Message-ID: <0009d6cc-143e-41e6-b240-eb526a9cb306@gmail.com>
Date: Sun, 28 Dec 2025 00:47:30 -0600
From: Jacob Bachmeyer <jcb62281@...il.com>
To: oss-security@...ts.openwall.com, Solar Designer <solar@...nwall.com>
Cc: contact@....fail
Subject: Re: Many vulnerabilities in GnuPG

On 12/27/25 22:27, Solar Designer wrote:
> On Sat, Dec 27, 2025 at 07:29:53PM -0500, Demi Marie Obenour wrote:
>> https://gpg.fail lists many vulnerabilities in GnuPG, one of which
>> allows remote code execution. All are zero-days to the best of
>> my knowledge.
> Thanks. I wish this were brought in here by the researchers, but since
> it was not and since we require actual content here (not just links),
> let me take care of this now. The website has it nicely formatted, so I
> also include the HTML versions, which brings the message to just below
> the maximum of 1 MiB here. Who knows how long this website will stay
> up, but oss-security archives will probably exist decades later.
>
> The website currently says:
>
>>   Slides, pocs and patches soon!
>>
>>   "in the hurry of leaving i forgot the sites src at home, sorry, had to
>>   rewrite the whole thing. expect a nicer site by tomorrow. im patching as
>>   we speak."
>>   - crackticker (<- to blame)
>>
>>   1. Multiple Plaintext Attack on Detached PGP Signatures in GnuPG
>>   2. GnuPG Accepts Path Separators and Path Traversals in Literal Data
>>   "Filename" Field
>>   3. Cleartext Signature Plaintext Truncated for Hash Calculation
>>   4. Encrypted message malleability checks are incorrectly enforced causing
>>   plaintext recovery attacks
>>   5. Memory Corruption in ASCII-Armor Parsing
>>   6. Trusted comment injection (minisign)
>>   7. Cleartext Signature Forgery in the NotDashEscaped header
>>   implementation in GnuPG
>>   8. OpenPGP Cleartext Signature Framework Susceptible to Format Confusion
>>   9. GnuPG Output Fails To Distinguish Signature Verification Success From
>>   Message Content
>>   10. Cleartext Signature Forgery in GnuPG
>>   11. Radix64 Line-Truncation Enabling Polyglot Attacks
>>   12. GnuPG may downgrade digest algorithm to SHA1 during key signature
>>   checking
>>   13. GnuPG Trust Packet Parsing Enables Adding Arbitrary Subkeys
>>   14. Trusted comment Injection (minisign)
> Each of the above 14 vulnerabilities has its own web page. I attach 14
> text (converted with ELinks at width 80) and 14 HTML files corresponding
> to them.

These are definitely edge cases and I think that item 9 is actually old
news.

Comments on a first reading:

Overall, I have one very major point of disagreement here: the OpenPGP
clearsign format is useful precisely because it enables the signed
message to be easily viewed with other tools, thus complicating attacks
and making large-scale attacks much more likely to be detected. (It
only takes one user who looks at a clearsigned digest list with less(1)
and sees a bunch of control sequences to raise an alarm.)

Item 1: Multiple Plaintext Attack on Detached PGP Signatures in GnuPG

Exploitation requires a very odd use of signatures. Bob knows that he
has a detached signature, and ordinarily a detached signature is simply
used to verify the signed file, after which the signed file is used
directly.

This is also the first time I have seen --decrypt suggested for reading
a signed message; does this also work if the message is encrypted?

Mallory can be assumed to have Bob's public key and could therefore
generate a second encrypted message to Bob if Alice's message was signed
and encrypted. There would be an interesting indicator of compromise:
an abnormally-large detached signature.

This exposes *another* bug in GPG: documentation says that --decrypt
will reject an unencrypted input, yet the PoC involves using it with
exactly that.

Item 2: GnuPG Accepts Path Separators and Path Traversals in Literal
Data "Filename" Field

While this is a potentially serious bug, as it enables an attacker to
potentially overwrite any file if the attacker can guess the file name,
it also relies more on a social-engineering attack. While a naive user
might use the suggested command, a more-experienced user should
immediately smell a rat.

The PoC uses ANSI escapes to cover up (erase, move left) the hash mark
that makes the fake message viable as a shell script, the actual payload
command (set as invisible text), and the prompt from GPG about writing
the output file (which appears to be sent to the window title). I am
uncertain how the user is supposed to see the next prompt, as the
invisible text mode does not appear to be reset.

Item 3: Cleartext Signature Plaintext Truncated for Hash Calculation

GPG uses in-band signaling if a line is too long. Comments in the GPG
sources acknowledge that this is a hack.

The exploit does have a problem in that it reports an error about
"invalid armor" due to a line being too long. Also, while it works if
the message is fed directly to a terminal, I suspect that reading the
message with less(1) would show interesting anomalies.

Item 4: Encrypted message malleability checks are incorrectly enforced
causing plaintext recovery attacks

This item admits not actually having a complete attack, and I am unsure
how exactly this leads to recovering plaintext, even if Bob cooperates.
(User willingness to sign a random key received by email without
verifying it breaks Web-of-Trust badly.)

If there is a bug here, it seems to be an out-of-bounds read in GPG's
Inflate implementation: the decrypted ciphertext is in the memory buffer
prior to the altered packets and thus prior to the beginning of the
compressed stream.

I am unsure how the garbled comment packet is produced, unless it is the
result of interpreting the decrypted data as a compressed stream and
"inflating" it.

Item 5: Memory Corruption in ASCII-Armor Parsing

This is a serious memory-safety error in GPG.

Item 6: Trusted comment injection (minisign)

This is really a terminal-manipulation trick: the fake trusted comment
must be longer in order to fully obscure the original text after the
inserted CR moves the cursor back to the left edge. Again, using almost
anything other than cat(1) to read the file will either show the
chicanery or at least raise the proverbial eyebrow.

```
Item 7: Cleartext Signature Forgery in the NotDashEscaped header
implementation in GnuPG

This is a misfeature that probably should not have been implemented, or
should have been implemented much more strictly.

Item 8: OpenPGP Cleartext Signature Framework Susceptible to Format
Confusion

Another logical solution to this issue would be to recognize when
processing something that looks like a clearsigned message and reject a
one-pass signature if a clearsigned message header has been seen.

Item 9: GnuPG Output Fails To Distinguish Signature Verification Success
>From Message Content

I think this is actually an old problem, previously affecting
Thunderbird and Git IIRC, and the existing --status-fd mechanism in GPG
is meant for exactly this case, at least for automated processing.

Item 10: Cleartext Signature Forgery in GnuPG

This is simply an implementation defect in GPG and should be fixed by
improving the validation of the "Hash" header.

Item 11: Radix64 Line-Truncation Enabling Polyglot Attacks

This is another implementation defect in GPG, although fixing it may be
more involved: the radix64 reader should be refactored to work by
characters (or arbitrary blocks) instead of by lines.

In fact, arbitrary limits are generally contrary to the GNU Coding
Standards, so this really should be fixed... :-)

Item 12: GnuPG may downgrade digest algorithm to SHA1 during key
signature checking

The root of this is another out-of-bounds read. There is a simple fix to
this: always, *always*, *ALWAYS* initialize stack-resident local variables.

I am also unsure about the actual insecurity of SHA1 in general. Have
there been more attacks since the first actual collision?

Item 13: GnuPG Trust Packet Parsing Enables Adding Arbitrary Subkeys

Keyrings are trusted stores, so this is more of a documentation problem.

The report is right that caching signature checks is probably a bad
idea, although it may have been justifiable in the past due to limited
computing power.

Item 14: Trusted comment Injection (minisign)

This is closely related to item 6, except that this time minisign itself
outputs the sequences that manipulate the terminal.


-- Jacob
```