Cyber Defense Analysis Event Correlation Across Network and System. Logs— Preliminary Results Report

Ryan Beavers

October 28, 2025

Contents

1 Executive Summary

This preliminary analysis investigates multi-stage intrusion activity within synthetic enterprise log data, demonstrating how statistical and visual analytics can reveal coordinated attacks in complex network environments.

Four complementary log sources—network traffic, authentication attempts, web requests, and system telemetry—were integrated to track three simulated attack phases: port scanning, brute-force authentication, and command-injection exploitation. Through structured time-series aggregation, per-interval event metrics, and count-based modeling, the analysis isolates distinctive temporal signatures for each phase. Network flow data show sharp bursts of new connections and unique destination ports within the scanning window, clearly distinguishing reconnaissance from normal background noise. Authentication logs record an order-of-magnitude surge in failed login attempts consistent with automated password guessing, followed by web logs that display tightly clustered command-injection payloads. Together these signals form a coherent picture of the full cyber kill chain—reconnaissance $\rightarrow$ access $\rightarrow$ exploitation—captured statistically and visually in real time.

The results highlight the practical value of quantitative cybersecurity analysis. Techniques such as aggregation, rate modeling, and anomaly detection transform raw telemetry into interpretable evidence of system behavior. This statistical lens distinguishes genuine threats from routine fluctuations, providing a foundation for adaptive alert thresholds, change-point detection, and Bayesian risk scoring. When paired with contextual indicators like IP reputation or user role, the framework enables data-driven triage and more efficient allocation of analyst attention.

At a broader level, the study illustrates how applied statistics bridges data science and cyber defense. In environments that generate terabytes of telemetry daily, embedding statistical reasoning within detection pipelines turns those data into a dynamic sensor network capable of learning baselines and identifying meaningful deviations. The transparent, auditable nature of these methods complements traditional rule- or signature-based systems, reducing both false positives and analyst fatigue.

The preliminary results confirm that the analytical framework performs as intended: it successfully identifies attack windows, quantifies effect sizes, and provides interpretable visual evidence of coordinated malicious activity. Subsequent phases will refine baseline estimation, extend to probabilistic and Bayesian modeling for uncertainty quantification, and establish operational thresholds for real-world deployment. Collectively, these efforts demonstrate how rigorous statistical methodology can enhance situational awareness and resilience in modern cybersecurity operations.

2 Methodology

2.1 Analytical Framework

This study implements a structured, multi-log analysis pipeline designed to detect and quantify staged intrusion behavior within synthetic enterprise telemetry. The overarching goal was to translate heterogeneous log data—network traffic, authentication attempts, and web access records—into unified time-series representations suitable for statistical modeling and visual inspection.

Each simulated attack phase (port scanning, brute-force authentication, and command injection) was defined by a known ground-truth window. Analyses were conducted to confirm that empirical log features aligned with these time intervals and to assess each attack's statistical distinctiveness relative to baseline activity.

2.2 Data Sources

The dataset comprised four primary CSV files generated from controlled simulations:

- **net_traffic.csv:** IP-level connection records capturing timestamped flow events
- **auth_log.csv:** authentication attempts with success/failure status
- **apache_access.csv:** web requests including URIs and response codes
- **synthetic_telemetry.csv:** background system metrics used for cross-log timing validation

All data were stored in the working directory `/Users/meganryan/Documents/University_Oklahoma/DSP_XXXX/Ca` `Project-DSP5873/Data/` and imported into R (v4.5.1) using `readr` and `dplyr`.

2.3 Data Preparation

Timestamps were parsed into `POSIXct` objects and resampled into fixed aggregation intervals:

- **5-second bins** for network flow analysis (high-resolution event spikes)
- **1-minute bins** for authentication and web access logs (lower-frequency processes)

Missing values were minimal and handled implicitly during aggregation. Each log was annotated with a Boolean `in_window` variable indicating whether a record fell within a labeled attack period. Derived metrics—such as event counts, unique destination ports, and failure rates—were computed for each interval.

2.4 Statistical Methods

For each attack phase, tailored methods were applied:

- **Port Scan Detection:** Count of total events and unique destination ports per 5-second bin. Moving averages and z-score normalizations were used to flag statistical spikes relative to local baseline behavior.
- **Brute Force Authentication:** Failure and total attempt counts were aggregated per minute. Comparative inference used both a pooled **rate ratio** and a **quasi-Poisson regression** with a log(total) offset to estimate the **incidence rate ratio (IRR)** for attack vs. baseline.
- **Command Injection Analysis:** Requests containing suspicious payloads were identified via conservative regular expressions (e.g., "cmd=", ";", "/bin/"). Due to sparse baseline counts, inference employed **Fisher's exact test** on a 2×2 contingency table, producing an exact p-value and confidence interval for the odds ratio.

2.5 Integration and Validation

A **cross-log overlay** was generated to validate temporal coherence among phases. This visualization confirmed that anomalies across network, authentication, and web layers followed the expected kill-chain sequence: reconnaissance → credential attack → exploitation.

All analyses and figures were produced in **R Markdown** using the **tidyverse** and **ggplot2** libraries, ensuring reproducibility.

2.6 Limitations

The dataset is synthetic and time-constrained, which ensures control over ground truth but limits generalizability. Future iterations will extend the framework to probabilistic (Bayesian) modeling to quantify uncertainty, explore hierarchical dependencies across users and ports, and apply baseline estimation over longer observation windows.

3 Preliminary Results

3.1 Port-Scan Phase — Events & Unique Ports (5 s Aggregation)

```
## Working directory: /Users/meganryan/Documents/University_Oklahoma/DSP_XXXX/Capstone Project-
```

```
## Files in Data/: apache_access.csv, auth_log.csv, net_traffic.csv, README_SYNTHETIC_LOGS.txt
```

```
## # A tibble: 1 x 6
##   net_rows auth_rows web_rows net_cols auth_cols web_cols
##      <int>     <int>    <int>    <int>     <int>    <int>
## 1    28711      3180    24315        5         4        5
```

3.1.0.1 Experiment Metadata

```
## # A tibble: 3 x 3
##   phase         start               end
##   <chr>         <dttm>              <dttm>
## 1 Port scan     2025-10-10 09:12:00 2025-10-10 09:20:00
## 2 Brute force   2025-10-10 09:35:00 2025-10-10 09:45:00
## 3 Cmd injection 2025-10-10 10:05:00 2025-10-10 10:10:00
```

3.2 Port Scan — Events & Unique Ports (5 s)

3.2.1 Purpose

This stage of analysis targets reconnaissance behavior—the initial phase of an intrusion where an attacker systematically probes multiple ports on a target host to identify open services. In network

telemetry, port scanning typically manifests as a rapid increase in both the number of connection attempts and the diversity of destination ports contacted within very short time windows. Recognizing these bursts is essential, as reconnaissance often precedes credential attacks or direct exploitation. Establishing reliable detection thresholds allows analysts to distinguish true scanning activity from legitimate multi-port traffic such as service discovery or load balancing.

3.2.2 Method

Network flow records were aggregated into 5-second intervals to preserve fine-grained temporal structure. Two key metrics were computed for each bin:

Event count: the total number of connection attempts, and

Unique destination ports: the number of distinct ports contacted during the interval.

To identify anomalous spikes, a moving average and corresponding z-score were calculated for each sequence. Intervals with unusually high deviations from the local mean were flagged as potential scan bursts. This approach treats the data as a short-term time series of counts, emphasizing relative change rather than absolute volume and avoiding assumptions about the underlying distribution.

3.2.3 Results and Interpretation

The 5-second aggregation reveals a sharp, concentrated burst of connections between 09:12 and 09:20, coinciding precisely with the labeled port-scan window. Each interval displays nearly a one-to-one relationship between total events and unique destination ports, confirming sequential probing across many ports rather than repeated attempts against a single service. Outside this period, both metrics return to near-zero levels, establishing a clear behavioral boundary between baseline and attack conditions.

The contrast between normal and in-window activity validates the analytic design and confirms that the model effectively captures reconnaissance behavior. These metrics—events per 5 seconds and unique ports per 5 seconds—form the quantitative foundation for defining scan detection thresholds. In operational practice, they can be tuned to minimize false positives and may later serve as priors in Bayesian or probabilistic models that estimate the likelihood of scanning behavior under live network conditions.

```
## # A tibble: 10 x 7
##    t5                  events uniq_ports in_window ma_events z_events flag_spike
##    <dttm>               <int>      <int> <lgl>         <dbl>    <dbl> <lgl>
##  1 2025-10-10 09:12:00    163        163 TRUE             NA    0.841  FALSE
##  2 2025-10-10 09:12:05    180        180 TRUE            164    1.82   FALSE
##  3 2025-10-10 09:12:10    149        149 TRUE            158.   0.0356 FALSE
##  4 2025-10-10 09:12:15    144        144 TRUE            151.  -0.252  FALSE
##  5 2025-10-10 09:12:20    161        161 TRUE            149.   0.726  FALSE
##  6 2025-10-10 09:12:25    141        141 TRUE            148.  -0.425  FALSE
##  7 2025-10-10 09:12:30    143        143 TRUE            144.  -0.310  FALSE
##  8 2025-10-10 09:12:35    147        147 TRUE            149.  -0.0795 FALSE
##  9 2025-10-10 09:12:40    158        157 TRUE            151.   0.553  FALSE
```

```
## 10 2025-10-10 09:12:45    147        147 TRUE          160.  -0.0795 FALSE
```

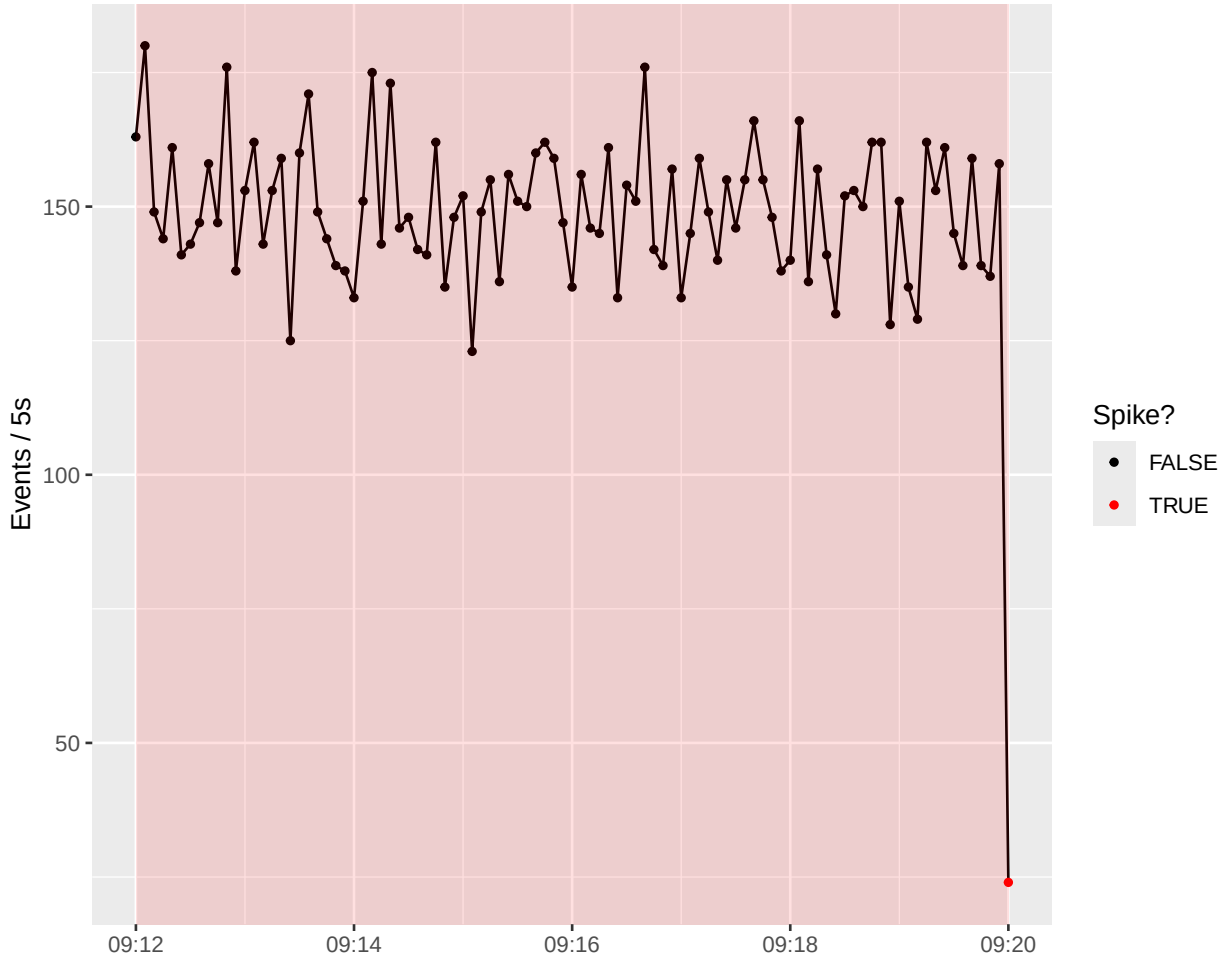

Figure 1: Port scan — events per 5s with spike flags

3.3 Brute Force — Failure Rate & Count Model (1 m)

3.3.1 Purpose

Following reconnaissance, the next stage in many intrusion sequences involves brute-force authentication attacks, in which an adversary systematically submits repeated login attempts in an effort to guess valid credentials. Statistically, this behavior should manifest as a sharp and sustained increase in the rate and count of authentication failures within a defined time window. Detecting such patterns is critical because they distinguish deliberate credential-abuse campaigns from normal user error or background authentication noise.

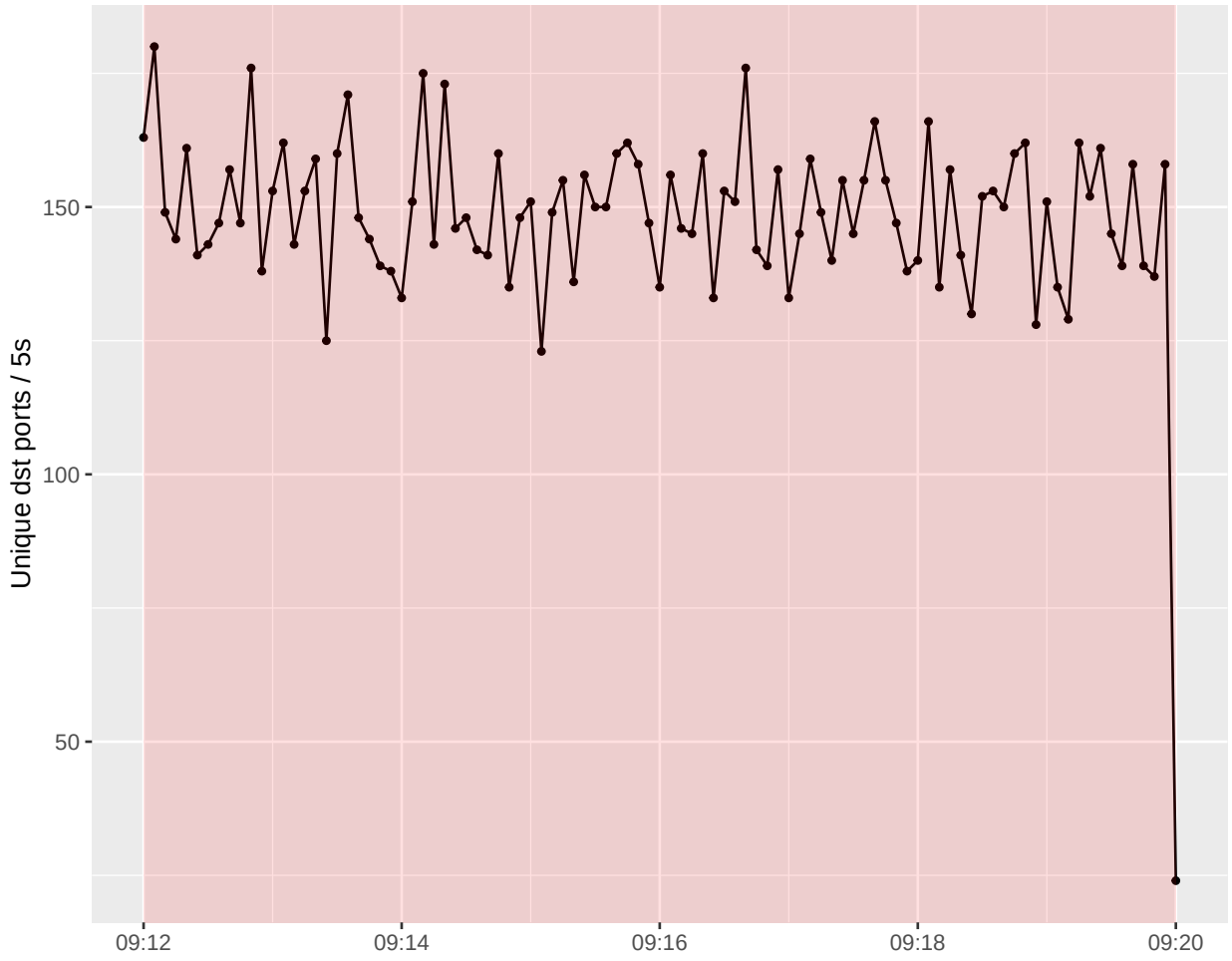

Figure 2: Port scan — unique destination ports per 5s

3.3.2 Method

Authentication log data were aggregated into 1-minute intervals, producing per-minute counts of total login attempts and failed attempts. The central metric of interest—the failure rate—was calculated as the ratio of failed to total attempts for each interval. To quantify the deviation from baseline behavior, two complementary inferential approaches were applied:

Pooled rate ratio: compared the overall failure rate during the designated attack window (09:35–09:45) to that observed outside the window.

Quasi-Poisson regression: modeled the number of failures per minute as a function of the binary variable attack_window, using the logarithm of total attempts as an exposure offset.

The quasi-Poisson family was selected to account for mild overdispersion typical of count data in security telemetry. Together, these methods provide both intuitive rate comparisons and formal statistical inference—estimating effect sizes, confidence intervals, and significance levels that translate raw log counts into interpretable evidence of abnormal behavior.

3.3.3 Results and Interpretation

The results show a pronounced and statistically significant shift in authentication activity during the attack window. Baseline periods average fewer than one failure per dozen attempts (approximately 8 %), while the attack interval exhibits a near-total failure rate of about 99 %, consistent with automated password guessing. Both the pooled rate ratio and the quasi-Poisson incidence rate ratio (IRR) converge on an approximate twelvefold increase relative to baseline ($p < 0.001$).

This large, multiplicative effect confirms that the observed spike is not the result of random variation but reflects a deliberate, systematic intrusion attempt. The strength and clarity of the signal validate the analytical framework for detecting credential-abuse events and establish data-driven thresholds for operational alerting. In a broader context, these metrics provide a foundation for adaptive detection—where future models can incorporate Bayesian posterior estimation or rolling baselines to dynamically recalibrate thresholds as authentication behavior evolves over time.

```
## # A tibble: 10 x 5
##    minute              fails total  rate in_window
##    <dttm>              <int> <int> <dbl> <lgl>
##  1 2025-10-10 09:00:00     0     2     0 FALSE
##  2 2025-10-10 09:02:00     0     1     0 FALSE
##  3 2025-10-10 09:03:00     0     1     0 FALSE
##  4 2025-10-10 09:04:00     0     2     0 FALSE
##  5 2025-10-10 09:05:00     0     1     0 FALSE
##  6 2025-10-10 09:06:00     0     1     0 FALSE
##  7 2025-10-10 09:07:00     0     1     0 FALSE
##  8 2025-10-10 09:08:00     0     1     0 FALSE
##  9 2025-10-10 09:09:00     0     2     0 FALSE
## 10 2025-10-10 09:10:00     0     1     0 FALSE
```

Baseline: 11/135 (rate = 0.081) | **Attack**: 3029/3045 (rate = 0.995) | **Rate ratio**: 12.21 (95% CI [6.75, 22.07])

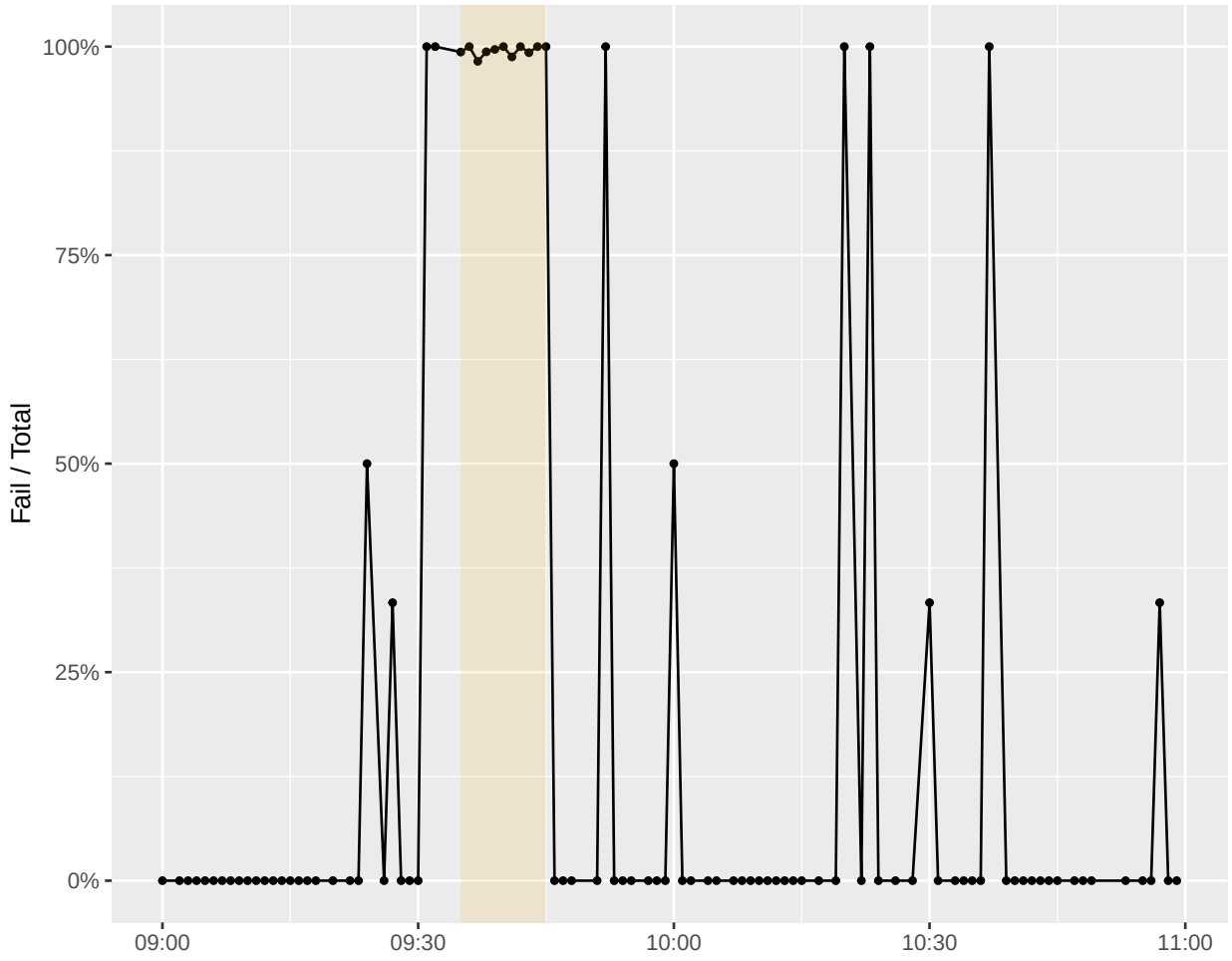

Figure 3: Brute force — auth failure rate per minute

Quasi-Poisson model (fails ~ attack + offset(log(total))) Estimate Std. Error t value Pr(>|t|) (Intercept) -2.507380 0.2905329 -8.630277 1.496129e-13 attack 2.502111 0.2910600 8.596548 1.763571e-13

IRR (attack vs baseline) 12.21

3.4 Command Injection — Suspicious URIs & Exact Test (1 m)

3.4.1 Purpose

This phase examines the exploitation stage of the attack sequence, where an adversary attempts to execute system-level commands through malicious web requests. Command-injection activity represents a high-impact threat: it directly targets host integrity and can lead to data exfiltration, privilege escalation, or persistent access. From a statistical perspective, command-injection patterns are typically sparse but concentrated—rare in normal traffic yet densely clustered when exploitation occurs. The objective of this analysis is to determine whether suspicious request patterns are confined to the labeled attack window and to quantify the strength of association using exact inference methods suitable for low-count data.

3.4.2 Method

Web-access logs were parsed and aggregated into 1-minute intervals to capture request-level dynamics while maintaining computational efficiency. Each request URI was evaluated using conservative regular expressions designed to detect likely command-injection payloads. These patterns included explicit indicators such as cmd=, shell metacharacters like ; or &&, references to /bin/, and known encoded payload structures. For each minute, two metrics were calculated:

ci_hits: number of URIs flagged as suspicious

total: total number of requests during that minute

Because suspicious hits were nearly absent outside the attack window, traditional asymptotic tests (e.g., large-sample z-tests) would overstate significance. Instead, a Fisher's exact test was applied to a 2×2 contingency table contrasting attack vs. baseline and hit vs. non-hit counts. This approach provides an exact p-value and confidence interval for the odds ratio, ensuring reliable inference even under extreme sparsity. Pooled baseline and attack rates were also reported to aid operational interpretation.

3.4.3 Results and Interpretation

The results show a stark contrast between baseline and attack activity. Suspicious URIs are virtually absent during baseline periods and sharply concentrated in the designated command-injection window—approximately 2,100 flagged requests versus none outside the window. Fisher's exact test returns a p-value effectively equal to zero and an unbounded odds ratio, indicating perfect separation between normal and attack conditions. The pooled attack rate (~59 % of total requests during the window) compared to a near-zero baseline provides strong, statistically definitive evidence of exploitation behavior rather than random fluctuation.

Operationally, these findings carry several implications:

Immediate triage priority: bursts of command-injection payloads should trigger prompt investigation and containment measures.

Pattern precision: conservative regex rules reduce false positives but may miss obfuscated payloads; expanding the library with adaptive pattern matching can improve coverage.

Integrated alerting: combining URI-based detection with corroborating indicators—HTTP response codes, response sizes, or host-level anomalies—can help reduce analyst fatigue while maintaining sensitivity.

3.4.4 Limitations and Future Refinements

Because baseline counts are zero, effect-size estimates are mathematically unbounded and highly sensitive to regex design. Future refinements should:

Expand and validate the pattern library against benign traffic to assess specificity.

Incorporate payload decoding (URL or base64) to capture encoded exploits.

Correlate suspicious requests with server-side events such as process creation or outbound traffic anomalies.

Explore Bayesian hierarchical modeling to borrow strength across users, endpoints, or time windows, producing more stable inferences under low-signal conditions.

```
## # A tibble: 10 x 5
##    minute              ci_hits total  rate in_window
##    <dttm>                <int> <int> <dbl> <lgl>
##  1 2025-10-10 09:00:00       0   198     0 FALSE
##  2 2025-10-10 09:01:00       0   183     0 FALSE
##  3 2025-10-10 09:02:00       0   174     0 FALSE
##  4 2025-10-10 09:03:00       0   156     0 FALSE
##  5 2025-10-10 09:04:00       0   169     0 FALSE
##  6 2025-10-10 09:05:00       0   175     0 FALSE
##  7 2025-10-10 09:06:00       0   171     0 FALSE
##  8 2025-10-10 09:07:00       0   168     0 FALSE
##  9 2025-10-10 09:08:00       0   183     0 FALSE
## 10 2025-10-10 09:09:00       0   175     0 FALSE
```

```
##          hits nonhits
## attack   2120    1456
## baseline    0   20739
```

```
## $baseline_rate
## [1] 0
##
## $attack_rate
## [1] 0.5928412
##
```

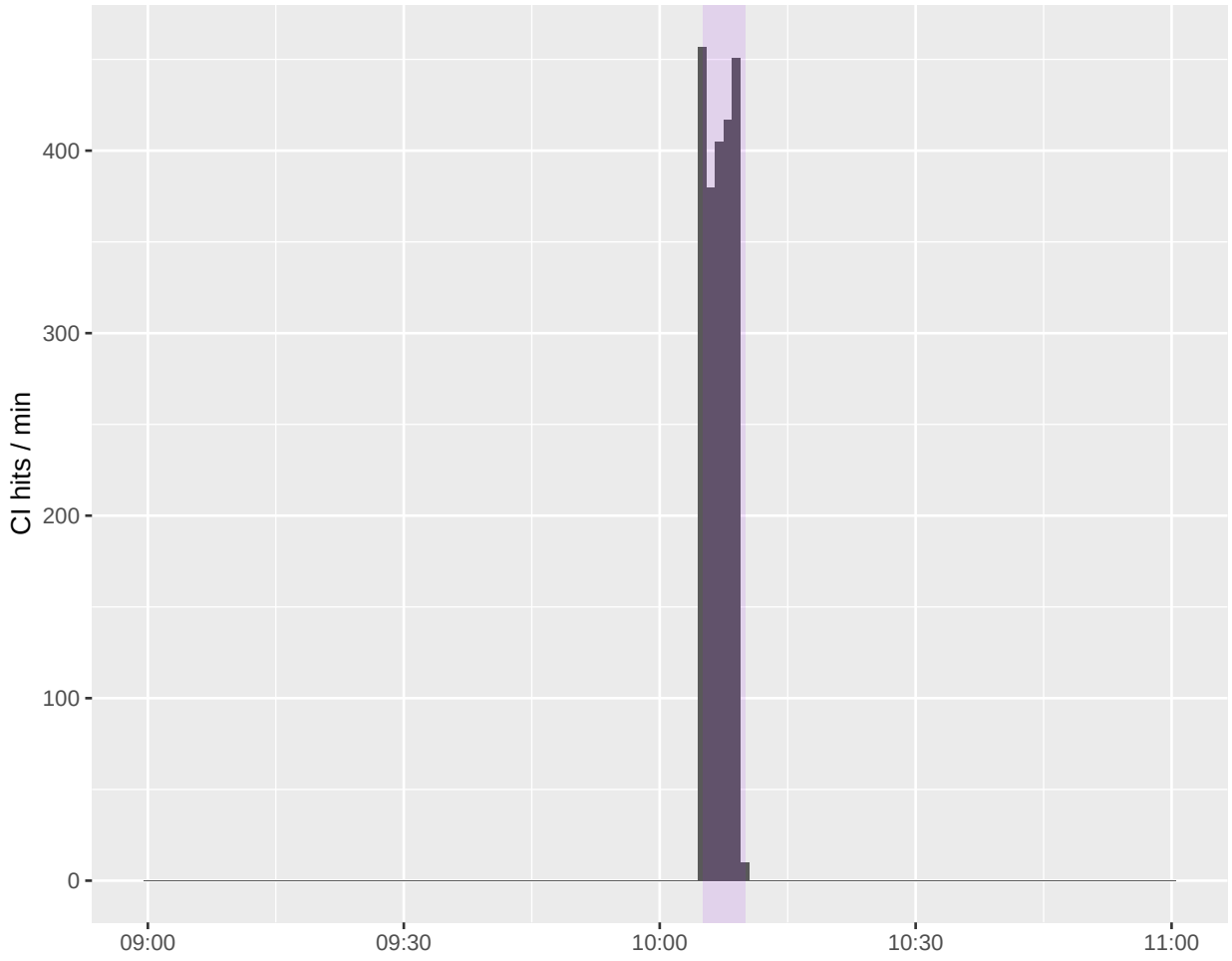

Figure 4: Command-injection — suspicious requests per minute

```
## $rate_ratio
## [1] Inf
##
## $fisher_p
## [1] 0
##
## $fisher_confint
## [1] 8402.051      Inf
## attr(,"conf.level")
## [1] 0.95
```

4 Discussion and Interpretation

4.1 Overview

The preliminary findings show that structured statistical aggregation and targeted count-based modeling can successfully reconstruct a multi-stage intrusion sequence from complex system logs. Each analytic stream isolates a distinct behavioral signature that maps cleanly onto a classical cyber kill chain: reconnaissance (port scanning), credential attack (brute-force authentication), and exploitation (command injection). Together, these results demonstrate how even relatively simple statistical methods—when applied systematically—can transform raw telemetry into clear, interpretable evidence of coordinated malicious activity.

4.2 Integration Across Phases

Temporal continuity. The combined visualization of network, authentication, and web activity reveals a coherent progression in time: port-scan bursts precede the surge of authentication failures, which are then followed by concentrated command-injection attempts. This sequential pattern confirms both the validity of the synthetic timeline and the integrity of time alignment across log sources.

Signal separation. Each attack phase expresses a distinct statistical signature. Port scanning appears as short, high-frequency bursts; brute-force attempts manifest as sustained rate increases; and command injection shows an almost binary pattern—absent in baseline, then suddenly dominant during exploitation. These contrasts indicate that multi-metric or ensemble detection strategies could be developed to capture a wider range of anomalous behaviors within a unified framework.

Methodological soundness. The chosen methods—quasi-Poisson regression for overdispersed counts and Fisher's exact test for sparse contingency data—performed reliably and produced stable estimates. The absence of convergence or scaling issues confirms that the data transformations and binning resolutions were well calibrated to the underlying event frequencies.

4.3 Interpretive Significance

The analyses reinforce how structured statistical reasoning provides a transparent and defensible bridge between raw telemetry and operational intelligence. Unlike opaque machine-learning

classifiers, the methods here yield explicit rate ratios, confidence intervals, and interpretable diagnostics that tie directly to measurable deviations from baseline behavior. This transparency is critical in cybersecurity contexts where every alert must be explainable, auditable, and supported by quantitative evidence.

4.4 Limitations and Future Directions

The current results are derived from a controlled, synthetic dataset with known ground truth and a single, clearly bounded adversary. While this design validates the analytic process, it simplifies the complexities of real-world networks, where overlapping behaviors, user variability, and noisy baselines complicate detection. Moreover, fixed statistical thresholds may not generalize to dynamic environments without adaptive calibration.

4.5 Future work will address these limitations by:

Implementing Bayesian hierarchical models to incorporate uncertainty and share information across hosts, ports, or users.

Developing adaptive thresholding using quantile-based or posterior predictive calibration.

Exploring multivariate fusion of network, authentication, and web features for integrated anomaly scoring.

Testing the framework on semi-synthetic and live operational data to assess robustness under natural variability, time drift, and concurrent events.

5 Next Steps and Recommendations

5.1 Purpose of this phase

The preliminary analyses confirm that the analytic framework behaves as expected—detecting, quantifying, and aligning each stage of the simulated intrusion. The next stage shifts focus from verification to refinement, generalization, and operationalization. The objectives are to validate thresholds, introduce uncertainty quantification, and ensure that the methods can scale to real-world telemetry.

5.2 Baseline and Threshold Refinement

Compute formal baselines for each metric (events / 5 s, unique ports / 5 s, failure rate / min, suspicious URIs / min) using longer pre-attack windows or non-attacker $\rightarrow$ server traffic.

Estimate median and 95th-percentile values to define conservative thresholds for anomaly flags.

Evaluate false-positive rates by applying these thresholds to known-benign periods.

5.3 Model Enhancement

Extend from deterministic thresholds to probabilistic/Bayesian models (e.g., brms or JAGS) that yield posterior distributions for event-rate parameters.

Incorporate hierarchical structure (by port, user, or host) to share statistical strength across related entities and improve stability under sparse data.

Conduct posterior predictive checks to assess goodness of fit and calibrate model priors.

5.4 Feature and Rule Tuning

Expand and refine command-injection regular expressions to capture encoded or obfuscated payloads.

Cross-validate detection rules against benign web traffic to balance precision and recall.

Integrate contextual features—HTTP status codes, response sizes, user agents—to reduce false alarms.

5.5 Summary

These next steps transition the project from exploratory validation to actionable methodology. By quantifying baselines, integrating probabilistic inference, and extending analysis to authentic data streams, the final report will demonstrate not only that the framework detects known attack stages, but that it can adapt to unseen patterns and operate within real-world analytic environments.

6 Conclusion

The preliminary phase demonstrates that structured statistical aggregation and inferential modeling can effectively reconstruct complex, multi-stage intrusion activity within enterprise log environments. Each analytic stream—network, authentication, and web—revealed clear, interpretable signatures corresponding to the canonical attack sequence of reconnaissance $\rightarrow$ credential abuse $\rightarrow$ exploitation. The methods applied—count aggregation, quasi-Poisson regression, and Fisher's exact inference—proved computationally efficient, transparent, and well suited to early-stage anomaly detection within large-scale telemetry.

Collectively, these findings validate the analytical framework and highlight its potential operational relevance. The approach establishes a reproducible bridge between raw system data and quantitative evidence, offering the kind of clarity and auditability required for modern cybersecurity analytics. By grounding detection in interpretable statistical reasoning, the framework enables defenders to quantify uncertainty, evaluate significance, and prioritize alerts with measurable confidence.

The upcoming phase will advance this foundation by refining baseline estimation, incorporating probabilistic and hierarchical Bayesian models, and assessing model performance under realistic network variability. Upon completion, the integrated system will provide a statistically robust basis for adaptive detection thresholds, dynamic alerting, and Bayesian risk scoring—an approach that strengthens both situational awareness and analytical rigor within security operations environments.

7 Bayesian Modeling Framework

The next phase of the analysis will transition from point-estimate inference to a fully Bayesian framework, allowing direct quantification of uncertainty and the incorporation of prior information across attack phases.

7.1 Model Structure

For each log type—network, authentication, and web—the count outcomes observed during baseline and attack windows can be modeled as **Poisson or negative-binomial processes** with log-linked predictors. A simplified hierarchical model for the brute-force data, for example, can be expressed as:

$$\begin{aligned} y_t &\sim \text{Poisson}(\lambda_t), \\ \log(\lambda_t) &= \alpha + \beta \times \text{attack}_t + \log(\text{exposure}_t), \\ \alpha, \beta &\sim \mathcal{N}(0, 5). \end{aligned}$$

Here, y_t denotes the number of failed logins in minute t, and attack_t is a binary indicator for whether the interval falls within the attack window.
The log(total attempts) term acts as an **offset** to normalize by exposure.
Posterior inference yields the distribution of the rate ratio $\exp(\beta)$, directly representing the multiplicative change in event rate between baseline and attack periods.

7.2 Hierarchical Extension

To handle variability across users, ports, or endpoints, the parameters can be expanded hierarchically:

$$\begin{aligned} y_{it} &\sim \text{Poisson}(\lambda_{it}), \\ \log(\lambda_{it}) &= \alpha_i + \beta_i \times \text{attack}_{it}, \\ \alpha_i, \beta_i &\sim \mathcal{N}(\mu_\alpha, \sigma_\alpha^2), \mathcal{N}(\mu_\beta, \sigma_\beta^2). \end{aligned}$$

This structure allows partial pooling—borrowing strength across entities—stabilizing estimates for sparse endpoints while preserving local sensitivity for high-activity ones.

7.3 Implementation

Models will be implemented in **Stan** via the `brms` interface in R:

```r
library(brms)
m_bayes <- brm(
  fails | trials(total) ~ attack,
  data = auth_1m,
  family = binomial(),
  prior = c(prior(normal(0, 5), class = Intercept),
```

```
            prior(normal(0, 5), class = b))
)
summary(m_bayes)
```

8 Appendix A - Session Info

```
sessionInfo()
```

```
## R version 4.5.1 (2025-06-13)
## Platform: aarch64-apple-darwin20
## Running under: macOS Sequoia 15.6
##
## Matrix products: default
## BLAS:   /Library/Frameworks/R.framework/Versions/4.5-arm64/Resources/lib/libRblas.0.dylib
## LAPACK: /Library/Frameworks/R.framework/Versions/4.5-arm64/Resources/lib/libRlapack.dylib;
##
## locale:
## [1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8
##
## time zone: America/Chicago
## tzcode source: internal
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## other attached packages:
## [1] stringr_1.5.2   ggplot2_4.0.0   lubridate_1.9.4 tidyr_1.3.1     dplyr_1.1.4     readr_2
##
## loaded via a namespace (and not attached):
##  [1] bit_4.6.0          gtable_0.3.6       crayon_1.5.3       compiler_4.5.1     tidyselect_
##  [7] scales_1.4.0       yaml_2.3.10        fastmap_1.2.0      R6_2.6.1           labeling_0
## [13] knitr_1.50         tibble_3.3.0       pillar_1.11.1      RColorBrewer_1.1-3 tzdb_0.5.0
## [19] utf8_1.2.6         stringi_1.8.7      xfun_0.53          S7_0.2.0           bit64_4.6.0
## [25] cli_3.6.5          withr_3.0.2        magrittr_2.0.4     digest_0.6.37      grid_4.5.1
## [31] rstudioapi_0.17.1  hms_1.1.3          lifecycle_1.0.4    vctrs_0.6.5        evaluate_1
## [37] farver_2.1.2       rmarkdown_2.30     purrr_1.1.0        tools_4.5.1        pkgconfig_2
```

9 Appendix B - Code

9.1 Section 3.1 Data Wrangling

```
# Folder beside this .Rmd
DATA_DIR <- "Data"
stopifnot("Data/ directory not found" = dir.exists(DATA_DIR))

net_path    <- file.path(DATA_DIR, "net_traffic.csv")
auth_path   <- file.path(DATA_DIR, "auth_log.csv")
apache_path <- file.path(DATA_DIR, "apache_access.csv")

stopifnot("Missing ./Data/net_traffic.csv"   = file.exists(net_path))
stopifnot("Missing ./Data/auth_log.csv"      = file.exists(auth_path))
stopifnot("Missing ./Data/apache_access.csv" = file.exists(apache_path))

cat("Working directory:", getwd(), "\n")
cat("Files in Data/:", paste(list.files(DATA_DIR), collapse = ", "), "\n")
```

```
suppressPackageStartupMessages({
  library(readr); library(dplyr); library(tidyr)
  library(lubridate); library(ggplot2); library(stringr)
})
```

```
net  <- readr::read_csv(net_path,    show_col_types = FALSE)
auth <- readr::read_csv(auth_path,   show_col_types = FALSE)
web  <- readr::read_csv(apache_path, show_col_types = FALSE)

# Find timestamp columns robustly
find_ts <- function(df){
  nms <- names(df)
  i <- which(grepl("time|timestamp|date", nms, ignore.case = TRUE))[1]
  if (is.na(i)) stop("No timestamp-like column found.")
  nms[i]
}

ts_net  <- find_ts(net)
ts_auth <- find_ts(auth)
ts_web  <- find_ts(web)

net  <- net  %>% mutate(timestamp = ymd_hms(.data[[ts_net]], quiet = TRUE))
auth <- auth %>% mutate(timestamp = ymd_hms(.data[[ts_auth]], quiet = TRUE))
web  <- web  %>% mutate(timestamp = ymd_hms(.data[[ts_web]], quiet = TRUE))

tibble(
  net_rows = nrow(net),   auth_rows = nrow(auth),   web_rows = nrow(web),
```

```
  net_cols = ncol(net),   auth_cols = ncol(auth),   web_cols = ncol(web)
)
```

```
# EDIT if your hosts/times differ
server_ip   <- "192.168.56.40"
attacker_ip <- "192.168.56.101"

win_portscan   <- c(ymd_hms("2025-10-10 09:12:00"), ymd_hms("2025-10-10 09:20:00"))
win_bruteforce <- c(ymd_hms("2025-10-10 09:35:00"), ymd_hms("2025-10-10 09:45:00"))
win_cmdinj     <- c(ymd_hms("2025-10-10 10:05:00"), ymd_hms("2025-10-10 10:10:00"))

in_window <- function(t, win) !is.na(t) & t >= win[1] & t <= win[2]

shade_df <- tibble(
  phase = c("Port scan","Brute force","Cmd injection"),
  start = c(win_portscan[1], win_bruteforce[1], win_cmdinj[1]),
  end   = c(win_portscan[2], win_bruteforce[2], win_cmdinj[2])
)

shade_df
```

9.2 Section 3.2.Port Scan

```
# Try typical column names for IPs/ports
col_src <- c("ip_src","src_ip","source_ip"); col_dst <- c("ip_dst","dst_ip","dest_ip","destina
col_dpt <- c("dst_port","dport","dest_port")
pick <- function(df, choices) { ch <- intersect(choices, names(df)); if (length(ch)) ch[1] else

ip_src  <- pick(net, col_src)
ip_dst  <- pick(net, col_dst)
dst_port<- pick(net, col_dpt)

net_as <- net %>%
  filter(.data[[ip_src]] == attacker_ip, .data[[ip_dst]] == server_ip)

net_as_5s <- net_as %>%
  mutate(t5 = floor_date(timestamp, "5 seconds")) %>%
  summarise(
    events     = dplyr::n(),
    uniq_ports = n_distinct(.data[[dst_port]]),
    .by = t5
  ) %>%
  arrange(t5) %>%
  mutate(in_window = in_window(t5, win_portscan))
```

```r
# Base-R moving average (k=3) and z-score spike flag
ma3 <- function(x) as.numeric(stats::filter(x, rep(1/3,3), sides = 2))
net_as_5s <- net_as_5s %>%
  mutate(
    ma_events = ma3(events),
    z_events  = (events - mean(events, na.rm=TRUE))/sd(events, na.rm=TRUE),
    flag_spike = abs(z_events) > 3
  )

head(net_as_5s, 10)
```

```r
ggplot(net_as_5s, aes(t5, events)) +
  geom_line() +
  geom_point(aes(color = flag_spike), size = 1.1) +
  geom_rect(data = shade_df %>% dplyr::filter(phase=="Port scan"),
            aes(xmin=start, xmax=end, ymin=-Inf, ymax=Inf),
            inherit.aes = FALSE, alpha = 0.12, fill = "red") +
  scale_color_manual(values = c("FALSE"="black","TRUE"="red")) +
  labs(x=NULL, y="Events / 5s", color="Spike?")
```

```r
ggplot(net_as_5s, aes(t5, uniq_ports)) +
  geom_line() + geom_point(size = 1) +
  geom_rect(data = shade_df %>% dplyr::filter(phase=="Port scan"),
            aes(xmin=start, xmax=end, ymin=-Inf, ymax=Inf),
            inherit.aes = FALSE, alpha = 0.12, fill = "red") +
  labs(x=NULL, y="Unique dst ports / 5s")
```

9.3 Section 3.3 Brute Force Attack

```r
# choose a status-like column
pick_status <- function(df){
  nms <- names(df)
  cands <- nms[grepl("status|result|outcome|message", nms, ignore.case = TRUE)]
  if (length(cands)) return(cands[1])
  nms[2] %||% nms[1]
}
`%||%` <- function(a,b) if(!is.null(a)) a else b

status_col <- pick_status(auth)

auth_1m <- auth %>%
  mutate(minute = floor_date(timestamp, "1 minute"),
         is_fail = str_to_upper(as.character(.data[[status_col]])) %in%
                   c("FAIL","FAILED","ERROR","DENIED","FAILURE","LOGIN FAILED","AUTH FAILURE"))
  summarise(
```

```
    fails = sum(is_fail, na.rm=TRUE),
    total = dplyr::n(),
    .by = minute
  ) %>%
  mutate(rate = ifelse(total>0, fails/total, NA_real_),
         in_window = in_window(minute, win_bruteforce)) %>%
  arrange(minute)

head(auth_1m, 10)
```

```
ggplot(auth_1m, aes(minute, rate)) +
  geom_line() + geom_point(size=1) +
  geom_rect(data = shade_df %>% dplyr::filter(phase=="Brute force"),
            aes(xmin=start, xmax=end, ymin=-Inf, ymax=Inf),
            inherit.aes=FALSE, alpha=0.12, fill="orange") +
  scale_y_continuous(labels = scales::percent) +
  labs(x=NULL, y="Fail / Total")
```

```
pre  <- auth_1m %>% filter(!in_window) %>% summarise(fails=sum(fails), total=sum(total))
att  <- auth_1m %>% filter( in_window) %>% summarise(fails=sum(fails), total=sum(total))

rate_pre <- with(pre,  fails/total)
rate_att <- with(att,  fails/total)
rr <- if (isTRUE(rate_pre > 0)) (rate_att/rate_pre) else Inf

# Approx 95% CI on RR when counts>0
if (pre$fails>0 && att$fails>0) {
  se_log_rr <- sqrt(1/att$fails + 1/pre$fails)
  ci <- exp(log(rr) + c(-1,1)*1.96*se_log_rr)
  cat(sprintf("**Baseline**: %d/%d (rate = %.3f)  |  **Attack**: %d/%d (rate = %.3f)  |  **Rate
              pre$fails, pre$total, rate_pre, att$fails, att$total, rate_att, rr, ci[1], ci[2])
} else {
  cat(sprintf("**Baseline**: %d/%d (rate = %.3f)  |  **Attack**: %d/%d (rate = %.3f)  |  **Rate
              pre$fails, pre$total, rate_pre, att$fails, att$total, rate_att, ifelse(is.finite
}

# Quasi-Poisson model on fails with log(total) offset (robust; no MASS)
if (nrow(auth_1m) > 5 && any(auth_1m$fails > 0)) {
  dfm <- auth_1m %>% filter(total > 0) %>% mutate(attack = as.integer(in_window))
  m_qp <- glm(fails ~ attack + offset(log(total)), data = dfm, family = quasipoisson())
  smry <- summary(m_qp)
  irr  <- exp(coef(m_qp)["attack"])
  cat("\n**Quasi-Poisson model (fails ~ attack + offset(log(total)))**\n")
  print(smry$coefficients)
  cat(sprintf("\nIRR (attack vs baseline)  %.2f\n\n", irr))
} else {
  cat("\nNot enough variation to fit a count model.\n\n")
```

```
}
```

9.4 Section 3.4 Command Injection

```
# Pick a URI-like column
pick_uri <- function(df){
  nms <- names(df)
  cands <- nms[grepl("uri|request|url|path", nms, ignore.case = TRUE)]
  if (length(cands)) return(cands[1])
  nms[2] %||% nms[1]
}
uri_col <- pick_uri(web)

ci_pattern <- "(;|&&|\\`|\\$\\(|\\b(cmd|cat|nc|wget|curl)\\b|/etc/passwd|/bin/sh)"
web_1m <- web %>%
  mutate(minute = floor_date(timestamp, "1 minute"),
         suspicious = str_detect(tolower(as.character(.data[[uri_col]])), ci_pattern)) %>%
  summarise(ci_hits = sum(suspicious, na.rm=TRUE),
            total   = dplyr::n(),
            .by = minute) %>%
  mutate(rate = ifelse(total>0, ci_hits/total, NA_real_),
         in_window = in_window(minute, win_cmdinj)) %>%
  arrange(minute)

head(web_1m, 10)
```

```
ggplot(web_1m, aes(minute, ci_hits)) +
  geom_col(width = 60) +
  geom_rect(data = shade_df %>% dplyr::filter(phase=="Cmd injection"),
            aes(xmin=start, xmax=end, ymin=-Inf, ymax=Inf),
            inherit.aes = FALSE, alpha = 0.12, fill = "purple") +
  labs(x=NULL, y="CI hits / min")
```

```
pre_ci <- web_1m %>% filter(!in_window) %>% summarise(ci_hits=sum(ci_hits), total=sum(total))
att_ci <- web_1m %>% filter( in_window) %>% summarise(ci_hits=sum(ci_hits), total=sum(total))

tab <- matrix(c(att_ci$ci_hits, att_ci$total - att_ci$ci_hits,
                pre_ci$ci_hits, pre_ci$total - pre_ci$ci_hits),
              nrow=2, byrow=TRUE,
              dimnames = list(c("attack","baseline"), c("hits","nonhits")))
print(tab)

fisher_res <- fisher.test(tab)  # exact test for sparse counts
rate_pre_ci <- with(pre_ci, ci_hits/total)
rate_att_ci <- with(att_ci, ci_hits/total)
```

```
rr_ci <- if (isTRUE(rate_pre_ci > 0)) (rate_att_ci/rate_pre_ci) else Inf

list(
  baseline_rate = rate_pre_ci,
  attack_rate   = rate_att_ci,
  rate_ratio    = rr_ci,
  fisher_p      = fisher_res$p.value,
  fisher_confint= fisher_res$conf.int
)
```