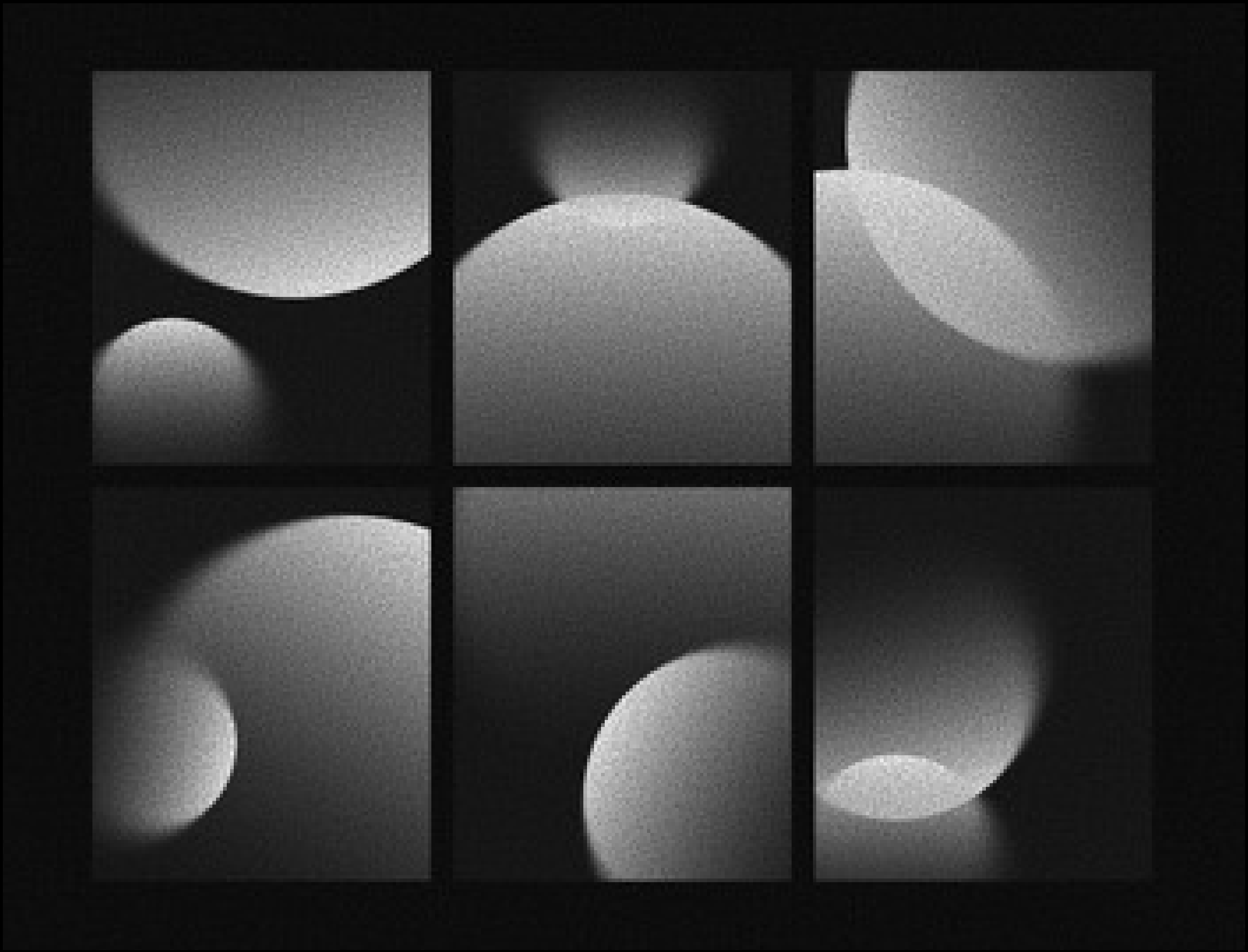
scanner

Our Test of AI in the SOC Proves Humans and AI Are Better Together



Cliff Crosland
CEO, Co-founder

Recently we conducted a deep dive of experiments into integrating Artificial Intelligence and Large Language Models (LLMs) into the Security Operations Center (SOC). Our goal was to see if AI could genuinely streamline security operations without inadvertently sidelining the invaluable human element.

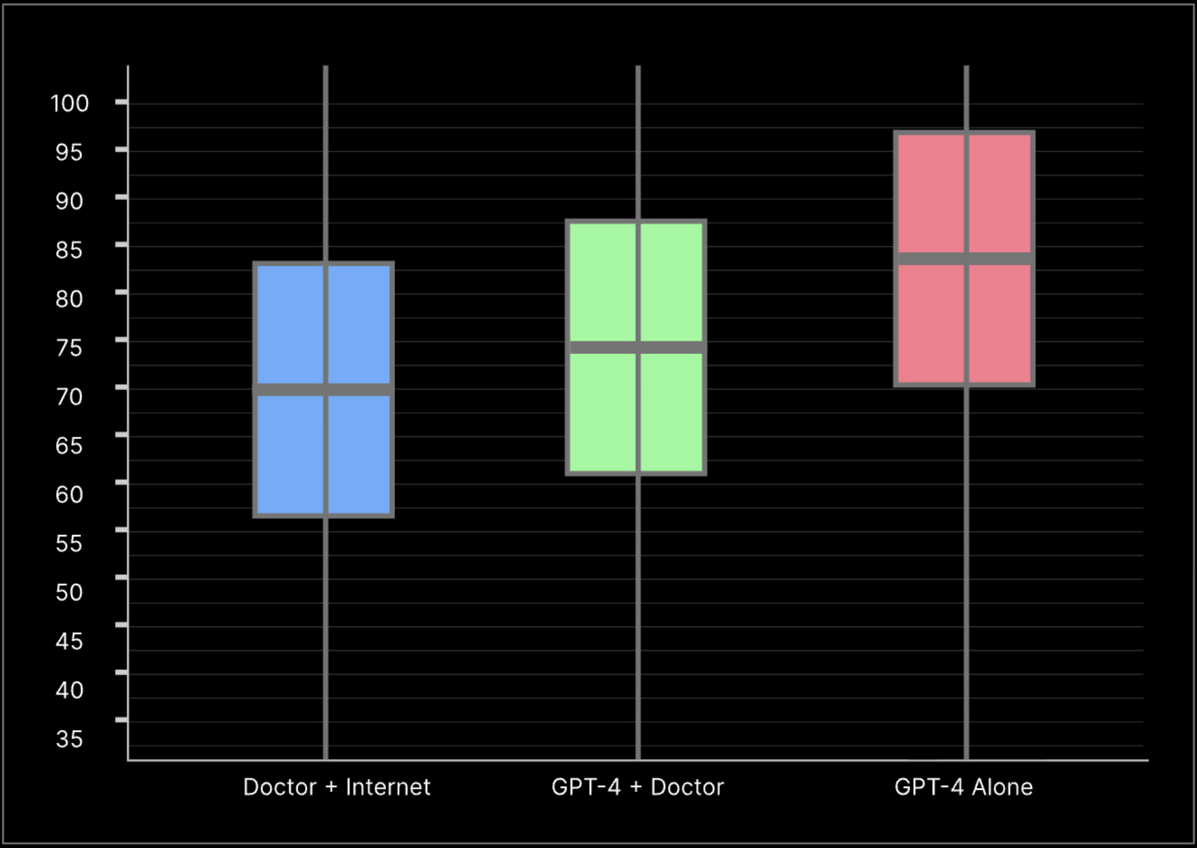
The rapid evolution of LLMs has sparked considerable discussion across industries, and cybersecurity is no exception. We believe that LLMs are poised to become a significant force multiplier for numerous SOC tasks. Imagine faster understanding of complex alerts and events data, the rapid translation of a security analyst's ideas into actionable queries, and the ability to highlight genuinely important alerts amidst a deluge of noise.

These capabilities could fundamentally change how SOC teams operate, making them more efficient and effective. However, and this is a crucial point we cannot emphasize enough, **any important task an LLM works on absolutely needs human review**. This fundamental belief guided our entire exploration.

Lessons from Beyond the Perimeter: What Other Industries Taught Us

Before diving headfirst into our own experiments, we took a moment to look at how other fields are leveraging LLMs. We believe there's a lot to learn from their experiences.

Our first stop was medicine. It's becoming increasingly clear that LLMs are developing a remarkable proficiency in diagnosis. We looked at a randomized controlled trial involving 50 doctors who were asked to diagnose clinical vignettes. The results were eye-opening: doctors working with Google assistance performed well. Doctors paired with GPT-4 performed slightly better. But astonishingly, **GPT-4 working entirely alone achieved the highest scores hitting 90-100% in medical diagnosis.** This success in diagnosis suggested that LLMs could potentially be game-changers in areas requiring complex pattern recognition and information synthesis.



However, a stark reality quickly set in: LLMs, despite their prowess in medicine, are currently **better at medical diagnosis than they are at cybersecurity diagnosis.** Why the discrepancy?

The staggering difference in available training data. While there are approximately 38 million medical papers available, cybersecurity research papers number only around 16,000. This vast difference in data volume means LLMs have a much richer, deeper well of knowledge to draw from in the medical domain. Furthermore, human biology evolves relatively slowly, providing a stable knowledge base. In contrast, IT environments and threat landscapes change at an incredibly rapid pace, making it challenging for LLMs to maintain up-to-date and nuanced understanding of evolving cyber threats.

Next, we turned our attention to the **legal profession**. Lawyers, dealing with immense volumes of complex, nuanced information, have also begun to integrate AI. The key lesson here was: **don't trust the LLM fully**. Instead, lawyers use a review iteration loop to create a refined artifact together with the AI. For example, AI can summarize outlines of documents written in complex legal language. Lawyers can use natural language shorthand, which the AI then translates into formal legal language. The AI can also tap into a vast repository of legal literature and frameworks to provide broader context. This iterative process, where humans guide and refine the AI's output, was a powerful model.

Finally, we examined **software engineering**. Software developers frequently deal with intricate systems and large codebases, where context and state are paramount. The team at Cognition, makers of the software development agent Devin, shared some interesting insights. Summarizing key moments and decisions significantly benefits future tasks. It works surprisingly well to keep context and state within a codebase, allowing for better future navigation and understanding. Tools and practices in software development suggest that creating summaries at the end of each task can provide valuable context for subsequent work. This is particularly important because the "context windows" of LLMs are not infinite, requiring thoughtful work to manage and retain relevant information.

Our First Foray: The "LLM Alone" Experiment in the SOC

Inspired (and perhaps a little swayed) by the impressive "LLM alone" results in medical diagnosis, we decided to try a similar approach in the SOC for investigating security alerts. We wanted to see if an LLM could effectively investigate security alerts without human intervention. We equipped our LLM agent, built using the Claude Code SDK, with a set of tools: our own Scanner.dev for searching alerts and logs, and VirusTotal for threat intelligence related to Indicators of Compromise (IOCs). The agent was configured to define these tools and execute queries via an SDK, then print out messages in a response stream. We then planned to review the resulting investigations for accuracy.

```python
from claude_code_sdk import query, ClaudeCodeOptions, Message

MCP_SERVERS = {
    "mcpServers": {
        "scanner-dot-dev": {
            "command": "node",
            "args": [os.getenv("SCANNER_MCP_PATH") + "/server/index.js"],
            "env": {
                "SCANNER_API_KEY": os.getenv("SCANNER_API_KEY"),
                "SCANNER_API_BASE_URL": os.getenv("SCANNER_API_BASE_URL"),
            },
        },
        "virustotal": {
            "command": "npx",
            "args": ["@burtthecoder/mcp-virustotal"],
            "env": {
                "VIRUSTOTAL_API_KEY": os.getenv("VIRUSTOTAL_API_KEY"),
            },
        },
    }
}



async def run(system_prompt: str, prompt: str) -> list[Message]:
    options = ClaudeCodeOptions(
        mcp_servers=MCP_SERVERS,
        system_prompt=system_prompt,
    )
    messages = []
    async for message in query(prompt=prompt, options=options):
        print(message)
        messages.append(message)
    return messages
```

Experiment 1: The False Positive Test

Our first experiment focused on false positives. We tasked the LLM with investigating 348 detection alerts, every single one of which was **already known to be a false positive.** A typical example was an alert named "Security Group Opened to Internet" with medium severity, describing security groups opened to 0.0.0.0/0 for HTTPS (port 443). The evidence pointed to AuthorizeSecurityGroupIngress events by scnr-deployer roles, a legitimate operation for opening Application Load Balancer (ALB) access to the internet, but one that could expose load balancers to potential attacks if misconfigured.

How did the LLM perform? It classified approximately 71% of these alerts correctly as "Benign". While not amazing, it wasn't horrendous either, correctly identifying 247 out of 348. However, it classified 50 as "Suspicious" and 51 as "Malicious," which were incorrect. The disheartening part was that these were **all easy false positives.** While this might save some initial work, the inability to trust it fully, especially with 29% incorrect classifications, was a significant drawback. It was clear that relying on an LLM alone for even seemingly straightforward tasks presented challenges.

Classified	Count	Percentage	Correct?
Benign	247	~71%	Y
Suspicious	50	~14%	N
Malicious	51	~15%	N

Experiment 2: The False Negative Test – A Scary Miss

Our second experiment was designed to test the LLM's ability to detect a genuine threat. We created a synthetic "Insider Threat" attack scenario involving three AWS IAM users attacking an AWS account. The attack comprised multiple layers of evidence: failed privilege escalation attempts as IAM policy modifications were denied, persistence established through modified Lambda functions and created event rules, and successful data exfiltration of Gigabytes of financial data downloaded from S3 buckets. This was a clear, multifaceted malicious attack.

The LLM's resulting report was, frankly, scary. It identified the activity as "Multiple failed AWS IAM operations" **but then classified it as "Benign"**. Its key finding was that it considered the activity "legitimate audit/compliance activity based on historical patterns showing regular CloudTrail access by these users". This was fundamentally wrong. The LLM completely **missed the actual threat,** resulting in a dangerous **false negative**. This outcome reinforced our conclusion that LLMs, when left entirely alone, are not yet worthy of full trust in critical security operations.

A Research Solution: SOC + Markdown = Socdown

Given the shortcomings of the "LLM alone" approach, we shifted our focus. We realized that the goal isn't to replace humans, but to augment them. We needed a way for LLMs to act as a **force multiplier**, not a replacement. This led us to develop a research project we call "**Socdown**," which stands for SOC + Markdown. This research project from the Scanner team is available on GitHub. Note: while this is not production ready (yet), you are welcome to experiment with it to experience its capabilities.

Our philosophy behind Socdown is simple yet profound: **"What if every security investigation became as reviewable, searchable, and improvable as code?"** In essence, each investigation is treated as a Markdown file stored in a Git repository. This approach allows us to apply the critical lessons we learned from other industries:

- Medicine: Leveraging LLMs for their diagnostic capabilities to get initial insights or identify patterns.
- Law: Working within a review iteration loop with the LLM to generate a comprehensive, human-reviewed artifact.
- Software Engineering: Keeping context and state consistent within a codebase, ensuring that past investigations inform future ones.

Socdown is designed to be simple and easily replicable. We implemented a custom Claude slash-command, /investigate_alert <id>, to kick off investigations. Furthermore, it integrates with various Security Operations tools like Elasticsearch, Splunk, Scanner.dev, VirusTotal, Panther, Slack, and GitHub as Model Context Protocol (MCP) servers. Simple Python utility scripts handle tasks like generating codenames for investigations, compressing older investigation markdown files, and automatically generating summaries of key moments and decisions from past investigations.

Socdown uses a CLAUDE.md file for instructions and example investigation templates. In our case, we gave it simple instructions such as follow a methodology like the “High-Probability Hypothesis Framework” and use a specific template to generate the markdown report.

Socdown in Action: Re-testing Our Experiments

With Socdown in place, we decided to re-run our initial experiments, now with the LLM acting as a "sidekick" within our defined iterative workflow.

Do-over of Experiment 1: False Positive Test

When we re-tested the 348 known false positives using Socdown, the results were markedly better. The LLM now correctly classified approximately **78% of the alerts as "Benign"**, an improvement from the initial 71%. More importantly, **zero alerts were classified as "Malicious."** The LLM demonstrated much more caution in its classification, and the **context of past investigations proved incredibly helpful in guiding its analysis**. This showed that by providing a structured framework and leveraging historical data, the LLM could be significantly more reliable in correctly identifying benign activity.

Classified	Count	Percentage	Correct?
Benign	272	~78%	Y
Suspicious	76	~22%	N
Malicious	0	~0%	Y

Do-over of Experiment 2: False Negative Test – A Triumphant Detection!

The true litmus test was the "Insider Threat" scenario using Socdown .

Initially, the LLM's investigation stopped short, identifying data access to sensitive S3 buckets but failing to fully explore it. It was at this point that the human element became critical. We encouraged the LLM to delve deeper into the S3 reads to determine if malicious data exfiltration was occurring with this simple prompt:

"Can you dig into the sensitive S3 bucket access further? Are they doing a data exfiltration attack?"

Notice that we are not giving it specifics on how to query and investigate that data, only guidance on what to do. The LLM then translated this into multiple queries. This led the LLM to correctly classify the incident as **"SUSPICIOUS" with "VERY HIGH" confidence** and accurately document the data exfiltration taking place.

We then asked it to explore what else the users were doing at the same time, specifically any activity that was malicious.

"Can you explore what else these users were doing at around the same time? any other malicious activity? what else?"

The LLM then uncovered persistence backdoor and Command and Control (C2) mechanisms the attackers had established by modifying Lambda functions.

The executive summary generated by the LLM provided key findings that precisely identified the attack. It even provided critical remediation action recommendations such as deleting the specific Lambda function, blocking the C2 domain, and more.

Finally, the first draft of the timeline the LLM provided was woefully incomplete.

It was missing much of the detail required. Once again we had to ask the LLM to expand the timeline in the report to add full details.

"The timeline is very much incomplete. Can you fill it in with all of the relevant events related to the attack?"

It then generated a much more detailed and accurate timeline.

In the end, after much human prompting, this was a remarkable transformation from completely missing the threat to accurately identifying and detailing a complex attack.

This demonstrates that human judgment, expertise, and creative ideas are essential. However, when paired with the LLM, the investigation accelerates dramatically.

AI was most definitely a force multiplier here, helping us speed up the investigation dramatically. We did need to guide it, albeit only in natural language with just a few sentences (as shown in the prompts above). The LLM crafted and executed over twenty queries in a fairly technical query language, interpreted and summarized the results for human consumption, and provided remediation recommendations. However, it failed at first. The lesson learned is that In security operations, the LLM needs to be managed and mentored!

This dramatic improvement highlights how the LLM, used within the Socdown framework, functioned as a true **force multiplier,** not a replacement.

- **Summarization**: It rapidly created clear summaries of alerts, event timelines, and the tools used during the investigation.
- **Translation**: It translated human analyst ideas, expressed in natural language shorthand, into precise queries for security tools.
- **Broader Context:** It utilized MCP tools to fetch relevant security log data and even grep prior investigation files to gain historical context.
- **Review Iterations:** While the first version generated by the LLM was in the right direction, it still required human expertise to push it along and refine it into a comprehensive artifact.

Conclusion: Humans Remain Essential, AI is the Force Multiplier

Our journey through these experiments has reinforced a fundamental truth: **humans are still needed!** Large Language Models are not here to replace the skilled analysts in your SOC. Instead, they are powerful tools that, when integrated thoughtfully and iteratively, can significantly enhance human capabilities. By adopting principles from other data-intensive professions like medicine, law, and software engineering, we can create frameworks like Socdown that allow AI to excel as an intelligent sidekick. This approach makes security investigations much faster, more reviewable, searchable, and continuously improvable - ultimately leading to a more robust and responsive security posture. The future of the SOC isn't about AI replacing us; it's about AI empowering us.