Contents

[3.2] Commitment Escalation Traps

1. Operational Definition: A cognitive bias where individuals, after making an initial small commitment to a request (even a benign one), are more likely to comply with a larger, subsequent request that violates security protocols. This creates a trap where security is gradually eroded.

2. Main Metric & Algorithm:

- **Metric: Escalated Request Ratio (ERR)**. Formula: `ERR = N_escalated / N_initial`, where `N_initial` is the number of initial requests granted and `N_escalated` is the number of those followed by a security-relevant escalated request.
- **Pseudocode:**

  python

```
def calculate_err(access_logs, chat_logs, user_id, time_window='7d'):
    """
    access_logs: Logs from IAM, endpoint security, or cloud consoles.
    chat_logs: Logs from communication platforms like Slack or Teams.
    time_window: The period to analyze.
    """
    # 1. Find initial, seemingly benign requests (e.g., "can you share this doc?")
    initial_requests = query_chat_logs(chat_logs, keywords=["can you share", "send me the'

    escalated_count = 0
    # 2. For each initial request, check if it was followed by a security-related escalati
    for req in initial_requests:
        subsequent_msgs = get_messages_after(chat_logs, req.timestamp, window_hours=48)
        # Look for escalated requests (e.g., higher privileges, disable security)
        if contains_escalation_keywords(subsequent_msgs, ["admin rights", "disable MFA", '
            # 3. Verify if the escalated action was performed by checking access logs
            if action_performed(access_logs, user_id, escalated_action, req.timestamp + de
                escalated_count += 1

    initial_count = len(initial_requests)
    ERR = escalated_count / initial_count if initial_count > 0 else 0
    return ERR
```

- **Alert Threshold:** `ERR > 0.2` (More than 20% of initial grants lead to an escalation).

3. Digital Data Sources (Algorithm Input):

- **Communication Platform API (Slack/Teams):** To search for request patterns in direct messages and group chats. Fields: `user`, `message_text`, `timestamp`, `channel_type`.
- **Identity & Access Management (IAM) Logs (e.g., Azure AD, Okta):** To verify privilege changes. Fields: `user`, `action` (e.g., `assignRole`, `disableMFA`), `target_resource`, `timestamp`.

- **Endpoint/Cloud Security Logs (e.g., CrowdStrike, AWS CloudTrail):** To verify security-disabling actions. Fields: `user`, `event_name` (e.g., `StopLogging`, `DeleteAlarm`), `timestamp`.

4. Human-to-Human Audit Protocol: Conduct a simulated social engineering exercise (with prior ethical approval) where a red team member makes a small initial request (e.g., asking for a document link) and later follows up with a request that requires a minor policy bypass. Measure the success rate. Alternatively, in interviews, ask: "Has anyone ever asked you for a small favor that later turned into a much bigger request affecting security? Can you describe what you did?".

5. Recommended Mitigation Actions:

- **Technical/Digital Mitigation:** Implement workflow automation that requires manager approval for specific high-risk actions (e.g., role assignment, security policy changes) regardless of how the request was made.
- **Human/Organizational Mitigation:** Conduct training focused on the "commitment and consistency" principle, teaching staff to be wary of requests that escalate in privilege or sensitivity and to report them.
- **Process Mitigation:** Establish a clear protocol that any security-related request, regardless of how small or who it comes from, must be routed through the official ticketing system (Jira, ServiceNow) for tracking and approval.