Winnie-the-PWNED: Measuring Adversarial Behavior Against MCP Servers

Ashley H. Dai
Stanford University

Devin T. Fung
Stanford University

Aolin E. Zhang
Stanford University

Abstract

The Model Context Protocol (MCP) enables large language models to interact with external tools through a standardized client–server interface, expanding the capabilities of agentic systems while introducing a new attack surface. Although recent work has proposed numerous MCP-specific threat models and attack vectors, there is little empirical evidence on how adversaries engage with MCP servers in the wild. To address this gap, we investigate how adversaries behave when attempting to attack MCP servers. To do so, we establish MCP honeypot servers to measure and characterize real-world techniques. Specifically, we (1) observe end-to-end attack behavior from initial enumeration to potential exploitation and data access, (2) quantify attacker prevalence and indicators of exploitation or exfiltration intent, and (3) analyze how different security controls deter or prevent adversarial behavior. Over a two-week measurement period, we observed no evidence of MCP-specific enumeration or exploitation attempts, and only standard internet scanning and exploitation activity targeting conventional web vulnerabilities. These results suggest a disconnect between modeled MCP threats and current attacker incentives, indicating that MCP has not yet been incorporated into large-scale automated attack ecosystems.

1 Introduction

Large language models (LLMs) are increasingly deployed as agentic systems that interact directly with external tools and services, allowing models not only to generate text but also to execute actions, retrieve data, and modify system state. The Model Context Protocol (MCP), introduced by Anthropic in late 2024, standardizes these interactions through a uniform client–server interface, lowering the barrier to integrating agents with complex software ecosystems. As a result, MCP has rapidly become a foundational layer for agent–tool integration across personal assistants, developer tooling, and enterprise automation.

At the same time, MCP introduces a new and distinct attack surface. By design, MCP exposes structured interfaces through which models access tools, resources, and potentially sensitive operational contexts. Recent research has highlighted a variety of MCP-specific risks (such as tool poisoning, namespace squatting, indirect prompt injection, and sandbox escape), suggesting that MCP servers may become attractive targets for adversaries seeking to compromise agentic workflows or downstream systems. These concerns have motivated a growing body of theoretical threat models and vulnerability analyses.

Despite this attention, it remains unclear how these risks translate into real-world adversarial behavior. Much of the existing MCP security literature focuses on what could go wrong under adversarial conditions, but offers limited evidence on whether and how attackers currently discover, probe, or attempt to exploit MCP servers in practice. This gap mirrors a broader challenge in security research: distinguishing between plausible attack surfaces and those that attackers are actually incentivized to target at scale.

Measurement-driven studies, particularly those using honeypots, have historically played a critical role in addressing this challenge. Instrumented decoy systems have been used to empirically characterize attacker behavior, reveal dominant attack vectors, and identify mismatches between assumed and observed threats. Applying similar measurement methodologies to MCP offers an opportunity to ground emerging concerns about agent–tool security in empirical observation, especially at an early stage in MCP's adoption.

In this work, we present an empirical study of adversarial interaction with MCP servers using purpose-built MCP honeypots. At a high level, our measurements show that while MCP servers are exposed to substantial volumes of background Internet scanning and exploitation activity, this activity overwhelmingly targets conventional web vulnerabilities rather than MCP-specific interfaces. We do not observe evidence that MCP has yet been incorporated into large-scale automated attack workflows. Thus, the main contributions of this paper are:

(1) Proposing a novel MCP honeypot implementation.

(2) Establishing an initial baseline for MCP attacking against which future data can be measured.

(3) Finding that MCP attacks are not large scale, nor are they automated.

2 Related Work

Recent research has begun mapping out the security landscape around MCP. Zhang et al. provide the first systematic analysis of the MCP ecosystem, examining its architecture, workflow, and security implications to standardize tool definition, discovery, and invocation for AI applications [10]. They construct a threat taxonomy covering four major attacker archetypes: malicious developers, external attackers, malicious users, and general security flaws, encompassing 16 specific threat scenarios, such as Namespace Typosquatting, Tool Poisoning, Rug Pulls, Indirect Prompt Injection, and Sandbox Escape. Our work builds directly on this taxonomy by shifting from theoretical modeling to empirical observation, using an instrumented honeypot to capture how these threat classes manifest in the wild.

Guo et al. introduce the MCP Attack Library (MCPLIB), the first unified, plugin-based framework designed to simulate reproducible MCP vulnerabilities [2]. Their taxonomy expands prior work by enumerating thirty-one distinct attack techniques across Direct Tool Injection, Indirect Tool Injection, Malicious User, and LLM-Inherent categories, each evaluated along four quantitative dimensions: risk level, success rate, impact scope, and implementation difficulty. This work informs our methodology and analysis by establishing a structured baseline of known attack primitives, which we use to guide the design of our honeypot.

Building on this groundwork, Rahman et al. conducted a large-scale static analysis of 1,899 open-source MCP repositories, revealing that 7.2% contained general vulnerabilities and 5.5% exhibited domain-specific tool-poisoning patterns undetectable by conventional static analysis [6]. Their findings underscore the prevalence of insecure MCP implementations but are limited by the ability of static approaches to capture dynamic exploitation behaviors, motivating our decision to employ runtime observation that static scanners miss.

Meanwhile, honeypot systems represent a long-standing methodology for empirically observing attacker behavior. The foundational concepts were established and popularized by Spitzner, who defined low-interaction honeypots (simulated services for scalable data collection) and high-interaction honeypots (fully functional systems that capture deep behavioral traces) [8]. His work laid the conceptual groundwork for using controlled deception to both observe and contain adversaries.

Singh et al. advance this line of work with deception strategies (camouflage, bluffing, and playing dead), demonstrating that subtle environmental cues in SSH and Telnet honeypots–two popular protocols that attackers scan–can increase and prolong attacker engagement [7]. We use this to refine our methodology by anticipating and attempting to take anticipate and take advantage of adversary behavior, to maximize engagement time and behavioral yield.

The recent popularity of LLMs has also given way to new honeypotting techniques, such as using LLMs to augment honeypot design. The Beelzebub framework, for instance, uses LLMs to dynamically change how a honeypot interacts, allowing a low-interaction honeypot to mimic a high-interaction honeypot [1]. This new framework allows honeypot operators to extract deep attacker behavior without increasing security risk by actually giving attackers more system access.

Despite the clear need for securing MCP interactions, and the demonstrated ability to incorporate LLMs into honeypots, there has not been any research done to produce an MCP honeypot, where the honeypot itself is disguised as a vulnerable MCP server.

Together, these works inform our motivation and design of an actual MCP honeypot that applies classical deception principles to a new protocol layer. By combining insights from MCP security research with engagement strategies from literature on honeypots, our approach enables empirical measurement of attacker behavior against live MCP endpoints, bridging the gap between theoretical attack frameworks and real-world adversarial behavior.

3 Methodology

In this section, we describe the implementation of our MCP honeypot application, including its user-facing functionality, creative design, logging capabilities, and security.

3.1 MCP Profiles

We have created two MCP server profiles: a personal AI assistant profile, and a corporate tools profile. We aim to make the servers look as attractive of a target as possible to attackers, without raising suspicion. To make the target attractive, we advertise functionality that provides access to PII or similar. We argue that our servers are not suspicious, because of the novelty of MCP servers–there is also no precedent for what an MCP honeypot would look like to an attacker.

Personal Assistant Profile: Simulates a personal AI assistant with access to a fictitious owner's emails, finances (credit cards, transactions), password manager, and SSH private keys. **Corporate Tools Profile:** Simulates enterprise infrastructure access: payroll and HR systems, cloud credentials (AWS, GitHub), database connection strings, development tool access.

This approach allows for comparison of attacker behavior between targeting personal vs. corporate infrastructure. Addi-

tionally, we achieve some baseline redundancy by deploying more than one honeypot.

3.2 Server Implementation

We use the FastMCP library to construct our honeypot application. [3] This library is a popular choice for MCP servers in Python, with over 20 thousand stars on GitHub. [4] Additionally, it contains middleware functionality that we use for logging. [3]

3.3 Hosting

Our honeypot is hosted with Google Cloud Platform[1] in a Cloud Run service, where, within a Docker container, we run our Python script. We record logs on multiple network layers to observe attacker behavior (see Section 3.5).

3.4 Security

Here, we outline the security of our honeypot, showing that with proper system administration, there is no increased system risk with our low-interaction honeypot.

Authentication: We allow for unauthenticated access to our honeypot since doing so is both realistic and allows for easier exploitation. As mentioned, the novelty of MCP servers means that a lack of security configurations is reasonable. In any case, this makes little security difference, since authenticated honeypots are usually intended to be accessed anyways.

Interaction level and data simulation: We create a low-interaction honeypot, since we are able to achieve the full functionality of the MCP server through simulated API responses. For example, an attacker who attempts to fetch the email inbox of the server's fictitious owner will simply receive a hardcoded string full of fake email addresses and meaningless email contents. With this isolation, any attacks up to and including full machine takover will provide the attacker with no access beyond the machine. A security limitation is that any vulnerabilities in the fastMCP library may lead to such a container compromise.

Containerization: Here, we rely on the security of Docker. In the case of denial-of-service or container compromise, we can simply migrate to a new server or restart the container.

Hosting: By hosting on GCP Cloud Run (or any similarly robust platform), the worst case scenario is machine compromise following Docker sandbox escape. If this is possible, there are much more concerning consequences for the world than someone misusing our MCP honeypot, and it is unlikely someone would conduct that attack on our server. By using a load balancer, we can assign a static IP address to increase discoverability, while allowing us to rate limit to prevent DoS.

[1] GCP was selected because one author had extra credits.

3.5 Data Collection

We log data across four layers to understand attacker behavior.

Layer 1: HTTP Connection Logging. All HTTP requests are intercepted via Starlette middleware, [9] [3] which allows us to log request contents, request timing, volume, and client metadata. This allows for analysis of any scanning or attacking behavior.

Layer 2: MCP Session Establishment. Upon successful MCP protocol initialization, [5] we log session establishment events, including client data. This enables us to distinguish MCP-aware clients from background Internet noise.

Layer 3: Tool Enumeration. This layer maps to the reconnaissance phase in cyber kill chain models. We log tool discovery requests (e.g., `list_tools()`), frequency and patterns of enumeration, and timing from session start.

Layer 4: Tool Execution (Potential Exploitation). Actual tool invocations represent exploitation attempts. For each call, we log the tool and parameters called, what response was given, and the timing and volume of such requests.

3.6 Data Analysis

We implement several pattern detection mechanisms.

Timing Analysis. We calculate the intervals between consecutive tool calls for a particular client–who is identified by IP and user agent–tracking average, minimum, and maximum intervals. If five consecutive requests appear within 2 seconds of each other, the requests are flagged as `likely_automated: true`.

Path Traversal Attempts. We scan all parameter values for directory traversal indicators. The pattern `path_traversal_attempt` is triggered if any parameter contains `".."` `"/"`, excluding legitimate protocol patterns such as `http://` in URLs).

SQL Injection Attempts. We analyze parameter content for common SQL injection keywords: `UNION`, `SELECT`, `DROP`, `INSERT`, `DELETE`, `1=1`, `OR 1=1`, `';`, - Detection of any keyword triggers a log entry with `sql_injection_attempt`.

Command Injection Attempts. Similarly, we log `command_injection_attempt` for these keywords: `;`, `|`, `&`, `` ` ``, `$(`, `<`, `>`

Tool Enumeration Pattern. We identify systematic tool discovery behavior. For clients with at least eight requests, we examine the last (up to) 20 requests and count unique tool names. If at least six distinct tools are called, the `tool_enumeration` pattern is triggered. This indicates reconnaissance behavior where attackers systematically explore available capabilities.

4 Results

Across our two-week deployment, our MCP honeypots received a total of 18,594 requests. The personal-profile server accounted for 10,000 requests, while the corporate-profile server received 8,594 requests. Of these, 9,622 requests were filtered out, consisting of Google Cloud Run internal health checks or otherwise malformed data (e.g. empty source IP fields). We analyzed the remaining requests, originating from 1,193 unique IP addresses.

4.1 MCP Traffic

We recorded zero requests that attempted to engage with the MCP protocol. Specifically, no requests were made to the /mcp endpoint of our servers, which is the entry point for any mcp communication, and thus we also did not see any traffic relating to MCP tool or endpoint usage (such as the list_tools() call). Additionally, we did not find any strings or other payload data that suggested any attempt to discover or communicate with an MCP server.

This result suggests that MCP-aware adversaries are limited in number and activity. It is possible that scanning for MCP does not yet exist (at least, meaningfully so) in the wild, and the only scanning that occurs are isolated and manually conducted scans for research or individual attack attempts. This selective scanning theory is supported by the idea that MCP is a much more variable protocol than traditionally honeypotted protocols, such as SSH. In MCP, all configurations (tool names, outputs from LLMs, etc) are arbitrary text values, and the only aspect that is easily scripted is the underlying HTTP protocol. Thus, designing a scanner for vulnerable MCP services may be a more complex undertaking than using an existing web-scanner that easily discovers and attempts to attack SSH services, or perhaps requires a significant degree of human or LLM-assisted direction.

4.2 Non-MCP Traffic

Though our traffic records did not demonstrate any MCP-aware adversarial behavior, we describe the traffic we did receive, both to demonstrate what an MCP server administrator may encounter, and simply for the sake of completeness and acknowledging our collected data. Since the MCP protocol runs on top of HTTP, on port 80, our honeypot was largely scanned and attacked as if it were a website.

Traffic Composition. We recorded 2,156 root-directory requests (`/`) and 1,680 unique path requests. Despite this wide variety, traffic was highly concentrated around a small number of paths. The 25 most common paths accounted for more than 23% of all path requests (Table 1). These included well-known CMS paths (e.g., `/wp-admin/`), vulnerable IoT device endpoints (`/GponForm/diag_Form`), and common misconfiguration probes (e.g., `/.env`).

Path	Count
/	2156
/.env	503
/cgi-bin/luci/;stok=/locale	259
/index.php	215
/favicon.ico	153
/.git/config	126
/public/index.php	75
/api/.env	75
/hello.world	68
/cgi-bin/%252e%252e[...]/%252e%252e/bin/sh	68
/containers/json	62
/robots.txt	47
/backend/.env	47
/admin/.env	45
/admin/config.php	43
/json/	40
/geoserver/web/	37
/.env.example	33
/sitemap.xml	32
/actuator/gateway/routes	31
/.aws/credentials	30
/app_dev.php/_profiler/phpinfo	27
/1.php	26
/app/.env	24
/developmentserver/metadatauploader	21
/.well-known/security.txt	21
/vendor/phpunit/phpunit/src/Util/PHP/eval-stdin.php	21

Table 1: Paths with more than 20 requests.

Attack Pattern Classification. To analyze attacker intent at scale, we grouped all observed paths into different behavioral categories using a semi-automated labeling workflow. This workflow combined rule-based filters with an LLM-assisted classification step (using GPT-4o). The resulting categories were: (1) reconnaissance and fingerprinting, (2) secret and configuration probing, (3) known exploit attempts, (4) botnet and IoT malware scanning, and (5) post-exploitation and backdoor probing. Table 2 reports the final distribution of requests across categories. As shown, reconnaissance and secret probing together accounted for the majority of requests, while known exploit attempts and IoT malware traffic made up smaller but still meaningful portions of observed behavior.

In the next paragraphs, we provide detailed breakdowns of each behavioral class.

Category 1: Reconnaissance and Fingerprinting Traffic The largest volume of traffic consisted of reconnaissance, where scanners or attackers probe servers to determine the

Category	Requests	(% of Total)
Recon & Fingerprinting	3,808	42.4%
Secrets / Config Probing	2,472	27.6%
Post-Exploitation Indicators	1,220	13.6%
Other	847	9.4%
Botnet / IoT Scanning	434	4.8%
Known Exploit Attempts	121	1.3%
Total	8,972	100%

Table 2: Distribution of total requests across attack categories, sorted by request volume.

Category	Paths	(% of Total)
Secrets / Config Probing	573	34.1%
Post-Exploitation Indicators	559	33.3%
Recon & Fingerprinting	343	20.4%
Other	148	8.8%
Botnet / IoT Scanning	43	2.6%
Known Exploit Attempts	14	0.8%
Total	1,680	100%

Table 3: Distribution of unique paths across attack categories, sorted by unique-path count.

types of services that are exposed, any potential weaknesses or vulnerabilities, or any identifying features that provide more context to the server's existence.

In our deployment, reconnaissance requests consisted of probes for: (1) Health or version endpoints (e.g., `/metrics`, `/api/version`, `/openapi.json`) that may reveal framework information or build metadata, (2) Platform-specific directories such as `/wp-admin/`, `/magento_version`, and `/symfony/_profiler/`, which identify well-known content management systems, (3) Administrative dashboards including `/solr/admin/`, `/nifi/`, and `/hudson/`, which may indicate potential for administrator-level access. (4) Identifiers, such as file names and service names, or known patterns used to fingerprint server behavior.

These reconnaissance requests were nearly identical across both honeypots, suggesting that most traffic originated from large-scale, indiscriminate scanning.

Category 2: Secrets, Source Code, & Misconfiguration Probing A substantial portion of traffic attempted to identify exposed secrets and misconfigurations that would allow for trivial exploitation.

Specifically, we observed: (1) Attempts to fetch environment variable files (e.g., `.env`), (2) Attempts to access version control directories, such as `/.git/config`, (3) Attempts to retrieve backup or configuration archives such as `backup.tar.gz` and `database.sql`, (4) Path traversal attacks attempting to access sensitive files such as `/etc/passwd`

Category 3: Known Exploit Attempts We also recorded requests attempting to exploit documented vulnerabilities (CVEs). Examples included: (1) Command execution attempts against legacy CGI interfaces using URL-encoded payloads designed to invoke system shells, (2) Telerik UI exploitation patterns (CVE-2017-11317), which target arbitrary file uploads and code execution, (3) Trend Micro OfficeScan bypass probes, targeting authentication weaknesses in older enterprise security products.

Category 4: Botnet and IoT Malware Activity A notable fraction of traffic are suspected to have originated from IoT malware (Mirai, Mozi, and Gafgyt) which continuously scan the public IPv4 space for vulnerable embedded devices. These malware variants rely on predictable URL paths and web interfaces that are common across consumer devices.

We observed: (1) GPON router exploit attempts (CVE-2018-10561 / 10562), enabling unauthenticated command execution. (2) Probes targeting outdated embedded Boa web servers still present in millions of devices, (3) `/goform/*` command injection patterns, a recurring weakness across many consumer networking devices, (4) Camera firmware probes associated with botnet propagation activity.

These patterns further emphasize that our honeypots were treated like ordinary Internet hosts by automated malware infrastructure.

Category 5: Post-Exploitation / Backdoor Scanning We also observed traffic consistent with attackers attempting to determine whether a host had already been compromised by another adversary. This behavior is common: rather than exploiting systems themselves, some attackers scan for existing webshells or backdoors that grant ready made access.

These requests targeted: (1) Common webshell filenames such as `wso.php`, `alfa.php`, and `shell.php`, (2) Fake WordPress administrative paths used by certain malware droppers to maintain persistence, (3) Obfuscated command injection payloads using techniques such as `$IFS` to bypass simple input validation filters.

5 Discussion

This study evaluates whether recently proposed security concerns surrounding MCP servers are reflected in real-world adversarial behavior. We observe no evidence of MCP-aware scanning, enumeration, or exploitation, suggesting that MCP currently occupies an early stage in the attacker adoption lifecycle: while potential attack surfaces exist, they have not yet been operationalized at scale.

Consistent with this interpretation, the observed traffic does not differentiate itself from what is to be expected from adversarial scanning and background internet noise. Adversaries overwhelmingly target misconfigurations, exposed configuration files, outdated dependencies, and long-known vulnerabilities, rather than engaging with protocol-specific interfaces. From an economic perspective, this behavior is unsurprising. MCP servers currently lack the deployment density and standardized ecosystem support needed to justify the development of bespoke attack tooling, especially when compared to ubiquitous targets such as content management systems or vulnerable IoT devices that can be exploited at scale with minimal effort. Any MCP attacks might instead be manually driven and targeted at specific companies or individuals that present themselves as attractive MCP targets (perhaps knowing that they handle sensitive information).

However, the absence of MCP-specific activity should not be interpreted as a contradiction existing MCP threat models. Instead, it reflects differences in both timing and attack channel. Our findings suggest that, at present, MCP risks are more plausibly exercised through targeted ecosystem interactions than through opportunistic internet-wide scanning, and are therefore unlikely to surface in passive honeypot measurements.

These results indicate that near-term internet-facing risk is dominated by conventional web vulnerabilities rather than MCP-specific mechanisms. For researchers, this establishes an empirical baseline against which future shifts in attacker behavior can be evaluated. For practitioners deploying MCP today, it reinforces that foundational controls—authentication, rate limiting, dependency management, and secrets hygiene—remain the most effective mitigations, while MCP-specific telemetry should be instrumented in anticipation of future attacker adaptation.

Independent of the empirical findings, this work makes a methodological contribution by introducing the first MCP-specific honeypot framework for measuring real-world adversarial behavior. Prior MCP security analyses have focused on threat modeling or case studies, but lacked infrastructure for internet-scale measurement. Our honeypot operationalizes MCP servers in a realistic, instrumented setting, providing a reusable baseline that future work can extend to track attacker adaptation as MCP adoption grows.

Temporal scope and falsifiable predictions. These conclusions are inherently time-dependent and yield concrete, testable predictions. As MCP adoption increases, a transition toward commoditized attacks would be expected to manifest through systematic MCP discovery behavior, repeated invocation of protocol-specific operations (e.g., tool enumeration), and the emergence of recognizable MCP fingerprints in automated scanning frameworks. Sustained observation of such signals would indicate a shift in attacker incentives and warrant a reassessment of the threat landscape described here.

5.1 Limitations

While our honeypot deployment provides valuable initial insight into how adversaries behave toward MCP servers, there are limitations that constrain the scope of our findings.

Passive measurement bias. Our honeypots capture opportunistic, internet-scale adversarial behavior and are not designed to observe targeted or ecosystem-facing attacks. Because the deployment was fully passive, the dataset primarily reflects background scanning and automated exploitation rather than adversaries who locate MCP infrastructure through developer ecosystems, supply-chain compromise, or leaked assets. Accordingly, our results bound the prevalence of MCP-specific activity in large-scale scanning, but do not rule out targeted MCP exploitation through other channels.

Deployment locality. All honeypots were deployed within a single cloud hosting environment and IP region. Attacker behavior varies across network types and address space, and deployments on residential, enterprise, or academic networks may attract different attacker populations. In our case, the IP addresses of both honeypot profiles belonged to the same "/16" CIDR range owned by Google Cloud Platform.

Limited implementation diversity. We relied on a single MCP server implementation with two tool profiles. Real-world MCP deployments vary in authentication mechanisms, tool semantics, and protocol handling, and attackers may target implementation-specific behaviors that our honeypots did not expose. As a result, our findings should not be interpreted as representative of all MCP implementations. In particular, MCP may instead use standard input/output (stdio) instead of HTTP. [5]

No active publicity. We did not actively seed honeypot endpoints into developer-facing or adversary-facing channels such as public repositories, documentation, forums, or social platforms. This limits visibility into adversaries who discover MCP servers through ecosystem or social-engineering vectors rather than network probing, and likely underrepresents such attack pathways. For example, an opportunistic adversary may be tempted to attack an MCP server belonging to a company if an oversharing employee boasts online about using MCP for development around sensitive data.

5.2 Future Work

This study represents an early step toward understanding real-world attacker behavior toward MCP servers. Based on our observations and limitations, we identify several promising directions for future research.

Deployment and implementation diversity. Future work should expand MCP honeypot deployments across heterogeneous implementations, configurations, and network environments. Varying MCP servers by implementation frameworks, tool semantics, authentication models, hosting provider, geography, and consumer vs. enterprise IP addresses would enable measurement of how attacker behavior varies with deployment context and improve coverage beyond a single operational slice of the Internet.

Increased interaction fidelity. Our honeypots exposed static, predefined behavior. Introducing stateful responses, simulated tool execution, and multi-step task flows would allow measurement of whether deeper interaction elicits MCP-aware probing or multi-stage exploitation attempts. Higher-fidelity interaction is necessary to distinguish superficial scanning from adversaries testing protocol semantics.

MCP discoverability and fingerprinting. An open question is how MCP servers become detectable to automated scanners. Future studies should characterize protocol-level signatures, including WebSocket upgrade patterns and JSON-RPC behavior, and evaluate whether such features are sufficient for reliable MCP identification. Understanding discoverability is critical for predicting when MCP-specific probes may be incorporated into attacker toolchains. In the most minimal case, a honeypot that is able to respond to requests from the wrong protocol with an error message referencing MCP may lead to more engagement.

Ecosystem-based discovery channels. With appropriate safeguards, limited active seeding experiments—such as embedding MCP endpoints in controlled repositories or sample projects—could measure how attackers discover MCP infrastructure through public-facing channels.

Detection and longitudinal measurement. The traffic patterns observed in this study provide a basis for developing MCP-specific detection heuristics, including anomalous tool enumeration, timing irregularities, and malformed protocol interactions. Longitudinal deployment of such instrumentation would allow researchers to track when MCP transitions from a niche protocol to a commoditized attack surface and to correlate attacker adaptation with MCP adoption over time.

Acknowledgments

We would like to acknowledge Alex Keller for past instruction in honeypotting techniques, which has informed our design process, and the CS 356 teaching team for suggesting MCP honeypotting as a research topic.

References

[1] Mario Candela. Beelzebub, 2025. https://github.com/mariocandela/beelzebub.

[2] Yongjian Guo et. al. Systematic analysis of mcp security. *arXiv*, 2025. https://arxiv.org/abs/2508.12538.

[3] FastMCP. *FastMCP Documentation*, 2025. https://gofastmcp.com/getting-started/welcome.

[4] Jeremiah Lowin. Fastmcp, 2025. https://github.com/jlowin/fastmcp.

[5] Model Context Protocol. *Specification*, 2025. https://modelcontextprotocol.io/specification/2025-06-18.

[6] M. Rahman et al. Model context protocol (MCP) at first glance: Studying the security and maintainability of MCP servers. *arXiv*, 2025. https://arxiv.org/abs/2506.13538.

[7] A. Singh et al. Towards bio-inspired cyber-deception: A case study of SSH and telnet honeypots. In *Proceedings of the IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*, 2025. https://doi.org/10.1109/EuroSPW67616.2025.00079.

[8] L. Spitzner. Honeypots: Catching the insider threat. *IEEE Security & Privacy*, 2003. https://ieeexplore.ieee.org/abstract/document/1254322.

[9] Starlette. Introduction, 2025. https://starlette.dev.

[10] Z. Zhang et al. Model context protocol (MCP): Landscape, security threats, and future research directions. *arXiv*, 2025. https://arxiv.org/abs/2503.23278.