproducer.
>
> - 2022-06-22: I request that this be marked as a security vulnerability
>   and have a CVE assigned, and that an immediate security release be
>   made. I note exactly what an attacker who exploits this vulnerability
>   can do to a program relying on gpgme.
>
> - 2022-06-29: As Werner Koch has stopped replyng to my emails, and since
>   there is still no public indication that GnuPG has a security
>   vulnerability (despite the patch already being public), I am publicly
>   disclosing the issue.

CVE-2022-34903 is assigned for this issue.

Cf. https://www.cve.org/CVERecord?id=CVE-2022-34903

Regards,
Salvatore