PHOENIX SECURITY

MongoBleed (CVE-2025-14847)

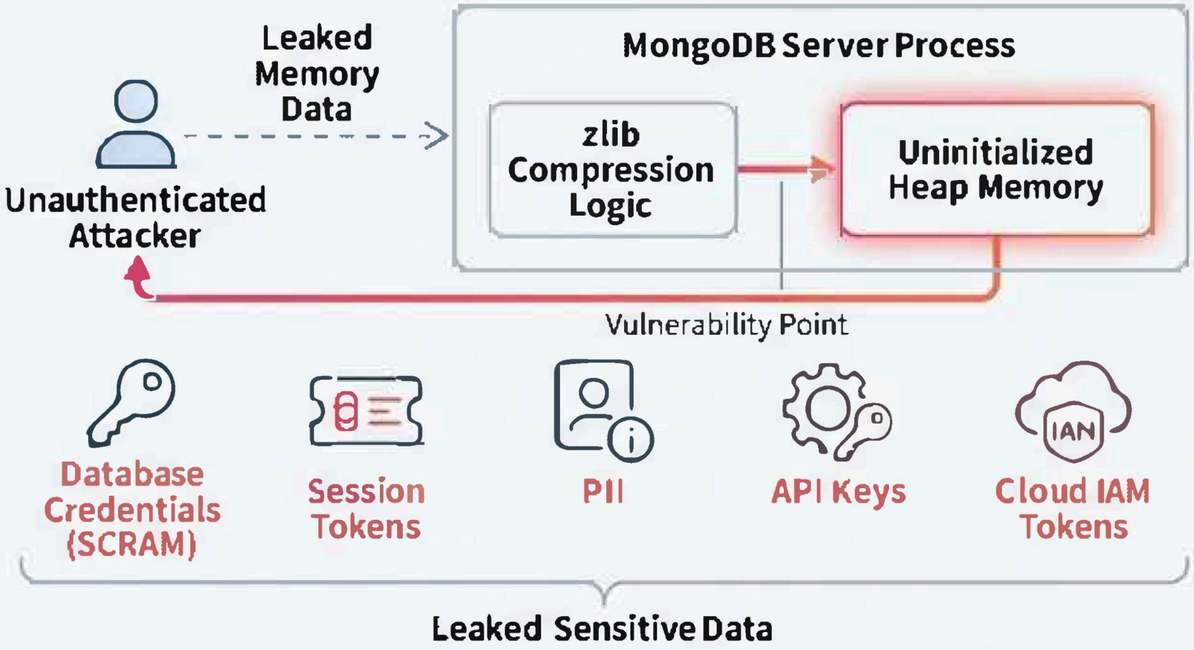
Unauthenticated Memory Disclosure in MongoDB Network Transport

Phoenix Security Research

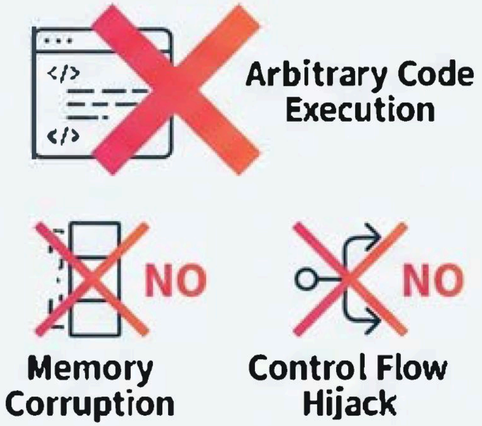
The Vulnerability: Unauthenticated Memory Disclosure

What it is: A flaw in MongoDB's zlib compression logic allows an unauthenticated attacker to read uninitialized heap memory from the server process.

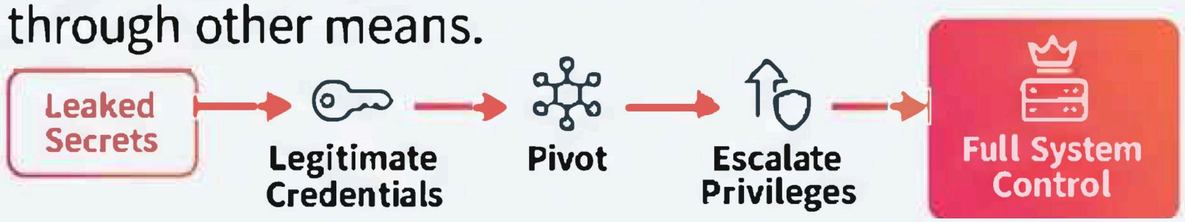
Why it's dangerous: The leaked memory can contain sensitive data: database credentials (SCRAM), session tokens, PII, API keys, and cloud IAM tokens.



It is not a classic RCE: The vulnerability does not allow direct, arbitrary code execution. There is no memory corruption or control flow hijack.



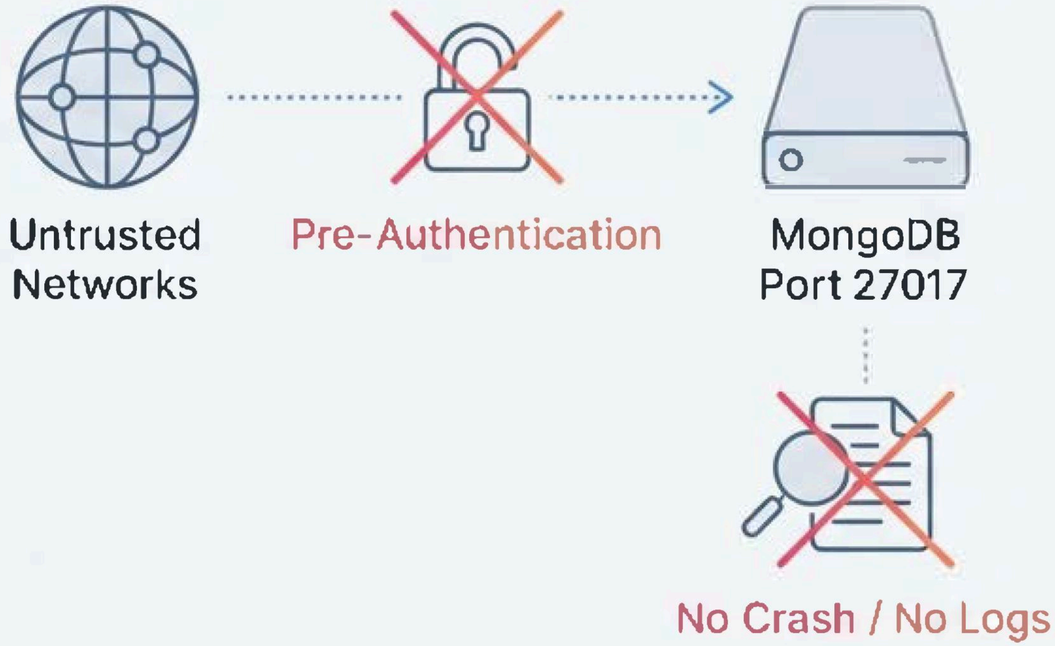
It enables real-world compromise: Leaked secrets provide attackers with legitimate credentials to pivot, escalate privileges, and achieve full system control through other means.



PHOENIX SECURITY
Phoenix Security Research

Impact Overview: Silent, Pre-Authentication Data Leakage

- **Unauthenticated Trigger:** Exploitation occurs during network compression negotiation, *before* any authentication checks are performed.
- **Heap Memory Disclosure:** The server returns fragments of its own process memory, containing whatever data was recently used or stored there.
- **Silent Exploitation:** The exploit does not crash the server or generate obvious error logs, making detection difficult. Attackers can probe repeatedly.
- **High-Risk Exposure:** Any MongoDB instance with port 27017 exposed to untrusted networks is immediately vulnerable. Censys data shows over 87,000 internet-facing instances.



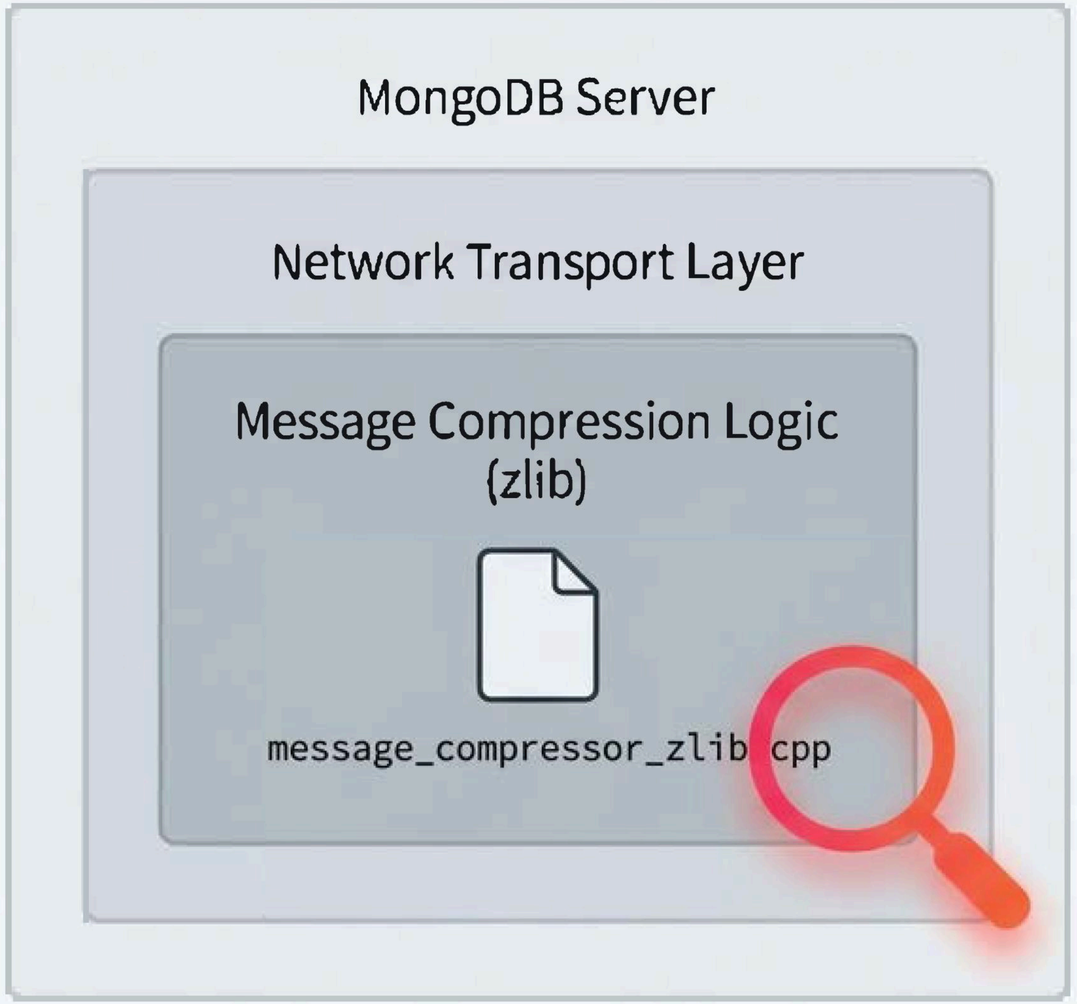
Affected Versions: Broad Impact Across Modern and Legacy Deployments

- **MongoDB 8.2.x:** 8.2.0 – 8.2.2
- **MongoDB 8.0.x:** 8.0.0 – 8.0.16
- **MongoDB 7.0.x:** 7.0.0 – 7.0.27
- **MongoDB 6.0.x:** 6.0.0 – 6.0.26
- **MongoDB 5.0.x:** 5.0.0 – 5.0.31
- **MongoDB 4.4.x:** 4.4.0 – 4.4.29
- **Legacy (End-of-Life):** All versions of 4.2, 4.0, and 3.6 are vulnerable and will not receive receive patches.

PHOENIX SECURITY
Phoenix Security Research

Where the Vulnerability Lives: A Trust Failure in the Transport Layer

- **Component:** The vulnerability resides in the MongoDB wire protocol's network transport layer.
- **Process:** Specifically, it is in the server-side logic that handles compressed client messages.
- **Trigger Path:** The flaw is triggered during the decompression of messages using the `zlib` library.
- **File Reference:** The faulty logic is located in `src/mongo/transport/message_compressor_zlib.cpp`.
- **Core Issue:** The server incorrectly trusts attacker-controlled metadata within the compressed message header.



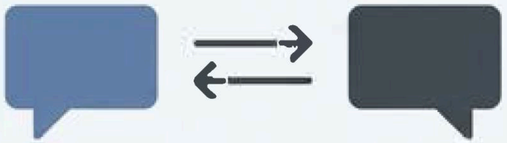
Technical Root Cause: Misinterpretation of Decompression Length

- **The Lie:** An attacker sends a compressed message declaring a large `uncompressedSize` in the header but with a small actual compressed payload.
- **The Trust:** The server allocates a large heap buffer matching the declared `uncompressedSize`.
- **The Flaw:** The zlib decompression wrapper incorrectly returns `output.length()`, which is the size of the large *allocated buffer*, not the smaller number of bytes actually written by the decompressor.
- **The Consequence:** The calling function receives a pointer to the large buffer and a length indicating it's full. Subsequent BSON parsing then reads beyond the valid decompressed data into uninitialized heap memory.

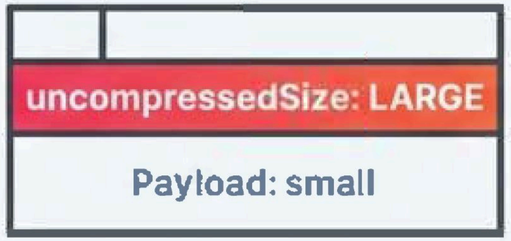
```
76      counterHitDecompress(input.length(),
output.length());
77  -   return {output.length()};
77  +   return length;
78    }
```

PHOENIX SECURITY
Phoenix Security Research

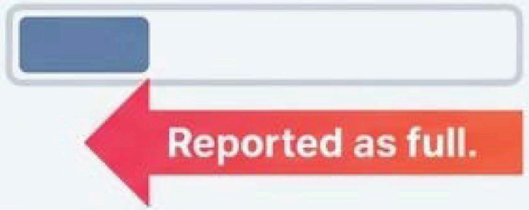
Exploit Trigger: Crafting the Malicious Request



1 Step 1: Negotiation:
The client initiates a connection and negotiates `zlib` as the network message compressor.



2 Step 2: The Payload:
The client sends a wire protocol message with an oversized `uncompressedSize` value (e.g., 64KB) and a small but valid `zlib`-compressed payload.

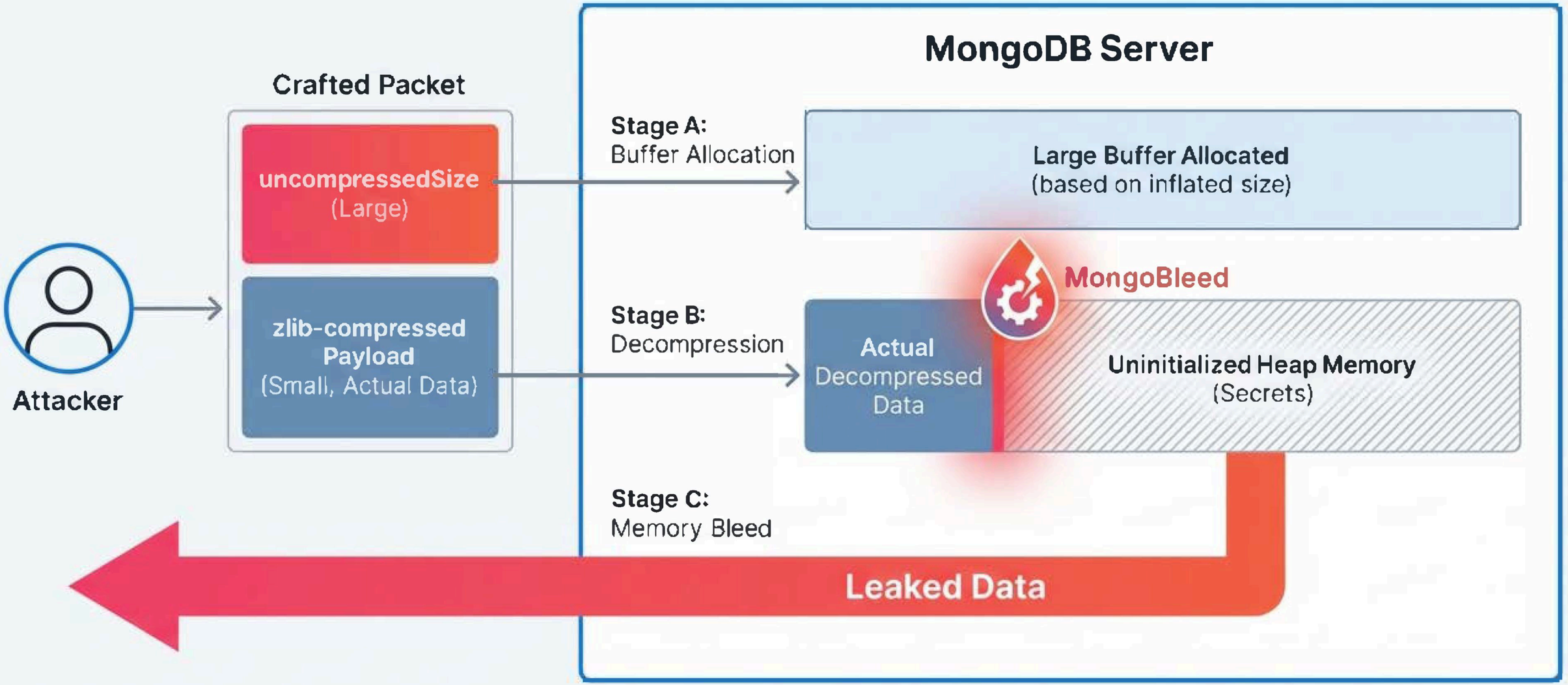


3 Step 3: The Bleed:
The server allocates 64KB, decompresses the small payload into it, but incorrectly reports that 64KB of data is valid.

4 Step 4: The Read:
The server's BSON parser attempts to process the message, reading the valid data followed by adjacent, uninitialized heap memory, which is then sent back to the attacker.

PHOENIX SECURITY
Phoenix Security Research

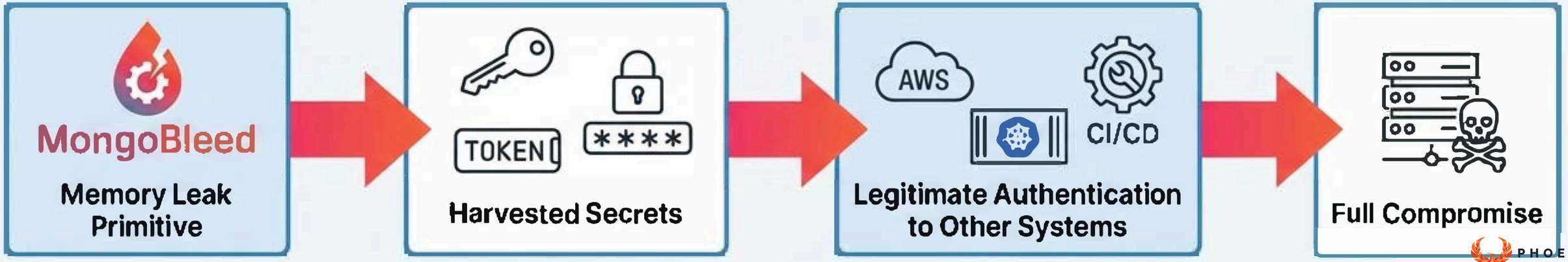
Exploit Anatomy: Visualizing the Memory Bleed



PHOENIX SECURITY
Phoenix Security Research

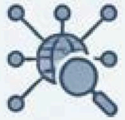
Why This Is RCE-Adjacent: From Memory Disclosure to Full Compromise

- **No Direct Execution:** MongoBleed is a primitive for reading memory, not writing or executing it.
- **Memory Contains Keys to the Kingdom:** Process memory is a rich target, often containing plaintext credentials, session cookies, API keys, and cloud metadata tokens.
- **Enabling Lateral Movement:** An attacker uses leaked secrets to authenticate legitimately to other systems. This is not an exploit; it is authorized access.
- **Cloud & CI/CD Impact:** A leaked AWS IAM key or Kubernetes token from memory can grant control over entire infrastructure stacks—far more impactful than a single shell.
- **Disclosure as the First Step:** A reliable, unauthenticated info disclosure is often more dangerous than a complex RCE, as it provides the material for simple, high-impact privileged access.



Detection Signals: Hunting for Quiet Exploitation Attempts

Network-Level



- Look for repeated, short-lived connections to port 27017 from a single source IP that never authenticate.
- Monitor inbound `zlib`-**compressed messages** with a large declared `uncompressedSize` but small payload.

Host-Level Anomalies

- Watch for the `mongod` process exhibiting **elevated CPU or memory allocation spikes** without a corresponding query load, as parsing garbage memory costs cycles.

Post-Exploitation (Credential Misuse)

- **The highest fidelity signal.** Alert on sudden, unexpected authentication events using credentials that could have been leaked.
- Specific Examples: Logins from new IPs, use of cloud credentials outside their expected context, or access to sensitive admin commands by normally dormant accounts.

Patch and Fix: Upgrading Is the Definitive Remediation

Patched Versions ✅

- 8.2.3+
- 8.0.17+
- 7.0.28+
- 6.0.27+
- 5.0.32+
- 4.4.30+

Explanation of Fix

- **What the Fix Does:** The patch changes the zlib decompression wrapper to `return length;`, ensuring the function reports the *actual* number of bytes decompressed, not the allocated buffer size.
- **Added Validation:** New regression tests (CheckUndersize) were added to explicitly fail if if the decompressed data length is less than the declared size, preventing this bug class from recurring.

Compensating Controls: Risk Reduction if Patching is Delayed

Disable zlib Compression

This is the most effective immediate mitigation. Reconfigure the server to remove `zlib` from the `networkMessageCompressors` list.

Use Safer Alternatives

If compression is required, switch to `snappy` or `zstd`, which are not affected by this specific implementation flaw.

Restrict Network Exposure

Ensure MongoDB instances are not exposed to the internet. Enforce strict firewall rules allowing access only from trusted application servers.

Assume Breach and Rotate

If an instance was exposed while vulnerable, assume its memory was compromised. Proactively rotate all credentials, API keys, and tokens.

Exposure at Scale: Why So Many Instances Are At Risk

Default Configurations

Network compression featuring `zlib` is a common and often default configuration valued for performance.

Accidental Internet Exposure

Misconfigured security groups, containers without network policies, and legacy firewall rules frequently lead to unintended public exposure of database ports.

Cloud-Native Blind Spots

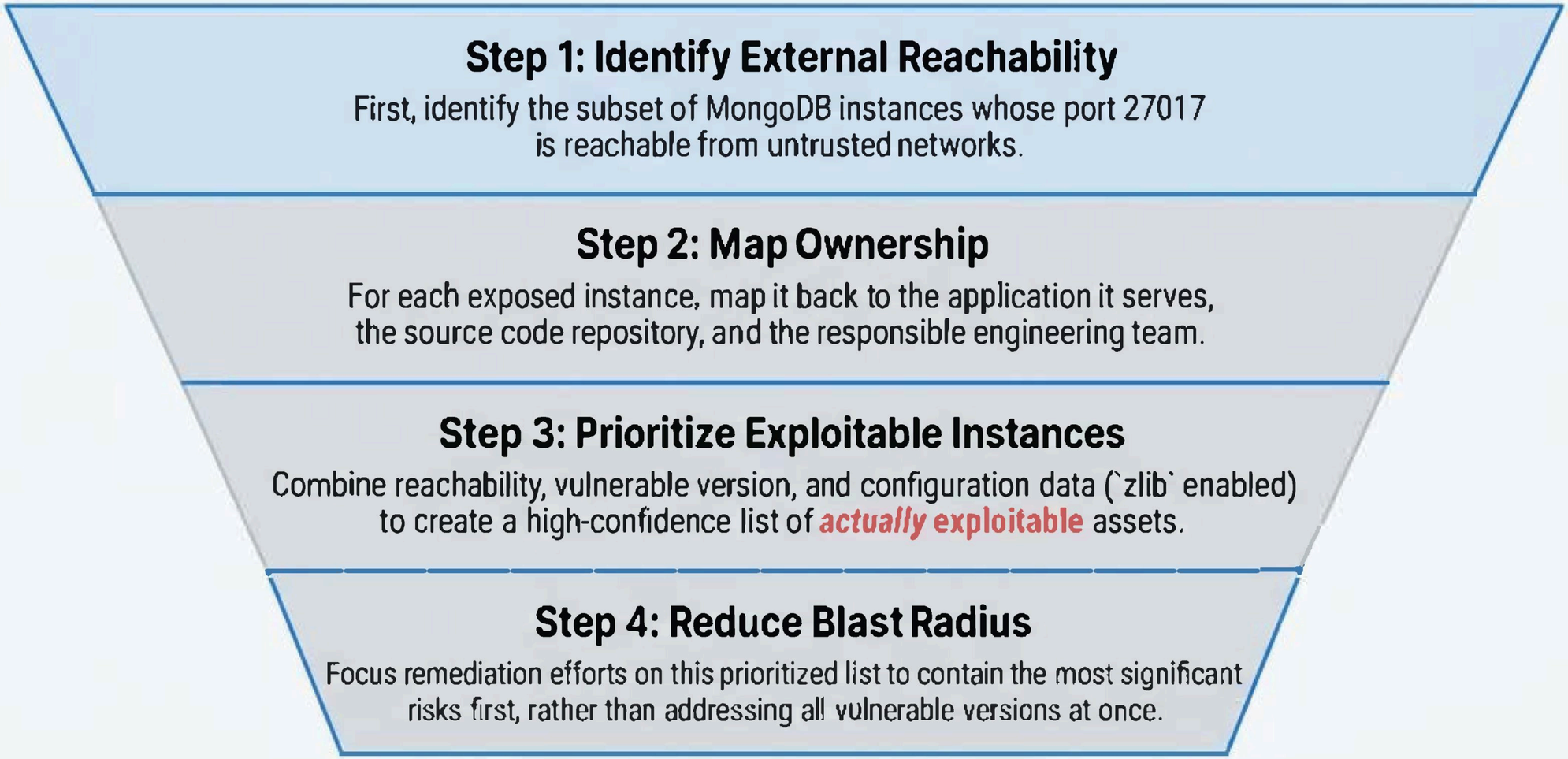
The dynamic nature of cloud deployments can make tracking network paths and true exposure difficult without dedicated tooling.

Scanning Is Insufficient

A simple port scan confirms reachability but not exploitability; the server must also have `zlib` enabled in its configuration.

PHOENIX SECURITY
Phoenix Security Research

Phoenix-Style Exposure and Attribution: From Vulnerability Count to Actionable Risk



Incident Response Guidance: Actions for Exposed and Compromised Systems

- [ ] **Assume Memory Was Read:** If an instance was vulnerable and exposed, operate under the assumption that an attacker has read fragments of its process memory.

- [ ] **Immediate Credential Rotation:** Rotate all secrets associated with the host: MongoDB users, cloud IAM roles/keys, service account tokens, and environment variables.

- [ ] **Audit Downstream Systems:** Scrutinize logs of connected systems (applications, cloud control planes, CI/CD pipelines) for signs of **credential abuse** originating from the time of exposure.

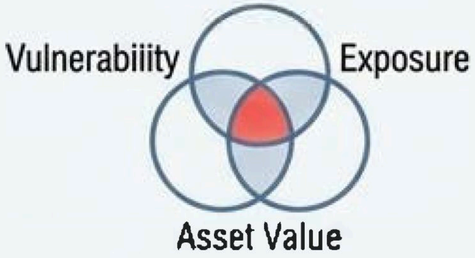
- [ ] **Patch and Isolate:** Apply the required patch immediately and **validate network controls** to permanently remove direct internet exposure.

PHOENIX SECURITY
Phoenix Security Research

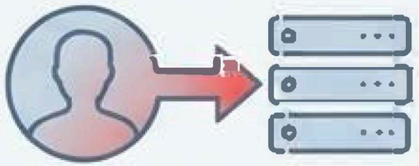
Key Takeaways: Core Lessons from MongoBleed

Trust-Boundary Failure: This is a classic vulnerability where a server incorrectly trusts attacker-controlled metadata (the `uncompressedSize` field).

Attacker-Controlled Metadata is Dangerous: Any data from the client, even metadata, must be validated before influencing memory allocation or control flow.



Exposure Context Defines Real Risk: A vulnerability's CVSS score is abstract. Its true risk is a function of its network reachability, configuration, and the value of the data it protects.



Ownership Enables Fast Containment: Knowing which team owns an exposed asset is critical for rapid response. A vulnerability without an owner is a breach waiting to happen.