

SMS of Death

A single malformed WAP Push SMS crashes `com.android.phone` on Android, dropping active calls (including emergency calls), with zero interaction from the victim.

 Download APK

 Source

Google Issue Tracker

⚠ Working proof-of-concept, authorised testing only. [Read the warning & license](#) before downloading.

COMPONENT	INTERACTION	VECTOR	VENDOR STATUS
WapPushOverSms frameworks/opt/telephony	None zero-click, silent	Remote SMS WAP Push, port 2948	<div>• Won't Fix</div> classified Non-Security
REPORTER	REPORTED	DISCLOSED	CVE
Mark Blaszczyk independent	5 May 2026 Android & Devices VRP	29 Oct 2026 after 90-day timeline	None no CVE issued

01 / The vulnerability

An unchecked cast turns a 5-byte header into a negative array size

When Android receives a WAP Push message, `WapPushOverSms.decodeWapPdu()` reads the WSP header length as a `uintvar`. The value comes back as a `long`, is cast to `int` with no range check, and is then used directly as the size of a new byte array:

```
// WapPushOverSms.java:217 header length read as long, cast with no bounds check
int headerLength = (int) pduDecoder.getValue32();

...

// WapPushOverSms.java:244 the cast value is used as an array size
byte[] header = new byte[headerLength];
```

A 5-byte `uintvar` of `88 80 80 80 00` decodes to `0x80000000`. Cast to a signed 32-bit `int`, that is `-2147483648`. Allocating `new byte[-2147483648]` throws an uncaught `NegativeArraySizeException` on the telephony main thread, killing `com.android.phone`:

```
E AndroidRuntime: FATAL EXCEPTION: main
E AndroidRuntime: Process: com.android.phone
E AndroidRuntime: java.lang.NegativeArraySizeException: -2147483648
E AndroidRuntime: at ...WapPushOverSms.decodeWapPdu(WapPushOverSms.java:244)
E AndroidRuntime: at ...WapPushOverSms.dispatchWapPdu(WapPushOverSms.java:339)
E AndroidRuntime: at ...InboundSmsHandler.processMessagePart(InboundSmsHandler.java:1169)
E AndroidRuntime: at ...InboundSmsHandler$DeliveringState.processMessage(InboundSmsHandler.java:62)
```

The message is delivered as a data SMS and is never shown to the user: there is nothing in the inbox, and the victim never sees the sender's number.

02 / Impact

Remote, silent, and recoverable only with effort

- Zero-click.** The victim does nothing. No tap, no open, no notification.
 - Silent.** Nothing appears in Messages; the attacker's number is never shown.
 - Calls drop, including emergency calls.** Active voice calls terminate with `DisconnectCause_CS_DEATH` as the process restarts.
 - Telephony blackout.** 1-2 seconds of no service per message while `com.android.phone` restarts.
 - Carrier redelivery amplifies it.** Unacknowledged messages are re-sent by the carrier, re-crashing the phone; a single message caused intermittent outages for up to a week in testing.
 - ~20 messages** render the device unusable for an extended period.
 - Persistent denial of service.** Rapid repeated delivery corrupts the SQLite database behind SMS storage, turning a momentary crash into a phone that crashes on every boot (detailed below).
- Tested across multiple makes and models (Pixel 9 on April 2026 firmware, various Samsung Galaxy and A-series) with a 100% reproduction rate.
- From a 2-second crash to a bricked phone: the SQLite race
- A single message is a repeated transient outage. Sending the payload rapidly in succession (~30 messages) is materially worse: the crashes interleave with the telephony provider's writes to the SMS store and hit a **race condition that corrupts the SQLite database** holding the device's messages.
- Once the database is corrupt, `com.android.phone` no longer needs a fresh message to fail: it **crashes immediately on every boot** as it reads the damaged store. The phone is now persistently without telephony, and a restart does not clear it.
- Recovery is hostile, and this is the critical part: a **factory reset only helps temporarily**, because the carrier re-sends the unacknowledged payload and re-triggers the crash. Full recovery required:
- Removing the physical SIM and inserting it into a **non-Android device** (e.g. an iPhone) so the carrier's retried messages are acknowledged and stop being redelivered;
 - Then clearing the SMS records / corrupt database over **ADB**.
- On a device using an **eSIM**, where the SIM can't simply be moved to another handset, recovery would be significantly harder, and for a non-technical user, effectively out of reach.
- This is the distinction the report turns on. It is not a transient crash: it is a **remote, zero-click, persistent denial of service requiring a factory reset (and more) to recover**, which is the exact shape of a "Destructive Remote DoS" in the VRP's own rules.

03 / Why this is serious

A remote attacker can cut a phone off from Triple Zero

Dropping calls is not an inconvenience when one of those calls is to emergency services. While a device is under this attack, or bricked by the SQLite corruption, it cannot reliably place a call to **000**. The attack is remote, needs only the victim's number, and shows them nothing.

Australia treats access to emergency calling as a mandatory property of a mobile service, not a nice-to-have. Under the [Telecommunications \(Emergency Call Service\) Determination](#), administered by the ACMA, carriers are required to **identify mobile phones that cannot reach Triple Zero, help the customer move to a working device, and stop providing service to the affected handset**. [Telstra states](#) plainly that it is required to block phones that can no longer call 000, notifying the customer and blocking the device if nothing is done within 28 days.

The consequence is perverse: the regulatory remedy for a phone that can't reach 000 is to **cut it off the network**, not to patch the OS. So an attacker who can remotely induce that condition on an Android handset can, in effect, get the victim's own device barred from carrier service until a fix exists. The regulation exists precisely because loss of emergency-call access is considered a public-safety failure; this bug manufactures that failure on demand.

The framework was [tightened in 2024](#) following the 3G shutdown, which left some phones able to make [ordinary calls but](#) unable to reach emergency services, the same functional state this vulnerability produces.

Sources: [ACMA: proposed amendments to the ECS Determination](#) · [Telstra: Emergency Call Service Determination](#) · [Dept. of PM&C: impact analysis](#)

04 / The fix

One bounds check

The remedy is a range check before the cast in `WapPushOverSms.java` (line 217), provided to Google on the day of the first report:

```
long rawHeaderLen = pduDecoder.getValue32();
if (rawHeaderLen < 0 || rawHeaderLen > pdu.length) {
    result.statusCode = Intents.RESULT_SMS_GENERIC_ERROR;
    return result;
}
int headerLength = (int) rawHeaderLen;
```

05 / Disclosure timeline

Communication with Google

The report was filed through the Android & Google Devices VRP and resubmitted twice as a mis-triage appeal. Across three tickets and four engineers, Google classified it as a non-security issue.

Researcher	Android Security Team	Status change
5 May 2026 Researcher		Report filed: zero-click WAP Push crash in <code>WapPushOverSms</code> (issue 509520497). Suggested fix provided the same day.
5 May 2026 Android Security		Assigned for investigation; researcher asked to follow coordinated disclosure.
15 May 2026 Researcher		Requests higher priority: reproduced on 5 handsets, emergency-call impact, Australian carrier regulations.
20 Jun 2026 Android Security		Closed Won't Fix (Infeasible) : "not a security vulnerability."
20 Jun 2026 Researcher		Clarifies this is a remote permanent DoS; ~20 messages permanently disable the phone. Resubmits as appeal (issues 525810560 and 525816147, "SMS of DEATH").
20 Jun 2026 Android Security		Resubmit 525810560 closed within 3 minutes Won't Fix (Not Reproducible) .
22 Jun 2026 Researcher		Reports SQLite corruption from rapid sends → crash on boot; recovery needs SIM swap to a non-Android device + ADB.
23 Jun 2026 Android Security		Requests disclosure plans; reminds of the 90-day VRP timeline and asks to review any write-up in advance.
23 Jun 2026 Researcher		Agrees to hold for public safety; offers to brief Australian carriers so they can schedule a block. Restates the one-line fix.
2-3 Jul 2026 Android Security		Appeal marked Duplicate of the original; re-evaluation confirms same root cause.
16 Jul 2026 Android Security		Original reopened after the issue surfaced on Discord, then confirmed a duplicate again.
16 Jul 2026 Researcher		Adds detail: the behaviour meets the VRP's own "Destructive Remote DoS" criterion.
17 Jul 2026 Android Security		Acknowledges the extra context; triage to take another look: "4 engineers will have evaluated the report."
27 Oct 2026 Researcher		Requests a status update after three months of silence.
29 Oct 2026 Android Security		Final decision: Won't Fix , quoting "does not meet the severity threshold ... considered a Non-Security Issue (NSI)."

Why this is public. The issue was reported privately and held well beyond the VRP's 90-day window. Google declined to fix it across three tickets and four engineers, classifying a remote, zero-click, emergency-call-dropping denial of service as a non-security issue. Full disclosure follows so that users, carriers and downstream OEMs can assess the risk and mitigate.

06 / Proof of concept

Source & APK

The proof-of-concept Android app and full source code are published on GitHub. The app crafts the malformed WAP Push PDU and sends it as a data SMS to a target number.

Warning: this is a working weapon. The repository contains a proof-of-concept that can remotely disrupt and brick mobile phones. It is published strictly for security research, education, and defensive testing.

Only run it against devices you own or are explicitly authorised to test. Using it against anyone else is a crime, and interfering with a person's ability to reach emergency services (000) is a serious criminal offence in Australia and most countries.

The software is provided "as is", with no warranty, under the [Apache License 2.0](#). **The author accepts no liability for any use, misuse, or damage.** By downloading the APK or the source, you accept full responsibility for what you do with it.

Download APK

Source on GitHub

