Catalogued but Dormant:
A Honeypot Measurement of Residual Memcached Amplification Abuse

Observing scan and test, but not execute, on a public Memcached honeypot

Maxim Hájek[1]
Supervisor: Harm Griffioen[1]

[1]**EEMCS, Delft University of Technology, The Netherlands**

A Thesis Submitted to EEMCS Faculty Delft University of Technology,
In Partial Fulfilment of the Requirements
For the Bachelor of Computer Science and Engineering
June 21, 2026

Name of the student: Maxim Hájek
Final project course: CSE3000 Research Project
Thesis committee: Harm Griffioen, Mitchell Olsthoorn

Abstract

The 2018 Memcached amplification attacks set the public DDoS record, after which UDP was disabled by default and port 11211 widely firewalled. Yet tens of thousands of misconfigured servers remain reachable, leaving open the question of whether, and how, attackers still exploit this residual surface. We address it with a custom, safety-bounded Memcached honeypot deployed on 16 public IPv4 addresses for 22 days. Over the window it recorded 4,261 packets from 465 sources. Traffic is dominated by `stats` reconnaissance bearing recognisable tooling fingerprints, and 23 sources swept the full address block: the *scan* and *test* phases of the established workflow are clearly present. The *execute* phase, however, never materialised: no source populated the cache, empirical amplification capped at $\approx 13\times$, and the only abuse-related traffic was neutralised. The honeypot egressed just 277 KiB in total. Memcached today behaves as a catalogued but dormant amplifier: continuously enumerated and re-measured, but rarely weaponised.

1 Introduction

Distributed Denial-of-Service (DDoS) attacks remain one of the most disruptive threats on the public internet. Among the techniques available to attackers, *UDP-based reflection and amplification* stands out for its leverage: by spoofing the source address of a small request, an attacker tricks a vulnerable intermediary into sending a much larger response to the victim. Memcached, a protocol never designed to be exposed to the internet, holds the current public amplification record. In February 2018 a Memcached-based attack against GitHub peaked at 1.35 Tbit/s, with measured amplification factors of up to roughly $51{,}000\times$ [1]. After that event the maintainers disabled UDP by default and operators were urged to firewall port 11211. The widespread perception is that the Memcached problem has been solved. In practice, internet-wide scans still find tens of thousands of misconfigured Memcached deployments reachable over UDP [2], and the protocol therefore continues to be of operational interest to adversaries searching for amplifiers. Whether, and how, attackers currently exploit this residual attack surface is the question that motivates this work.

A substantial body of research has characterised the amplification DDoS ecosystem. Krämer et al. [3] pioneered honeypot-based observation of reflectors with AmpPot, demonstrating that emulated services attract real attack traffic and can be used to fingerprint booter services and victims. Jonker et al. produced a longitudinal, macroscopic view of DoS activity by fusing honeypot telemetry with darknet backscatter, showing that a sizeable fraction of the internet is targeted on any given day [4]. More recently, Griffioen et al. [5] used a distributed honeypot deployment to reconstruct adversarial workflows: the *scan-test-execute* pattern in which attackers first discover reflectors, then probe them to gauge their amplification factor, before finally weaponising them in a campaign. Continued measurement work confirms that amplification remains a live problem and that the reflector population shifts as old protocols are mitigated and new ones are weaponised [6].

What remains *unanswered* is what this picture looks like specifically for Memcached in the post-patch era. Existing studies that cover Memcached either predate or briefly follow the 2018 events. They treat it as one of many UDP protocols rather than as a focused case study. As a result, we have limited public evidence on (a) whether scanners still actively probe for misconfigured Memcached servers, (b) what request shapes they use, text or binary protocol, which commands, in which order, and (c) whether the *scan/test/execute* model derived from older protocols still describes Memcached attacker behaviour today.

Research questions. This work asks one main research question and four sub-questions:

RQ. To what extent is the Memcached protocol still abused for DDoS amplification in the post-patch era, and which tactics, techniques, and procedures (TTPs) characterise the adversaries that engage with it?

Q1. How do attackers discover and probe vulnerable Memcached servers?

Q2. Under what conditions can Memcached be exploited to enable an amplification attack?

Q3. What features distinguish Memcached attack traffic from benign scanning or measurement traffic?

Q4. What is the typical attacker workflow, from first probe to attempted weaponisation?

The remainder of the paper is organised as follows. *Section 2* provides the necessary background on Memcached, UDP amplification, and honeypot-based measurement. *Section 3* describes the design and safety architecture of the honeypot and the data-collection methodology. *Section 4* discusses the responsible research considerations specific to operating an active deception system on the public internet, which frame the conditions under which the data was collected. *Section 5* analyses the observed traffic with respect to the four research questions. *Section 6* relates the findings back to prior related work, reflects on their limitations, and outlines directions for future work. Finally, *Section 7* concludes the paper.

2 Related work

This section reviews the literature on UDP-based amplification DDoS attacks, the role of honeypots in studying them, and the specific case of the Memcached protocol. It motivates the methodological choices that follow: why use honeypots to answer the research question, and what design constraints prior work has established for deploying one safely.

2.1 Reflection-amplification DDoS attacks

A reflection-amplification attack abuses a connectionless, UDP-based service in two steps. The attacker sends a small request to a publicly reachable server while spoofing the

source address of the intended victim. The server, having no way to validate the requester, sends its (typically much larger) reply to the victim. When the response is significantly larger than the request, the technique is called amplification. Many widely deployed UDP services have been weaponised this way over the past decade, including DNS, NTP, CLDAP, SSDP, and Chargen [7; 8].

The macroscopic scale of this ecosystem has been documented repeatedly. Jonker et al. [4] built a framework combining honeypot data, internet-wide darknet observations, and DDoS Protection Service measurements to characterise the global victim population, finding millions of distinct attack targets over their measurement window and demonstrating that DDoS-as-a-service platforms had dramatically expanded the population of potential perpetrators. More recently, Hiesgen et al. [6] compared academic and industry vantage points on the DDoS landscape and found that, despite mitigation efforts, reflection amplification attacks remain a persistent and evolving threat, one of the two dominant attack classes alongside direct path attacks. Crucially for this project, they confirm that legacy amplification protocols continue to appear in attack traffic long after the underlying vulnerability is publicly known and "patched" in the popular sense.

2.2 The Memcached amplification vector

Memcached is an in-memory key-value store designed for use within trusted data center networks to cache expensive database queries [9]. Until version 1.5.6 (released 2018-02-27) it shipped with a UDP listener enabled by default on port 11211 and no authentication [10]. That combination is what made it so potent: an unauthenticated, high-throughput UDP service whose `stats` and `get` commands can return very large responses to a spoofed victim.

The vulnerability was disclosed in late February 2018 and exploited within days. On 28 February, GitHub absorbed what was then the largest recorded DDoS attack: 1.35 Tbit/s via 126.9 million packets per second from over a thousand autonomous systems [1] with a worst-case amplification factor of $\approx 51{,}000\times$, turning a request of a few tens of bytes into a reply several hundred kilobytes in size. A second attack, peaking at 1.7 Tbit/s against a US service provider, followed within days [11]. Mitigation guidance recommended disabling UDP entirely (`-U 0`), binding to localhost, or blocking/rate-limiting traffic to UDP source port 11211.

Despite the patch, misconfigured internet-exposed servers persist, because the patch only changed the default: existing deployments, and any operator who explicitly enables UDP, remain exposed. This is the gap we target. Even after the consensus that "Memcached is fixed," does the threat ecosystem still treat it as a viable amplifier worth scanning for and weaponising?

2.3 Honeypots as a tool

A honeypot is a deliberately exposed system whose value lies not in the service it provides, but in the interactions it records. Because no legitimate client has any reason to contact it, every connection is, by construction, a scan, a probe, or an attack attempt. This suits the study of reflection amplification ecosystems, where reconnaissance and weaponisation both involve unsolicited UDP traffic. A honeypot that convincingly emulates a vulnerable amplifier receives precisely that traffic, without the signal-to-noise problems of passive measurement on production networks.

The foundational work is AmpPot [3], which emulates a range of amplification protocols, responds like a genuinely vulnerable server, and rate-limits its replies to avoid aiding ongoing attacks. Over seven months Krämer et al. recorded more than 1.5 million attacks against over 600,000 victims and fingerprinted individual booter services by clustering on request patterns, timing, and command sequences. Their design set two principles subsequent work has adopted: responses must be realistic enough to keep the adversary engaged through the full scan-test-execute cycle, and hard limits must prevent the honeypot from becoming a real amplifier itself.

Later work extended this. Thomas et al. [12] used a distributed deployment to trace booter infrastructure, showing that the same scanner sources and tooling recur across protocols. Griffioen et al. [5] used a global honeypot network to reconstruct the adversarial workflow itself: the *scan, test, execute* pattern in which attackers first sweep the address space for responsive hosts, then return with crafted probes to measure each host's amplification factor, and only then weaponise the validated reflectors. This model defines what we should see on a Memcached honeypot if the protocol is still weaponised. Not only indiscriminate scans, but follow-up amplification-factor probes and, ultimately, requests whose source addresses are spoofed victim IPs rather than the scanner's own.

A recurring theme is the tension between attractiveness and safety: a honeypot that answers too readily or too largely risks being recruited into a real attack, while one that answers too sparingly or with obviously synthetic data is filtered out in the testing phase and never observes the later stages. Krämer et al. [3] addressed this with per-source rate limits and capped response sizes; Griffioen et al. [5] additionally separated emulation logic from network exposure so volumes could be tuned without changing protocol behaviour. Hiesgen et al. [6] note that even well rate-limited honeypots disturb the ecosystems they observe, and recommend explicitly accounting for a honeypot's own outbound contribution. These constraints directly shape the design in Section 3.

3 Methodology and Experimental setup

This work is an observational study: we cannot ethically launch real DDoS attacks. Instead we place an emulated, deliberately misconfigured Memcached service on the public internet and record what reaches it. We emulate the protocol rather than run an unmodified `Memcached -U 11211` binary for two reasons. *Safety*: a real UDP-enabled server is itself an amplifier, so exposing one would force a choice between contributing to live attacks and firewalling egress so aggressively that scanners notice and leave. A custom emulator instead serves plausible replies under a strict bandwidth ceiling while suppressing amplification. *Observability*: a real

server treats traffic as a workload and records neither source-port, parse-error, nor per-source rate information in structured form. Our own parser lets the same packet flow that drives responses also drive a structured event stream that analysis can query directly.

3.1 Honeypot Architecture

The honeypot (Figure 1) is implemented in Go and structured as a small pipeline that processes each incoming UDP datagram. The pipeline has four stages, each in its own package so it can be unit-tested in isolation:

1. **Listener**. Binds a UDP socket and uses `IP_PKTINFO` to recover the destination IP each datagram was sent to. This is required because the VM is assigned multiple honeypot IPs and binding to `0.0.0.0` would otherwise lose the per-IP attribution that we want for cross-IP scan-pattern analysis.
2. **Parser**. Decodes the payload as either the Memcached text protocol (ASCII commands, optionally prefixed with the 8-byte UDP frame header) or the binary protocol (magic byte `0x80`/`0x81`). The parser is intentionally permissive: malformed packets are not rejected silently but recorded with a `parse_error` field, because *how* attackers malform requests is itself a fingerprint signal.
3. **Responder.** Builds plausible text-protocol replies for the commands that scanners actually exercise (Table 1). Recognised commands receive realistic responses; `stats` values (uptime, item counts, byte totals) are drawn from the config file so different honeypot IPs can present distinguishable "personalities" without code changes. Two responses are deliberately inert for safety: `get`/`gets` always returns a cache miss (`END`) and `set`/`add`/`replace` acknowledges with `STORED` but *discards* the value. The honeypot is therefore stateless: no attacker can populate the cache, which is the precondition for high amplification (subsection 5.2). The binary protocol is parsed and logged but answered only with minimal skeleton responses.
4. **Safety gate and sinks**. Every prospective reply passes through the rate limiter and the safety policy described in subsection 3.2 before being sent. Independently of whether a reply is sent, every received packet generates an event that is fanned out to all configured sinks.

A deliberate consequence of this split is that the responder does *not* decide whether to transmit anything. Decisions about transmission are concentrated in the safety gate, which is the only piece of code that can place bytes on the wire. This minimises the surface area across which a future change could accidentally bypass the rate limit.

3.2 Layered Safety Design

Because the honeypot is meant to attract the exact traffic that would turn a real Memcached server into an amplifier, its safety design is the most load-bearing part of the methodology. We adopt five independent layers, so that a bug or misconfiguration in any one of them does not by itself enable abuse.

Table 1: Implemented command surface of the Responder. Parsed commands are logged regardless; the right column is what the honeypot actually emits.

Command	Reply	Behaviour
`version`	`VERSION x`	from config
`stats`	`STAT...END`	fake personality
`get`/`gets`	`END`	always miss (no amplification)
`set`/`add`/`replace`	`STORED`	value discarded
`delete`	`NOT_FOUND`	
`flush_all`	`OK`	
`quit`	—	no reply
unknown / malformed	`ERROR`	text protocol

1. **Per-source byte budget.** A token bucket keyed by source IP, denominated in *bytes* (not packets), with up to 10^5 tracked sources. The byte-denomination is intentional: a packet counter would let a scanner extract amplification by sending small requests that produce large replies.
2. **Global outbound bandwidth cap.** A continuously-refilling token bucket capped at 1 Mbit/s across *all* outbound traffic, with a small burst budget (100 kbit) so legitimate probe flurries are answered without obvious starvation. This is the primary safety invariant: regardless of how many spoofed-source packets arrive, the honeypot cannot deliver a meaningful amplification volume to any victim.
3. **Same-port reflection drop.** If a packet's source port is 11211, no reply is sent. Legitimate Memcached clients never use the server port as their source; same-port traffic is therefore almost certainly a reflection attempt and we refuse to relay it.
4. **Attack-prep PPS threshold.** A simple per-source packets per-second ceiling triggers a reply drop without suppressing logging. This catches obvious burst test traffic before the global cap even engages.
5. **Silent-mode kill switch.** A single configuration flag (`silent_mode: true`) halts all outbound replies while leaving the listener and sinks running. It is intended to be the first switch flipped if an abuse complaint or anomaly is observed in the data-collection window.

The five layers live in the honeypot's own source tree. Their combined effect is summarised informally as: "even if everything we wrote is wrong, the host cannot exceed 1 Mbit/s outbound traffic", preventing any proper contribution to a DDoS attack.

3.3 Data Collection Pipeline

Every incoming packet generates one structured `Event`, whether or not a reply is sent. Each record holds the source/destination tuple, protocol variant, recognised command and keys, parse errors, raw payload (hex-encoded), whether a reply was sent, the response size, the amplification factor (response/request bytes), and the rate-limit state at decision time. Storing both "what we observed" and "what

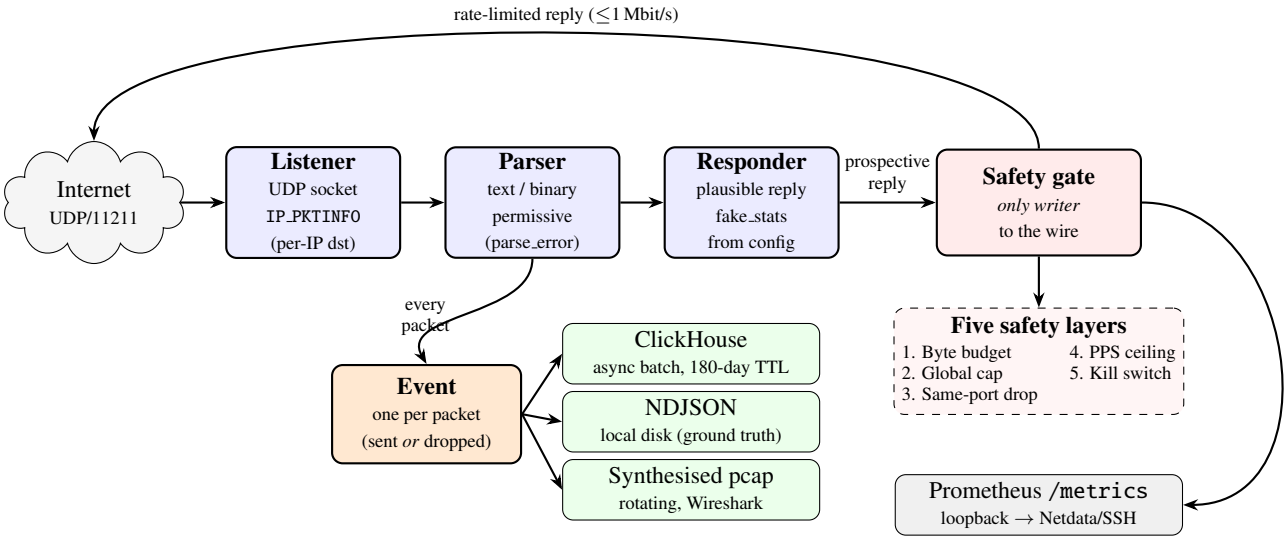

Figure 1: Honeypot architecture: a four-stage UDP pipeline (Listener, Parser, Responder, Safety gate) in which only the safety gate writes to the wire, behind five safety layers and a 1 Mbit/s cap. Every packet emits an Event to three sinks (ClickHouse, NDJSON, pcap), with metrics on a loopback-only Prometheus endpoint.

we decided" lets the analysis separate behaviour the honeypot shaped (e.g. dropped replies) from behaviour intrinsic to the source.

Events fan out to three parallel sinks:

- **NDJSON** on local disk, one line per event: the durable ground truth, always written so no event is lost if other sinks fail, and the format used for offline replay.
- **Synthesised pcap.** Rather than a raw libpcap capture (which would require `CAP_NET_RAW` and a libpcap dependency), we synthesise the Ethernet/IPv4/UDP headers from the recorded tuple and write rotating pcaps that open cleanly in Wireshark; payloads are preserved exactly. The trade-off: no TTL or IP-ID fingerprinting (this is out of scope here).
- **ClickHouse.** An asynchronous batched sink into a `MergeTree` table with a 180-day TTL, chosen over a row store for the aggregation-heavy analysis (top sources, command distributions, amplification histograms, per-source timing). If unreachable, only this sink drops events, which can be re-ingested from the NDJSON ground truth.

Operational counters (packets by variant, drop reasons, bytes egressed, amplification histograms, rate-limiter levels) are exposed on a Prometheus `/metrics` endpoint, bound to loopback (`127.0.0.1:9090`) and scraped by Netdata over an SSH tunnel so internals are not leaked to scanners.

3.4 Deployment and Data-Collection Window

The honeypot is deployed on a university-provided VM[1] with a set of 16 public IPv4 addresses. First boot is performed in silent mode with `tc` configured but no replies emitted, so that logging, pcap rotation, and ClickHouse ingest can be validated against real internet background traffic before any reply is sent. Silent mode is only disabled once these checks pass.

The data collection window for this paper is approximately 22 days, running continuously between d_{start}=`2026-05-29` and d_{end}=`2026-06-19`. During this window the honeypot is left untouched. No responder behaviour, fake-stats content, or rate-limit parameter is changed mid-experiment, so that the observed traffic distribution is not partially shaped by our own intervention.

3.5 Analysis Approach

Analysis runs offline against the collected NDJSON and ClickHouse data after the window closes; the honeypot does no online classification. Each sub-question maps to one analysis step:

- **Q1 (discovery/probing):** per source, count the distinct honeypot IPs hit, their time ordering, and inter-packet timing: a rapid sweep of all IPs signals an internet-wide scanner, a delayed return to one IP a target-list lookup.
- **Q2 (exploitation conditions):** compute the empirical amplification factor (response/request bytes) following the bandwidth amplification factor (BAF) metric of Rossow [7], bounding how attractive a misconfigured server is today.
- **Q3 (attack-vs-benign separation):** summarise each source by a set of signals (protocol variant, UDP-header presence, command, and the fractions of same-port and malformed packets) and check whether attack-

[1] A single virtual machine was used for the entire deployment, with the following specifications: 8 vCPUs (Intel Xeon Gold 6430), 7.7 GiB RAM, and 123 GB disk, running Ubuntu 24.04.4 LTS (Linux kernel 6.8.0, x86_64) under KVM. It carries 16 advertised honeypot IPs, one management IP, and the loopback address; traffic counts in this paper aggregate all datagrams reaching the host across these addresses.

preparation traffic separates cleanly from ordinary scanning.

- **Q4 (attacker workflow):** order each source's interactions chronologically and test the *scan, test, execute* hypothesis [5]: discovery probe, size-test probe, then repeated large-reply requests with spoofed source addresses. Classifying matching timelines as the canonical workflow.

3.6 Limitations of the Design

Three limitations are inherent to the design and should be borne in mind when reading the results.

First, the honeypot's emulation is *approximate*: protocol parity with a reference Memcached server is validated for the commands that matter to amplification but not for the full command surface. A scanner that uses an obscure command we mishandle may either fingerprint us as a honeypot or move on, biasing the observed sample toward less discriminating (less competent) adversaries.

Second, the 1 Mbit/s cap necessarily *distorts attacker behaviour*. A scanner that expects an immediate reply but receives one delayed by the bucket may give up before its full probing sequence completes. We attempt to mitigate this by sizing the burst budget for realistic probe flurries, but the cap is non-negotiable on safety grounds.

Third, the IPs we observe are necessarily *a small sample* of the internet's reflector population. Findings should be read as evidence of which attacker behaviours *exist*, not as unbiased estimates of their global frequency.

4 Responsible research

Active measurement of unsolicited internet traffic is a long-standing practice in network security research, and the deception employed here is narrowly targeted: only hosts that send unsolicited Memcached requests to our advertised IPv4 addresses receive a response, so no legitimate user or uninvolved third party is misled. The emulated service exposes no real data and offers no functionality beyond what is required to characterise the prober's behaviour. We consider this proportionate to the scientific value of measuring an attack surface that is already actively scanned in the wild.

4.1 Harm minimisation

Because a Memcached reflector is itself a potential DDoS amplifier, the honeypot is constrained by the layered safety architecture described in subsection 3.2, with the silent-mode kill switch serving as the operational backstop should any single mechanism fail.

4.2 Personal data

Captured events include the source IP address of each prober. Under Dutch law, a public IP address on its own cannot be linked to a natural person without a judicial order compelling the responsible internet service provider, and we hold no such linking information. The recorded IPs are therefore not, in our hands, personal data within the meaning of the GDPR. We nevertheless treat the data with the safeguards that would apply if it were: events are stored as NDJSON and synthesised pcap on local disk, and in ClickHouse with a 180 day TTL after which they are deleted. Access is restricted to the researcher and supervisor, the metrics endpoint binds to loopback only, and dashboards are reachable only through an authenticated SSH tunnel to the VM. No attempt is made to re-identify scanners by joining against external datasets that could close the identification gap.

4.3 Reproducibility and disclosure

The honeypot source code is published [13] to support reproducibility. Raw event data is not released, as it contains third-party IP addresses. This work observes existing misconfigured deployments and does not disclose any new vulnerability or release offensive tooling.

4.4 Use of Generative AI

In the interest of transparency, this section discloses the use of generative AI tools in this project. Three large language models (Gemini 3.5 Flash; GPT-5.5; Claude Sonnet 4.6) were used as support tools in the following roles: (i) editing and rephrasing author written text for grammar and clarity; (ii) generating boilerplate LaTeX (table and figure scaffolding, including the architecture diagram in Figure 1); and (iii) drafting Python scripts used to aggregate the collected event logs and render the figures.

The use of these tools complies with the university's standards for academic integrity, and all material derived from external sources is cited. The tools served only as productivity aids and the work is fully reproducible without them. No part of the project required a paid or account-gated tool. No raw event data, which contains third-party IP addresses, was entered into any of these tools.

5 Observed Traffic

Over the collection window the honeypot recorded **4,261** events from **465** distinct source IP addresses, distributed across the honeypot host's addresses (Table 2). Of these, **1,570** packets were answered and **2,691** were withheld by the safety gate or because no reply was defined for the request. The honeypot emitted a total of **283,732** bytes (≈277 KiB) over the entire window, an average outbound rate below 1.19 bit/s. This is nearly six orders of magnitude under the 1 Mbit/s ceiling of the rate limiter. We report the data against the four sub-questions.

5.1 Q1. Discovery and probing

The honeypot scanning was indiscriminate with respect to the honeypot address. The sixteen advertised honeypot addresses were hit at broadly similar volumes (between 137 and 179 packets each). This corresponds to a signature expected of an internet wide sweep that enumerates a contiguous block rather than of a target list lookup that revisits a single known host. Twenty-three distinct sources touched 15 or more of the honeypot addresses, and together these full block sweepers accounted for 2,789 packets and 925 of the replies sent.

The population is otherwise strongly long-tailed (Table 3): 201 of the 465 sources sent a single packet and a further 173

Table 2: Traffic overview, 2026-05-29 to 2026-06-19.

Total events	4,261
Distinct source IPs	465
Honeypot addresses receiving scan traffic	16
Replies sent	1,570
Bytes egressed	283,732
Peak per-source packet rate	26 pps
Protocol variant	
Text protocol	3,224 (76%)
Non-protocol / unrecognised	1,037 (24%)
Binary protocol	0 (0%)
Disposition of received packets	
Reply sent	1,570
Dropped: same-port reflection	1,646
Dropped: no reply for command	1,027
Dropped: silent mode (first-boot check)	18
Dropped: rate limit / global cap / PPS	0

Table 3: Source population by packet count. Most sources are one-shot; the two highest-volume sources (including the same-port reflection source of Section 5.4) dominate by volume.

Packets per source	# sources	# packets
1	201	201
2–5	173	481
6–20	69	684
21–100	20	786
>100	2	2,109

sent between two and five, consistent with background radiation and one shot reachability probes. Persistent scanners were rare, but regular. The most active benign scanner returned 146 times at a median inter-arrival of 410 s across the whole window.

A clear discovery fingerprint is present in the request encoding. **95%** of text-protocol packets (3,064 of 3,224) carried the 8-byte Memcached UDP frame header, and the request id field was overwhelmingly the constant value `0` (2,266 packets from 235 sources), with `1` (563 packets, 24 sources) and a single fixed value `23117` (178 packets, 8 sources) the only other common choices. The request id field is itself a coarse tool fingerprint, since different scanning tools populate it differently. The dominant value 0 is consistent with scanners that leave the UDP frame header zeroed, while a distinct subpopulation using request id 1 (563 packets) reproduces the exact header of the widely circulated 2018 Memcrashed proof of concept, and a third fixed value (23117, 8 sources) marks a separate tool.

5.2 Q2. Conditions for exploitation

The only recognised command that scanners issued in volume was `stats`: 1,408 `stats` requests from 267 distinct sources, of which 1,387 were answered. Under the honeypot's configured `fake_stats` personality, each `stats` reply was 203 bytes against a typical 14-15 byte request, giving an empirical amplification factor with a mean of **13.4**× and a maximum of **14.5**× (Table 4). No other command exceeded 2.5×, and the unrecognised/malformed traffic produced replies of at most 15 bytes (an `ERROR` response).

Table 4: Disposition of received packets by command. *Answered* rows give the amplification (response/request bytes); *not-answered* rows give the safety rule that suppressed the reply. Counts sum to the 4,261 packets of Table 2. No `set`, `get` (different from `gets`), `add`, `delete`, `version`, or binary-protocol request was observed.

Command / class	Packets	Mean	Max
Answered (reply sent: 1,570)			
`stats`	1,387	13.4×	14.5×
`gets`	5	0.7×	0.7×
Other (malformed → `ERROR`)	178	<1×	2.5×
Not answered (suppressed: 2,691)			
`flush_all` (same-port drop)	1,621	—	—
Non-protocol / unrecognised	1,027	—	—
Same-port / silent-mode (misc.)	43	—	—

This empirical factor is two to three orders of magnitude below the ≈51,000× that made Memcached notorious in 2018. The record setting factor does not come from `stats`, rather it comes from **retrieving a large value that the attacker first stored via a `set` followed by a spoofed source `get`**. The decisive precondition for high amplification is therefore a *populated cache under attacker control*.

Across the window we observed only a handful of retrieval probes: five well-formed `gets` requests from a single source, plus a malformed multi-key `gets` variant, and **not a single `set`, `get` (different from `gets`), `add`, `delete` or `version` command**. Because no source ever *stored* a value, every retrieval necessarily missed and returned only `END` (the `gets` replies amplified at 0.7×). The precondition for the high amplification regime was therefore never even attempted against the honeypot; observed interaction stopped at reconnaissance (`stats`) and, in a few cases, empty retrieval. Two caveats bound this result: our 1 Mbit/s cap and small synthesised `stats` payload deliberately suppress the achievable factor (we measure attacker behaviour, not the protocol's true ceiling), and a scanner that probes only with `stats` would observe roughly the same modest factor we report and could in principle deprioritise us.

5.3 Q3. Separating attack traffic from benign scanning

Looking at a few simple per-source signals (protocol variant, UDP-header presence, command, same-port fraction, malformed fraction) the population separates cleanly into three behavioural classes without any need for clustering subtlety:

1. **Memcached reconnaissance scanners.** Properly formed text protocol, UDP frame header present, request id `0` or `1`, `stats` requests, sweeping many or all honeypot addresses on a regular interval. These are the 23 full-block sweepers and the bulk of the 267 `stats` sources. They are clearly aware of Memcached, but the evidence points to measurement and enumeration rather than active weaponisation.

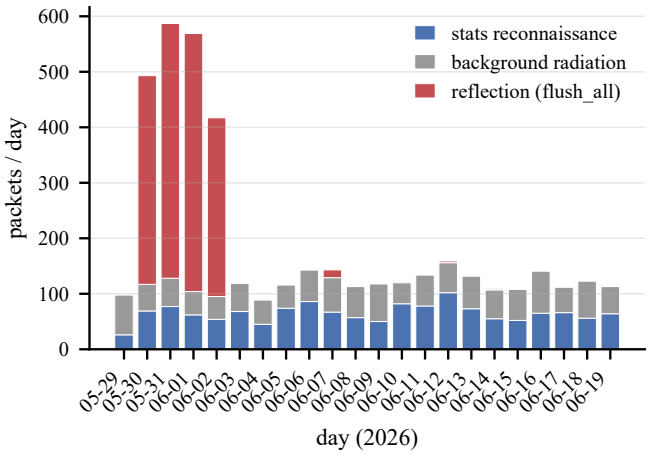

Figure 2: Daily packet counts by class: continuous `stats` reconnaissance and malformed background radiation, with a discrete four-day `flush_all` same-port burst (2026-05-30 to 06-02).

2. **Non-protocol background radiation.** The 1,037 unrecognised packets (24% of traffic, 183 sources) carried no UDP frame header (0% header presence vs. 95% for the text class), were flagged `malformed`, and consisted of single random looking payloads. This is generic internet noise and non-Memcached fuzzers that happen to reach UDP/11211. They never produced a meaningful reply.
3. **Same-port anomaly.** A single source sent `flush_all` from source port `11211` to the host's management IP, not an advertised honeypot address. Source port 11211 is never used by a legitimate client, so the traffic is anomalous, but its intent is ambiguous: a scanner pinning its source port to 11211, backscatter from an attack spoofing our address, or a crude reflection probe would all look the same. **It is the only class that is abuse-related rather than benign measurement or noise**. The same-port rule dropped all 1,646 such packets.

The two discriminators that carry essentially all of the separating power are (i) presence of the 8-byte UDP frame header, which splits genuine Memcached tooling from background noise, and (ii) the same-port fraction, which isolates the same-port anomaly. Command sequence and inter-packet timing refine the scanner class but are not needed to draw the primary boundaries.

5.4 Q4. Attacker workflow

Mapping each source's interactions onto the scan/test/execute model of Griffioen et al. [5], we observe the first two phases clearly and the third only in a degenerate form.

Scan and *test* are well attested: sources sweep the full address block (scan) and return to issue repeated `stats` requests after a regular interval (test). This is precisely the amplification-factor measurement behaviour the model predicts. *Execute*, repeated large reply requests bearing spoofed victim source addresses, was **not** observed. There were no store commands at all and only a handful of empty retrieval probes (subsection 5.2); no source progressed from testing to weaponisation against the honeypot.

The one event that resembles an execute phase artefact is the `flush_all` stream from source port `11211`. A single source produced 1,621 such packets, directed at the host's management IP rather than an advertised honeypot address, and only during a discrete four day burst (2026-05-30 to 2026-06-02: 376, 459, 464 and 322 packets per day, respectively, then nothing for the remaining seventeen days). Its same-port encoding and management IP destination mark it as the same-port anomaly of subsection 5.3 rather than a genuine probe of our amplification capacity. The safety gate dropped all of it, so it contributed zero amplification. By contrast, `stats` reconnaissance and background radiation fuzzing continued at a roughly constant daily rate across the whole window (Figure 2), with only modest day-to-day variation over the remainder of the period.

Collectively, the observations suggest the presence of a residual ecosystem focused on the continuous discovery and remeasurement of exposed Memcached hosts. However, from our vantage point and measurement window, activity almost never escalates to the weaponisation phase associated with Memcached based reflection attacks.

6 Discussion

6.1 Interpretation

The primary finding is a divergence between interest and action. Eight years after the patch, Memcached on UDP/11211 is still actively and systematically scanned: 465 sources in roughly three weeks, 23 of them sweeping the entire advertised block at regular intervals with recognisable tooling fingerprints. This suggests the protocol has not fallen off attackers' maps. Yet across the same window we saw no store then retrieve weaponisation, only a handful of empty `gets` probes that found nothing to retrieve. There was no spoofed victim large reply traffic, and an empirical amplification ceiling ($\approx$13$\times$ from `stats`) that no observed source attempted to push past with a populated cache. The reflector population behaves as if Memcached is a **catalogued, but dormant amplifier**. Worth enumerating and re-measuring, but rarely worth weaponising, at least not from the same hosts that perform the discovery, and not against a fresh, unpopulated amplifier like ours.

This is consistent with, and refines, the macroscopic literature. Hiesgen et al. [6] report that legacy amplification protocols keep appearing in attack traffic long after they are publicly "fixed". Our data shows that for Memcached this persistence is, at the level of an individual reflector, dominated by the *reconnaissance* stages of the Griffioen et al. [5] workflow rather than the execute stage. The scan/test pattern was observed fully. The execute pattern surfaced only in a degenerate form (isolated retrieval probes against an empty cache) and never materialised as weaponisation against an amplifier offering only a modest, capped factor.

6.2 Effectiveness of the safety architecture

The layered design behaved exactly as intended. The primary invariant (the 1 Mbit/s global cap) never engaged. The to-

tal egress was 277 KiB over the window and the peak per-source rate we ever observed was 26 packets per second, below (though no longer comfortably below) the 30 pps attack-prep threshold. No packet was ever dropped for a rate-limit, global-cap or PPS reason. The only active drops were the same port reflection rule (1,646 packets, which prevented us from relaying a four-day same-port stream aimed at the host's management IP) and the "no reply defined" default for malformed traffic. The 18 silent-mode drops correspond to the mandated first boot validation period. In short, the honeypot's own contribution to the ecosystem it measured was negligible, satisfying the accounting that Hiesgen et al. [6] argue responsible deployments owe. In particular, the same port reflection rule proved to be not a theoretical safeguard but the mechanism that actually neutralised the only abuse traffic we received.

6.3 Limitations

Beyond the three design limitations already noted in subsection 3.6 (approximate emulation, the behaviour-distorting 1 Mbit/s cap, and a small sample of the reflector population), the results carry four further caveats.

First, **the window is short and the vantage point single**. Twenty-two days from one VM and a single contiguous /28-scale block cannot distinguish "execute phase is rare" from "execute phase is bursty and we missed it". The four-day `flush_all` burst is a direct reminder that abuse traffic arrives in discrete campaigns. A longer or distributed deployment could catch a weaponisation event our window did not.

Second, **we never offered a high-value bait**. Because our `stats` response is small and we expose no retrievable large value, an attacker measuring our amplification factor sees only $\approx 13\times$ and may rationally move on. The handful of empty `gets` probes we did observe are consistent with this reading: a source checked for a retrievable value, found none, and did not return. We therefore cannot tell whether the absence of an execute phase reflects a genuine lull or our own unattractiveness as an amplifier. The honeypot's safety and its baiting power are in direct contention here.

Third, **the complete absence of binary-protocol traffic** (0 of 4,261 packets) is itself a finding that could be read two ways: either the binary Memcached protocol is simply not used by current UDP scanners, or our binary parser mishandles some probe shape and those sources filtered us out before we logged a recognisable command. We lean towards the former, since malformed text traffic *was* logged in volume, but we cannot fully exclude the latter.

Fourth, **we deliberately performed no external enrichment**. Per the responsible research constraints of Section 4, source IPs were not joined against geolocation, ASN, or threat-intelligence datasets. We characterise behaviour rather than location of origin. The tool-fingerprints (request-id 0, header presence) are the strongest attribution signal we permit for this work.

6.4 Future work

Several directions follow directly from the limitations above.

Longer and distributed deployment. Replicating the honeypot across multiple networks and address blocks, over months rather than weeks, would turn "we did not observe weaponisation" into a quantified rate and would reveal whether the same scanner sources reappear across vantage points, as Thomas et al. [12] found for other protocols.

Controlled execution phase baiting. The cleanest way to test whether the execute phase still exists for Memcached is to let the honeypot answer `set`/`get` so an attacker can stage and retrieve a value, while keeping the global cap so the retrieved value is never actually amplified onto a victim. This converts the honeypot from a passive recon target into an active execute-phase trap, at the cost of careful additional safety review.

Cross-correlation with darknet and backscatter. Fusing the honeypot's spoofed source observations with darknet backscatter, in the manner of Jonker et al. [4], would let a future study link the scan/test activity we see to the victim side evidence of any campaigns that do reach execute.

Personality and fingerprint experiments. Systematically varying the `fake_stats` personality and the advertised version across honeypot IPs would measure how scanner engagement depends on apparent amplification factor and patch level, directly probing the attractiveness/safety trade-off this study could only observe at a single operating point.

7 Conclusion

We built a custom, safety-bounded Memcached honeypot and exposed it on a block of 16 public IPv4 addresses for roughly 22 days to ask to what extent, and how, the Memcached amplification vector is still abused in the post-patch era. Our observations show Memcached is clearly not ignored: 465 distinct sources probed us, twenty-three of them systematically swept the entire advertised block at regular intervals. Additionally, the dominant activity of `stats` reconnaissance with a recognisable UDP header and a request id 0 fingerprint maps cleanly onto the *scan* and *test* phases of the established adversarial workflow. At the same time, the feared *execute* phase never arrived against us. No source issued a single store command, only a handful of empty `gets` retrieval probes (which found nothing to retrieve) appeared, and no spoofed victim weaponisation was observed. The only abuse-related traffic, a four-day burst of same-port `flush_all` from source port 11211, was directed at the host's management IP rather than an advertised honeypot address, and was neutralised by the same-port drop without contributing a byte of amplification. The empirical amplification we could be made to deliver topped out at $\approx 13\times$, far below the protocol's notorious ceiling, because its precondition of an attacker-populated cache was never established in our stateless honeypot.

Our five-layer safety architecture held under real traffic. A total of 277 KiB egressed over the window, the peak per-source rate was 26 packets per second (below the 30 pps attack-prep threshold), and the global rate cap never once engaged, demonstrating that a faithful amplification honeypot can be operated on the public internet with a negligible and measurable contribution to the very ecosystem it studies. Taken together, the picture for Memcached today is one of an attack surface that is continuously catalogued but, from this vantage point and in this window, only rarely weaponised.

References

[1] GitHub, "DDoS incident report," https://github.blog/news-insights/company-news/ddos-incident-report/, 2018, accessed: 2026-05-01.

[2] M. Kondo, R. Tanabe, N. Shintani, D. Makita, K. Yoshioka, and T. Matsumoto, "Amplification chamber: Dissecting the attack infrastructure of memcached DRDoS attacks," in *Detection of Intrusions and Malware, and Vulnerability Assessment (DIMVA)*, ser. Lecture Notes in Computer Science, vol. 13358. Springer, 2022, pp. 178–196.

[3] L. Krämer, J. Krupp, D. Makita, T. Nishizoe, T. Koide, K. Yoshioka, and C. Rossow, "AmpPot: Monitoring and defending against amplification DDoS attacks," in *Research in Attacks, Intrusions, and Defenses (RAID)*, ser. Lecture Notes in Computer Science, vol. 9404. Springer, 2015, pp. 615–636.

[4] M. Jonker, A. King, J. Krupp, C. Rossow, A. Sperotto, and A. Dainotti, "Millions of targets under attack: A macroscopic characterization of the DoS ecosystem," in *Proc. Internet Measurement Conf. (IMC)*, 2017, pp. 100–113.

[5] H. Griffioen, K. Oosthoek, P. van der Knaap, and C. Doerr, "Scan, test, execute: Adversarial tactics in amplification DDoS attacks," in *Proc. ACM SIGSAC Conf. Computer and Communications Security (CCS)*, 2021, pp. 940–954.

[6] R. Hiesgen, M. Nawrocki, M. Barcellos, D. Kopp, O. Hohlfeld, E. Chan, R. Dobbins, C. Doerr, C. Rossow, D. R. Thomas, M. Jonker, R. Mok, X. Luo, J. Kristoff, T. C. Schmidt, M. Wählisch, and K. C. Claffy, "The age of DDoScovery: An empirical comparison of industry and academic DDoS assessments," in *Proc. ACM Internet Measurement Conf. (IMC)*, 2024, pp. 259–276.

[7] C. Rossow, "Amplification hell: Revisiting network protocols for DDoS abuse," in *Proc. Network and Distributed System Security Symp. (NDSS)*, 2014.

[8] D. Kopp, C. Dietzel, and O. Hohlfeld, "DDoS never dies? an IXP perspective on DDoS amplification attacks," *arXiv preprint arXiv:2103.04443*, 2021.

[9] Memcached, "Memcached overview," https://docs.memcached.org/, 2024, accessed: 2026-06-02.

[10] ——, "Memcached 1.5.6 release notes," https://github.com/memcached/memcached/wiki/ReleaseNotes156, 2018, released 2018-02-27. Accessed: 2026-06-02.

[11] T. Foltýn, "New DDoS attack method breaks record again, adds extortion," WeLiveSecurity (ESET blog), https://www.welivesecurity.com/2018/03/08/new-ddos-attack-method-breaks-record/, 2018, accessed: 2026-05-01.

[12] D. R. Thomas, R. Clayton, and A. R. Beresford, "1000 days of UDP amplification DDoS attacks," in *Proc. APWG Symp. Electronic Crime Research (eCrime)*. IEEE, 2017, pp. 79–84.

[13] M. Hájek, "Memcached DDoS research honeypot," https://github.com/Maxim-Hajek/memcached-honeypot, 2026, accessed: 2026-06-21.