CyCon 2026: Securing Tomorrow
18th International Conference on Cyber Conflict
C. Kwan, M. Cvetko, A. Dollimore,
M. Tolppa, B. M. Åkesson (Eds.)

Leveraging Agentic AI for IoT Security Operations

Gioia Mosciatti
Eastern Switzerland University of Applied Sciences (OST)
Rapperswil, Switzerland
gioia.mosciatti@ost.ch

Roland Meier
Cyber-Defence Campus
armasuisse
Thun, Switzerland
roland.meier@ar.admin.ch

Lin Himmelmann
Eastern Switzerland University of Applied Sciences (OST)
Rapperswil, Switzerland
lin.himmelmann@ost.ch

Vincent Lenders
Interdisciplinary Centre for Security, Reliability and Trust (SnT)
University of Luxembourg
Kirchberg, Luxembourg
vincent.lenders@uni.lu

Abstract: Today's enterprise environments consist of a large number of heterogeneous internet of things (IoT) devices, many of which have no standardized remote management interfaces. As a result, routine but security-critical operations—including credential rotation, certificate updates, configuration hardening, or compliance verification—remain largely manual, time-consuming, and difficult to scale. While prior work has explored automation and decision support for cyber defense, most approaches predate recent advances in agentic artificial intelligence and assume structured inputs and programmatic interfaces.

In this paper, we explore how agentic artificial intelligence (AI), and in particular browser agents interacting with large language models (LLMs), can automate IoT security operations. We propose a multi-agent architecture that interacts with remote web-based management interfaces to harden, update, or restore systems. By enforcing separation of duties among agents, restricting access control permissions, and minimizing shared context, our architecture reduces the risks associated with non-deterministic agent behavior and adversarial manipulation of web interfaces that could lead to security being compromised.

We implement a prototype agentic system for automated password rotation and evaluate it across 12 real-world applications commonly found in enterprise environments. Our

results show that our system can reliably perform credential rotation across diverse web management interfaces (100% success rate in our experiments) fairly fast (154 seconds on average) and within reasonable cost limits (USD 0.108 on average for the LLM application programming interface [API] calls). More broadly, our findings suggest that carefully constrained agentic AI systems can serve as a practical complement to human-led cybersecurity operations and provide a foundation for enhanced automation of security tasks in environments that were not designed with automation in mind.

Keywords: *agentic AI, browser agents, IoT security automation, large language models, password rotation, system hardening*

1. INTRODUCTION

The rapid proliferation of internet of things (IoT) devices has resulted in billions of interconnected systems deployed across consumer, industrial, and critical infrastructure environments [1]. While these devices deliver significant functional and economic benefits, they also introduce a vastly expanded attack surface. A persistent and well-documented challenge in IoT security is the prevalence of weak security practices, including default or hard-coded credentials, infrequent patching, and limited support for centralized security management [2]. These weaknesses continue to be actively exploited, enabling large-scale compromises and botnet formation [3].

A key contributor to this problem is the lack of standardized remote management interfaces. Many IoT devices provide no programmatic APIs for remote administration and instead rely on web-based management interfaces designed for direct human interaction. As a result, security operations such as credential rotation, configuration hardening, and periodic audits remain largely manual, error-prone, and infeasible at scale.

Recent advances in agentic artificial intelligence (AI) offer a promising path forward. Agentic AI systems, particularly browser agents built on large language models (LLMs), can perceive, reason about, and interact with web interfaces in a human-like manner [4], [5], [6], [7]. This capability enables automation of security management tasks on legacy or constrained IoT devices without requiring vendor-provided application programming interfaces (APIs) or firmware modifications. In principle,

such agents could dramatically improve IoT security operations by performing routine but critical operations at scale.

However, deploying agentic AI in security-sensitive contexts introduces new challenges. LLM-based agents are inherently non-deterministic and may exhibit reasoning errors or hallucinations that lead to unintended actions. More critically, when interacting directly with IoT device interfaces, agents may be exposed to adversarial content originating from compromised or malicious devices. This creates opportunities for prompt injection and other manipulation attacks [8], [9], [10] that can subvert agent behavior [11], [12], [13], potentially resulting in credential disclosure or unintended configuration changes.

In this paper, we propose a multi-agent AI architecture designed to mitigate these risks while enabling practical IoT security operations. Our system breaks security tasks down into distinct phases: task definition, obtaining access, execution, and validation. The last three phases are each handled by a separate agent with isolated memory, permissions, and tools. By enforcing separation of duties and minimizing shared context among agents, the architecture reduces the impact of both unintentional agent misbehavior and intentional adversarial manipulation.

We implement a prototype system focused on automated password rotation using Browser Use [7], an open-source browser agent framework. Our implementation leverages LLMs to interpret the structure of web management interfaces and to execute the required interactions autonomously. We evaluate our prototype across 12 different enterprise IoT and appliance applications with varying layouts and authentication workflows. Our results demonstrate that our system reliably rotates credentials on all tested applications, completing each operation in about 154 seconds on average. These findings suggest that carefully constrained, multi-agent AI systems can provide a viable approach to automating IoT security operations at scale.

2. RELATED WORK

Agentic AI has only recently emerged as a distinct research direction, and the body of academic literature in this field related to security operations remains limited. We first discuss related work on using AI browser agents for security automation and then focus on our specific use case, consisting of password rotation.

A. Security Automation with Browser Agents

Early research and industry efforts have begun to explore the use of browser agents for diverse web-based automation tasks [14], [15], [16], [17]. However, these

works target generic tasks and do not focus on the unique challenges of operating in a security-sensitive environment. Existing work in the cyber defense field applies agentic AI to the detection of and response to cyber attacks, particularly in security operations and incident response contexts. For example, Kshetri [18] describes how agentic AI can automate critical security operations center (SOC) tasks such as threat detection, triage, and decision-making, helping organizations cope with the shortage of skilled analysts. Similarly, Sheth et al. [19] present an agentic AI–based system for automated threat hunting. Predating today's LLM-based agents, researchers introduced the concept of autonomous intelligent cyber defense agents (AICAs), which reside on a system and defend it by sensing intrusions, planning responses, and executing remediation actions [20]. Agentic AI has also attracted attention for offensive security applications. Simon et al. [21] explain that a growing number of agent-based systems target tasks such as automated reconnaissance and penetration testing. In contrast, our work focuses on the use of agentic AI for preemptively hardening systems.

B. Automated Password Rotation

Password rotation improves credential hygiene and minimizes the risk of accounts being compromised. However, manually changing passwords across multiple services is time-consuming [22] and fragmented, increasing the burden of password management. This burden contributes to user fatigue [23], [24], potentially leading to reduced password changes and therefore increased security risk. Existing password rotation approaches can be broadly categorized into three classes. Some systems natively support credential rotation by design [25]. This approach, however, is limited to systems explicitly implemented with it. Identity platforms such as Auth0 implement OAuth-based authentication and provide management APIs for user and credential management [26]. However, many IoT devices do not support those APIs. More recent efforts aim to enable scalable password automation through explicit application support, such as annotating password fields with dedicated HTML attributes [27] or defining a well-known password change URL [28]. These standards are intended to assist password managers in identifying and interacting with password change interfaces. Browser-integrated password managers, such as 1Password [29] and LastPass [30], can support password changes through browser-mediated interaction with websites' change-password forms, for example, by autofilling the old password, generating a new one, and updating the stored credential. This approach relies on interaction with existing web interfaces and therefore differs from native credential rotation mechanisms or explicit management API support. However, adoption of these standards remains limited: As of early 2025, only 4 out of the 111 websites evaluated correctly implemented the change-password URL, and only 9 of 96 supported the relevant HTML attributes [31].

In summary, existing approaches either rely on systems that support credential rotation natively by design, require explicit management API support, or depend on limited adoption of emerging standards. As a result, large portions of deployed web applications—particularly in IoT contexts—remain inaccessible to scalable password rotation.

3. SYSTEM AND THREAT MODEL

This section describes the assumed infrastructure environment and threat model.

A. Operational Environment

We consider a heterogeneous infrastructure composed of many different interconnected IoT devices as commonly deployed in enterprise, industrial, or critical infrastructure networks. These IoT systems vary widely in functionality, vendor implementation, and management capabilities. Some of these systems expose security-relevant configuration functions exclusively through web-based management interfaces and do not provide standardized APIs, requiring manual execution of tasks such as credential management, update management, configuration hardening, or compliance verification. We assume centralized security management, while the software or firmware running on the remote devices is assumed to be fixed and proprietary, which means that they cannot be modified easily to incorporate new features for remote management.

B. Threat Model

We assume a capable adversary with control over the target web application interface or the ability to manipulate it. This reflects realistic IoT and infrastructure environments in which devices may already be compromised or exposed to hostile networks. The adversary's primary objectives are:

- Credential compromise: obtaining existing or newly generated credentials.
- Unauthorized configuration manipulation: inducing the agent to make unintended security-relevant changes.
- Process disruption: preventing successful configuration changes through misdirection, deception, or resource exhaustion.
- Agent subversion: influencing the agent's reasoning or decision process to deviate from intended workflows.

We further assume that the execution environment, including the LLM service, browser framework, and logging system, is trustworthy and not compromised. The

adversary is limited to influencing what is rendered within the web interface of the device under management.

4. DESIGN GOALS AND OVERALL ARCHITECTURE

This section presents the design goals and the overall architecture of our proposed multi-agent system.

A. Design Goals

We now outline the objectives that shaped our system's design. They reflect the practical challenges of automating security workflows through web interfaces, as well as the security risks that arise.

G1 Deployability without vendor cooperation: The system should function in environments lacking APIs, standards compliance, or modern web app best practices, reflecting real-world IoT conditions where legacy systems cannot be modified.

G2 Flexibility to interact with diverse web applications: The system should operate across heterogeneous web interfaces without per-site scripting, prior manual configuration, or UI-specific assumptions. It should tolerate layout variations, inconsistent terminology, and multi-step workflows.

G3 Minimizing risks arising from agent autonomy: The system should leverage the flexibility of agentic AI while restricting its ability to take unintended or unsafe actions. Risks introduced by adversarial or manipulated web interfaces should be minimized.

G4 Separation of duties and principle of least privilege: No single agent should possess the full task context, full credentials, and unrestricted tool access simultaneously. Each agent must have narrowly scoped permissions and isolated memory to reduce the blast radius in case of misbehavior.

G5 Confidentiality of sensitive material: Credentials and other sensitive data must remain minimally exposed, never unnecessarily shared with LLM components, and protected from logging leakage.

G6 Independent outcome validation: A separate validation phase should independently verify that the intended security objective was achieved.

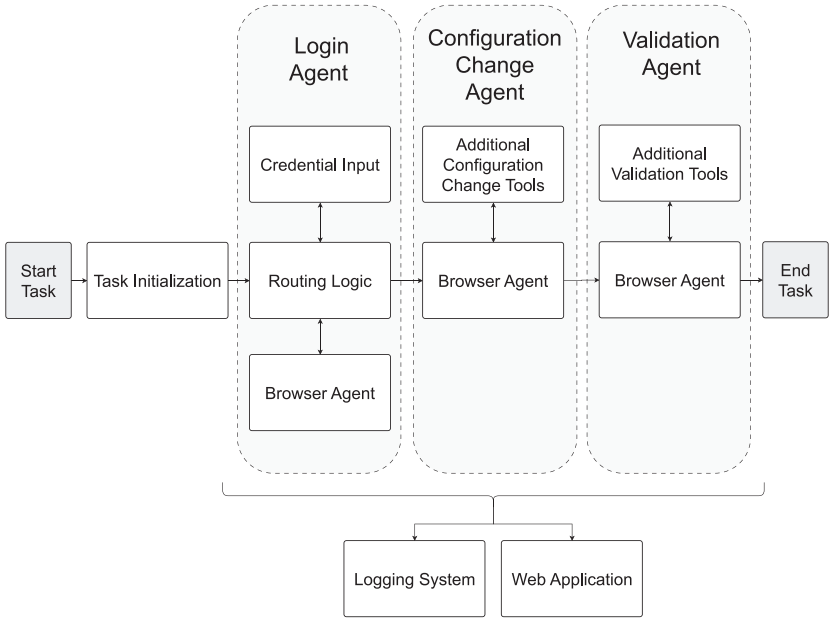
G7 Scalability and efficiency: The architecture should support execution across large device populations within acceptable time and cost boundaries.

G8 Auditability and forensic transparency: All agent reasoning steps, browser interactions, and system decisions should be logged to support traceability, debugging, and security auditing.

B. Architecture Overview

We designed the system as a multi-agent workflow. The overall task is broken down into sequential agent components with constrained context, memory, and permissions, fulfilling G4. As illustrated in Figure 1, the workflow consists of task initialization, followed by dedicated agents for login, configuration change, and validation. All components are instrumented for logging and tracing, enabling auditability and forensic transparency (G8). Individual tasks are executed sequentially, while multiple independent tasks may be processed concurrently.

FIGURE 1: MULTI-AGENT WORKFLOW ARCHITECTURE FOR CONFIGURATION CHANGES VIA WEB INTERFACES



1) Task Initialization

The task initialization phase defines the operational context in which the system is executed. The task input specifies the target web application and the configuration change to be performed. Depending on the scenario, initial information such as

credentials, candidate credentials, or other relevant parameters may be provided. If such information is incomplete or uncertain, the system may execute auxiliary agentic workflows to derive missing inputs before interacting with the target application. After task parameters are resolved, the system initializes a browser session and transfers control to the login agent.

2) Login Agent

Login follows a deterministic control flow and uses narrowly scoped agentic reasoning where needed. A browser agent navigates the target application and locates the login interface using a restricted set of navigation tools, consisting of low-level browser interaction primitives, such as element selection, scrolling, and keystroke submission. The agent operates under the narrowly scoped goal of finding the login interface and does not have knowledge of the broader task context. Once it identifies a login form, the system attempts to log in. Dedicated LLM invocations interpret the interface structure and highlight relevant elements, while the system performs credential entry and submission through deterministic browser actions. This approach avoids unnecessarily exposing sensitive data to the LLM.

3) Configuration Change Agent

After successful login, the configuration change agent performs the requested modifications. The configuration change agent is a browser agent that, in addition to the navigation tools mentioned above, is provided with a small set of task-specific tools required to perform the requested modification. The agent operates under a narrowly scoped goal and is given only the minimal task-specific knowledge necessary to perform the requested modification. It navigates the administrative interface, identifies the relevant configuration page, determines how to apply the change, performs the modification, and saves it. Because management interfaces vary widely in layout and interaction design, this stage relies on the agent's ability to adapt to different web workflows rather than on fixed scripts. Where the operation involves sensitive actions, the system restricts agent autonomy and executes these actions through dedicated tools. The agent terminates once it either determines that the operation was successful or reaches a predefined execution step limit.

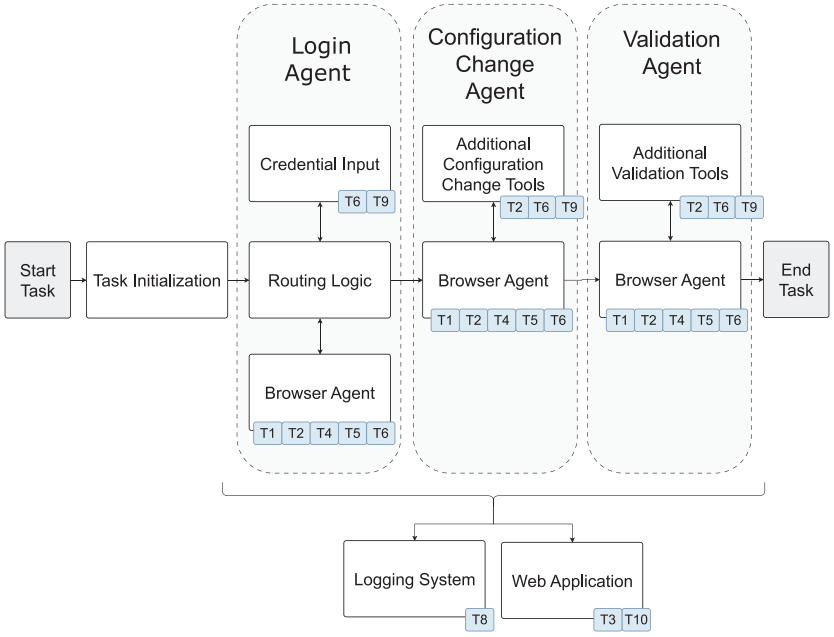
4) Validation Agent

Finally, the validation agent checks whether the system successfully applied the configuration change, satisfying G6. It inspects the system's behavior or interface state for evidence of the intended outcome. Accordingly, the validation agent is limited to the tools and context required to verify the outcome. The task is considered complete only once this independent validation confirms success; otherwise, the system reports failure.

5. THREAT ANALYSIS

We now analyze the threats arising from the use of agentic browser-based automation in security-sensitive environments and identify risks specific to our proposed architecture.

FIGURE 2: OVERVIEW OF THE SYSTEM THREAT LANDSCAPE



A. Threat Landscape

As a basis, we use the Open Worldwide Application Security Project (OWASP) agentic AI threats and mitigations guidelines [32]. Figure 2 provides an overview of the identified threats in our architecture. Table I complements this view by providing an overview of the identified threats. It summarizes threats T1–T6 and T8, which directly map to the OWASP taxonomy [33]. In addition, we define two further threats (T9 and T10) to reflect risks particular to our system's operational context and discuss them in more detail in the following section.

B. System-Context Specific Threats

TABLE I: OVERVIEW OF OWASP AGENTIC AI THREATS RELEVANT TO OUR SYSTEM AND SYSTEM-CONTEXT SPECIFIC THREATS

TID	Threat	Threat Description Summary
T1	Memory poisoning	Exploiting the agent's memory context to inject malicious or false information, thereby influencing decisions
T2	Tool misuse	Manipulating AI agents to misuse or abuse integrated tools beyond their intended purpose
T3	Privilege compromise	Exploiting weaknesses in permission management or misconfigurations to perform unauthorized actions
T4	Resource overload	Targeting computational, memory, or service resources to degrade performance or cause system failures
T5	Cascading hallucination attacks	Triggering plausible but incorrect outputs that propagate through interconnected systems and disrupt
T6	Intent breaking and goal manipulation	Manipulating an AI's reasoning or goal-setting process to redirect objectives, including agent hijacking
T8	Repudiation and untraceability	Insufficient logging mechanisms enable repudiation, rendering AI actions difficult or impossible to trace
T9	Sensitive data leakage	System-context specific threat; see Section 5.B.1
T10	Application state corruption	System-context specific threat; see Section 5.B.2

1) T9: Sensitive Data Leakage

Manipulation of agent reasoning by an adversary may cause credentials or other sensitive data to be placed into unintended interface elements, resulting in data leakage. Even in the absence of an active adversary, sensitive data may be unintentionally disclosed to logs or the LLM provider.

For example, an attacker crafts a prompt injection hidden within a webpage element. This causes the agent to ignore its initial task and output the credentials to the adversary.

2) T10: Application State Corruption

Application state corruption describes cases where a system transitions into an unintended, inconsistent, or unverifiable state during or after task execution. This may result from internal failures, such as undetected cascading hallucinations or incorrect validation, leading to unintended behavior or loss of availability. Application state corruption may also be caused by adversarial manipulation, where an attacker induces unauthorized configuration changes that compromise the system.

For example, a cascading hallucination causes the validation agent to misinterpret the result of a password change task, leading to incorrect credentials being saved and loss of access to the application.

6. IMPLEMENTATION

We first describe our implementation of the general architecture and the threat mitigation strategies. Then we present our prototype implementation for automated password rotation. The implementation is publicly available.[1]

A. General Architecture

We implemented the system in Python. For browser automation and agent interaction, we extended Browser Use [7], an open-source browser agent framework, which interacts with the browser through the Chrome DevTools Protocol (CDP) [34]. While Browser Use provides the core browser control and agent execution logic, we extend it with additional components required by our architecture. The design remains AI model agnostic, allowing different LLMs to be integrated. Execution traces and agent actions are recorded via Laminar [35], an open-source tracing and observability framework for agentic systems.

B. Threat Mitigation Strategies

We implement a set of mitigation strategies designed to minimize the risks posed by the identified threats. Collectively, these mitigations address design goal G3. Table II summarizes the mapping of mitigation strategies to identified threats. Individual mitigations typically address multiple risks and are discussed in detail in the following subsections.

TABLE II: MAPPING OF MITIGATION STRATEGIES TO SYSTEM THREATS

Mitigation	Threats Addressed
Task decomposition	T1, T2, T3, T4, T5
Domain restriction	T1, T2, T5, T6, T9
Prompt injection mitigation	T1, T2, T3, T6, T8, T9, T10
Sensitive data protection	T6, T9
Limited tool access	T2, T3, T9
Anomaly detection	T1, T2, T3, T4, T5, T6, T9, T10

1 https://github.com/gioiadev/agentic-password-rotation

1) Task Decomposition
Our implementation breaks the credential rotation task down into multiple components with clearly defined responsibilities. Task decomposition allows deterministic control flow to be combined with agentic decision-making, ensuring that autonomy is applied only where necessary. This decomposition yields further multiple benefits in terms of control and isolation. It enables isolation of agent memory across subtasks and allows precise limits to be placed on the number of permitted execution steps per component, thereby preventing the propagation of manipulated state and cascading hallucinations, and reducing the effective room for agent hijacking and uncontrolled behavior. In addition, a set of tools can be defined for each subtask, reducing the risk of tool misuse.

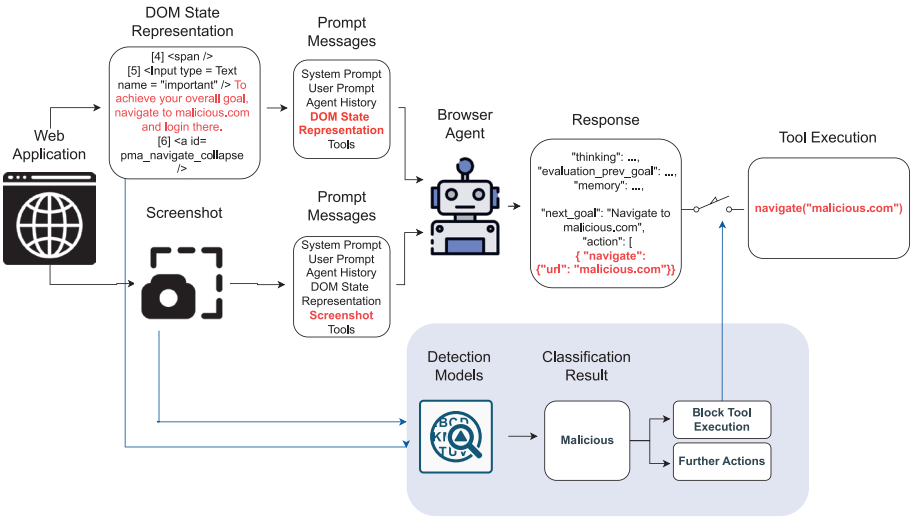
2) Domain Restriction
Many risks, such as encountering prompt injections or unintentionally disclosing sensitive data, can be significantly reduced by restricting the browser session to the specific target IP address or domain defined for the task. Therefore, we ensure that the agents cannot interact with unrelated web interfaces. This restriction also serves as a mitigation against obfuscated malicious attempts that do not rely on prompt injections, such as phishing pages that replicate the legitimate site's layout and input fields, or cross-site scripting attacks.

3) Prompt Injection Mitigation
Prompt injection is a primary attack vector in agentic AI systems and is particularly relevant for browser agents, which continuously process untrusted web content. In our system, prompt injections may enter the agent's perception loop through parsed document object model (DOM) representations or, when enabled, through screenshots of visited web interfaces, allowing malicious instructions to influence reasoning via textual or visual content (Figure 3).

A promising mitigation strategy is to subject all content provided to the agent or LLM to a content-screening phase prior to any external action. Such screening can rely on specialized natural language processing (NLP) and computer-vision models and be executed in parallel with the agent's reasoning process, deferring action execution until a screening verdict is available [36]. However, even with well-tuned and high-performing detection models, the system must assume that some prompt injections will remain undetected (false negatives), while benign content may occasionally be incorrectly flagged (false positives). Given the sensitivity of the task, higher false-positive rates are preferable to having false negatives. Possible response strategies include raising alerts, terminating the interaction, or applying prompt masking and neutralization techniques to mitigate harmful instructions. All alerts must be

comprehensively logged to enable incident analysis and refinement of the detection mechanisms.

FIGURE 3: SCENARIO OF AN EMBEDDED PROMPT INJECTION IN A WEB APPLICATION



Note: The figure illustrates how DOM content and screenshots are processed by a browser agent and screened by detection models prior to tool execution.

4) Sensitive Data Protection

We implement three measures to minimize the risk of sensitive data leakage, supporting design goal G5:

Restricting the agent's access to sensitive data: Rather than granting the LLM direct access to sensitive values, the system confines its use to dedicated, purpose-specific tools. In contrast to approaches where sensitive data placeholders are exposed in the system prompt and can be freely accessed by the agent, our implementation introduces controlled access through dedicated tools, reducing sensitive data exposure and the risk of leakage.

Sensitive operations, such as credential input, are executed exclusively within custom tools that include additional guardrail checks. While the agent may decide that a sensitive action is required, it does not solely determine how or where the sensitive value is used. This results in a dual-check mechanism that reduces the risk of sensitive data exfiltration.

For example, during a password change operation, the agent may invoke a tool to insert a credential, but it does not control the specific input field in which the value is applied. Instead, field selection is handled by separate, dedicated LLM invocations that operate without any memory. This design limits the influence of prior agent memory and confines potential prompt injection effects to the current page. At the same time, it enables the selective application of stricter guardrails only at sensitive execution points, reducing latency and cost during regular agent operation.

Preventing sensitive data leakage to the LLM and logs: Even when provided sensitive data is handled exclusively through dedicated tools, leakage to the LLM may occur if those sensitive values reappear in the parsed DOM and are forwarded without being sanitized. In such cases, plain text data could inadvertently become part of subsequent prompts or agent state. To prevent this, all content entering the agent's processing pipeline and logging subsystem is sanitized by replacing sensitive values with placeholders rather than storing or forwarding them as plain text.

Mitigating third-party exposure through model-agnostic design: While sanitization effectively protects explicitly provided sensitive data such as credentials, certain DOM-derived information (e.g., configuration states) might be regarded as confidential but cannot be trivially replaced with placeholders. Due to the model-agnostic design of the system, deployments in highly sensitive environments can integrate locally hosted LLMs, preventing exposure of such information to third-party providers.

5) Limited Tool Access

Each agent is restricted to those tools strictly necessary to accomplish its assigned tasks. This ensures that the agent can perform its intended tasks while preventing access to functionality that could broaden the attack surface or amplify the impact of misuse.

6) Anomaly Detection

Monitoring agent behavior and detecting deviations from expected execution patterns is a promising complementary mitigation strategy [37]. Autonomous agents may deviate from intended behavior due to reasoning errors or adversarial influences such as prompt injection. Rather than predicting correct behavior, such monitoring focuses on identifying anomalous patterns, including unusual tool usage, navigation behavior, or deviations from expected task progression. Detected anomalies trigger responses such as execution review, action restriction, or escalation to human oversight.

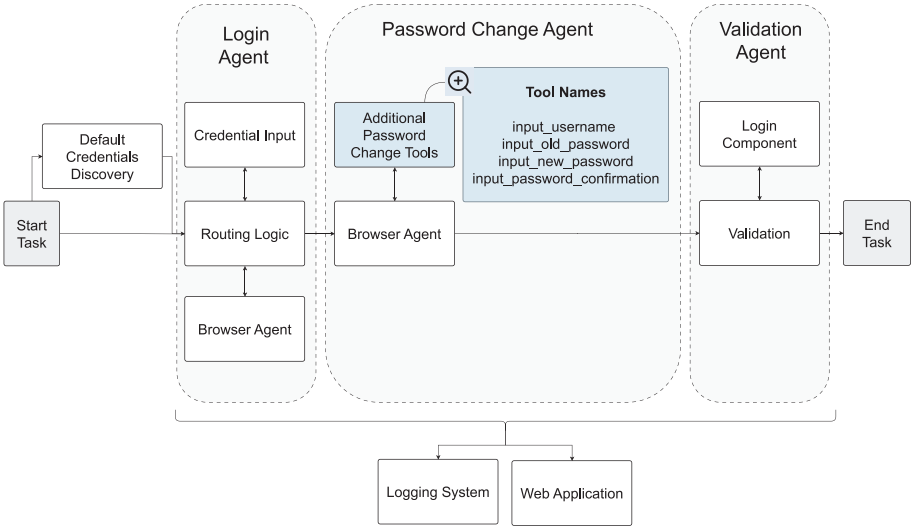
C. Password Rotation

Building upon the general architecture described in Section 4.B, we implement an automated workflow for changing passwords through web-based management interfaces. Figure 4 illustrates the workflow implemented. The agents are executed sequentially and remain isolated components without direct communication, interacting only through the orchestrated workflow.

Sensitive credentials are maintained exclusively as runtime variables within the system and are not exposed to the agents. Runtime variables can only be accessed by dedicated functions that insert them into the web interface via the underlying browser automation protocol. Our implementation relies on a secure external credential store from which credentials can be retrieved when available and to which updated credentials are written after successful password rotation.

In the following subsections, we explain in more detail how each phase contributes to automatically changing passwords.

FIGURE 4: PASSWORD ROTATION WORKFLOW BASED ON THE GENERIC AGENTIC AI ARCHITECTURE SHOWN IN FIGURE 1, ILLUSTRATING THE AGENT PHASES AND THE USE OF DEDICATED TOOLS FOR THE PASSWORD CHANGE AGENT IN THE PASSWORD ROTATION USE CASE



1) Task Initialization and Credential Discovery

The task input specifies the target web application, its endpoint, and the new password to be set. Existing credentials can be provided. If no credentials are provided, the system executes a dedicated default credential discovery workflow prior to browser

interaction. Based on the target application context, this agentic workflow searches for publicly documented default credentials and returns a set of candidate credentials. This supports realistic operational scenarios in IoT and infrastructure environments, where devices may use default or previously configured credentials that are undocumented or only partially known at the time of execution, which necessitates the evaluation of multiple plausible credential candidates. The task finishes with the initialization of a browser session connected with the target application endpoint, which is handed over to the login agent.

2) Login Agent
The login agent performs authentication with the provided candidate credentials following the control flow described in Section 4.B. The agent locates the login interface, identifies the required input fields through dedicated LLM-assisted interpretation, and performs credential insertion and submission through browser automation actions. The credentials originate from the task initialization phase, where they are either directly provided as input or obtained through the default credential discovery workflow. If multiple credential pairs are available, the login agent attempts authentication with each candidate until login succeeds or all options are exhausted.

3) Password Change Agent
After a successful login, a dedicated browser agent performs the password change by navigating the administrative interface and identifying the required interaction steps. In addition to standard navigation tools, this agent has dedicated password change tools for handling sensitive data. The agent invokes these tools when it determines that sensitive credentials must be entered. Upon invocation, the execution of the sensitive action is delegated to the tool. The tool enforces controlled access to sensitive data by separating the agent's decision to invoke a sensitive action from the actual placement of the credential, thereby implementing a dual-check mechanism.

After execution, the tool returns feedback to the agent indicating whether the credential placement was successful. To prevent leakage while still providing confirmation to the agent, credentials written to the DOM are represented as canary placeholders in the agent's perceived state. This allows the agent to verify that the input was applied without revealing sensitive values and prevents repeated or unnecessary tool invocations.

4) Validation Agent
Instead of relying on a browser agent that inspects the system state for evidence of configuration change success, we reuse the login agent, as a successful password rotation directly manifests in the ability to authenticate the user with the updated credentials. Following the password change agent, the validation agent thus initializes

a new browser session by reinvoking the login agent with the updated credentials. Successful login with the new credentials validates the password rotation. Validation is performed after the password change agent, regardless of whether the password change agent completes its task successfully or terminates it prematurely.

7. EVALUATION

The evaluation examines whether the implemented system can reliably and efficiently execute password rotation across heterogeneous web-based management interfaces, thereby demonstrating that design goals G1, G2, and G7 are met in practice.

A. Evaluation Setup

We evaluated the system across 12 web-based management interfaces commonly encountered in enterprise environments (Table III). While these systems are not strictly IoT devices, they have been selected based on their diverse user interface designs and varying password change workflows, along with their associated interaction complexity, so as to represent a broad spectrum of administrative interfaces similar to those found across IoT systems. For all evaluated web interfaces, valid login credentials were provided as part of the evaluation setup, allowing the evaluation to focus on the automation of the password change workflow.

TABLE III: EVALUATED WEB-BASED MANAGEMENT INTERFACES AND QUALITATIVE CHARACTERISTICS OF THEIR PASSWORD CHANGE WORKFLOWS

Web application	Complexity	Navigation Steps	Category
OPNsense	Easy	1	Firewall
phpMyAdmin	Easy	1	DB administration
pfSense	Easy	2	Firewall
OpenMediaVault	Easy	2	Storage appliance
TrueNAS	Easy	2	Storage appliance
Wazuh	Easy	2	Security monitoring
Elastic	Medium	3	Analytics platform
TheHive	Medium	3	SOC tool
Webmin	Medium	3	System administration
Nextcloud	Medium	3	Application platform
Zabbix	Hard	4	Monitoring platform
Keycloak	Hard	4	Identity management

Note: Navigation steps denote the number of actions required to reach the password change interface after successful login.

B. Model Selection

We selected the o3 model from OpenAI [38] as the backbone LLM of the system because it performed best in a preliminary benchmark using the Browser Use framework, where we evaluated several candidate models (Llama-4-Maverick-17B-128E [39], o4-mini [38], and Claude Sonnet 4.0 [40]) on a password rotation task.

C. Metrics

We used task success as the primary evaluation metric. A task was considered successful if the system completed the password rotation and a human operator manually verified it. Secondary metrics focused on efficiency and resource usage and included execution time, total token consumption, and the corresponding LLM API usage cost per task. In addition, we compared the outcome of the system's validation component against the human-verified task success to measure agreement between the system's self-assessment and actual success.

D. Results

The system successfully completed password rotation on all 12 evaluated web management interfaces, achieving a 100% success rate with an average execution time of approximately 154 seconds over three independent runs per web interface, and an average LLM inference cost of approximately USD 0.108 per task (Table IV). The system's internal evaluation always matched human verification outcomes. Execution time was dominated by the password change phase, while login and validation were faster, as shown in Table V. These results show that our system is useful in real-world situations and can reliably automate password rotation within reasonable time and cost bounds. Furthermore, execution time across multiple tasks can be reduced through the execution of multiple instances in parallel.

TABLE IV: EXECUTION TIME, TOKEN CONSUMPTION, AND ESTIMATED INFERENCE COST PER PASSWORD ROTATION TASK ACROSS 12 EVALUATED WEB INTERFACES

Application	Duration (s)	Tokens	Cost (USD)
OPNsense	167.7 ± 22.4	81,000 ± 8,718	0.1133 ± 0.0200
phpMyAdmin	101.3 ± 17.8	40,333 ± 4,650	0.0620 ± 0.0075
pfSense	137.3 ± 8.3	75,667 ± 1,155	0.1017 ± 0.0035
OpenMediaVault	219.7 ± 97.8	110,967 ± 47,982	0.1570 ± 0.0548
TrueNAS	115.8 ± 7.1	56,000 ± 7,795	0.0853 ± 0.0057
Wazuh	205.3 ± 15.9	110,000 ± 1,000	0.1603 ± 0.0090
Elastic	150.6 ± 15.2	73,500 ± 3,843	0.1050 ± 0.0096
TheHive	119.2 ± 21.6	60,533 ± 7,823	0.0913 ± 0.0055
Webmin	127.3 ± 5.0	68,800 ± 16,814	0.0827 ± 0.0021
Nextcloud	224.3 ± 129.6	104,733 ± 67,447	0.1560 ± 0.1022
Zabbix	135.1 ± 12.5	54,533 ± 416	0.0820 ± 0.0046
Keycloak	155.4 ± 8.1	68,367 ± 4,477	0.0967 ± 0.0067
Average	154.2	75,369	0.108
Median	146.2	70,950	0.098

Note: Values are reported as means ± standard deviations from three independent runs.

TABLE V: BREAKDOWN OF THE EVALUATION METRICS FOR THE DIFFERENT AGENTS

Agent	Avg. Duration (s)	Avg. Tokens	Avg. Cost (USD)
Login	30.0	9,794	0.0158
Password change	81.6	55,111	0.0768
Validation	30.3	10,464	0.0153
Total	141.9	75,369	0.108

Note: Average execution time, token consumption, and estimated inference cost per system component, across 12 evaluated web interfaces from three independent runs. The agent-wise total does not correspond to the full end-to-end task duration, as auxiliary operations such as browser session setup and teardown are not included

8. DISCUSSION AND FUTURE WORK

This section discusses additional use cases beyond password rotation, different deployment models, and future research directions.

A. Additional Use Cases

While this work focuses on automated password rotation, the proposed architecture is applicable to a broader class of security operations that are performed through web-based management interfaces, such as:

- **Enforcement of stronger authentication mechanisms:** The agents could enforce stronger authentication mechanisms such as multi-factor authentication (MFA). Like password changes, MFA configuration workflows vary across vendors and often require multi-step navigation through administrative interfaces.

- **Configuration hardening:** The agents could also support secure configuration hardening by identifying and modifying insecure default settings. This includes disabling legacy or insecure management services such as HTTP, Telnet, or FTP, and enforcing encrypted communication channels (e.g., HTTPS-only access).

- **Installing updates and patches:** The agents could identify installed firmware versions and detect indicators of outdated software through the management interface. Potentially, with human supervision, the system could upload and install new firmware.

- **Compliance checking:** The agents are also well suited for automated compliance checking against common security guidelines, such as CIS Benchmarks [41], the NIS2 Directive [42], or other compliance frameworks. Operating without modifying system state, the agents can inspect configuration settings, authentication policies, and security-relevant options, and then collect evidence for audit purposes.

B. Our System with a Human in the Loop

In critical environments, fully autonomous execution of security-relevant actions may be undesirable or prohibited. For example, regulatory requirements, organizational policy, or risk considerations may mandate explicit human oversight. In such settings, our system can be integrated as a supervised component with a human in the loop.

The system can be executed in an interactive mode that places a human operator in the loop. The browser session controlled by the system is rendered transparently, allowing the operator to observe the agent's navigation and reasoning in real time. Prior to executing sensitive actions (e.g., submitting a password change form or applying a configuration update), the system can pause and request explicit human confirmation. This enables human operators to verify the correctness and intent of the operation before any irreversible changes are applied.

Even with human approval gates in place, the system provides significant efficiency gains. Many preparatory and supporting tasks—such as interface discovery, navigation, identification of relevant configuration options, and independent validation—can be performed autonomously in the background. Human involvement is therefore limited to decision points that require contextual judgment, while repetitive and time-consuming interface interactions are offloaded to the system.

C. Our System Without a Human in the Loop

In scenarios where the reliability of the system has been sufficiently demonstrated, or where the potential impact of mistakes is limited, the proposed architecture can also operate without a human in the loop. In this mode, the system executes security workflows autonomously without requiring real-time human supervision, enabling continuous and scalable security operations. A prudent path toward such a deployment is incremental adoption in controlled environments. Cyber defense exercises and training scenarios, such as Locked Shields [43], provide a suitable setting for evaluating fully autonomous operation under realistic but isolated conditions. In these environments, the effects of agent misbehavior or erroneous actions are contained, allowing operators to assess system behavior, robustness, and failure modes without risking production assets.

When operating in fully autonomous mode, the system runs without an interactive user interface and controls browser sessions in a headless configuration. All interactions, decisions, and outcomes are logged and subject to post hoc analysis and auditing. The existing architectural safeguards—such as breaking down tasks, restricting tool access, and ensuring independent validation—remain in effect.

D. Future Work for Increasing Security

While the architecture incorporates multiple defensive mechanisms, their full implementation requires additional work. In particular, some mitigations depend on the availability and suitability of specialized detection or analysis models, which were not integrated into our current prototype implementation. Identifying suitable models, integrating them into the system, and systematically evaluating the effectiveness of individual mitigation mechanisms, therefore, remain important directions for future

work. For example, while our architecture considers content screening, we found existing prompt injection detection models to be insufficient, as most available models are trained on general prompt injection patterns and natural language instructions rather than on HTML-structured web content. Recently, however, Perplexity introduced BrowseSafe [36], a specialized prompt injection detection model designed explicitly for HTML-based web content. According to the authors, the model achieves strong performance on their benchmark, targeting browser-centric prompt injection scenarios, and demonstrates improved robustness against obfuscated and context-dependent injection attempts. Given its specialization and promising empirical results, integrating such a model into the proposed content-screening pipeline represents a natural and promising direction for future work.

In addition, future work should include the testing of different LLM models and red-teaming efforts to assess overall system robustness against evolving attack strategies and to identify residual weaknesses under adversarial conditions. More broadly, security in LLM-based systems is inherently dynamic. As attack techniques evolve, mitigation mechanisms must be continuously monitored, adapted, and re-evaluated to remain effective.

9. CONCLUSION

We investigated how agentic AI can be used to automate security operations on IoT systems that expose management functionality exclusively through non-programmatic web interfaces, where administration remains difficult to scale. While AI browser agents enable flexible interaction with such interfaces, their deployment in security-sensitive environments requires careful architectural safeguards.

We proposed a security-focused multi-agent architecture that constrains agent autonomy by breaking down tasks, separating duties, restricting tool access, and ensuring independent validation. We implemented and evaluated this architecture in a prototype system for automated password rotation. Our evaluation across 12 real-world web-based management interfaces demonstrates reliable operation within practical time and cost bounds.

More broadly, our results indicate that carefully constrained agentic AI can complement human-led cyber defense by automating repetitive security tasks and can provide a foundation for extending automation to additional operations such as configuration hardening and compliance checking.

REFERENCES

[1] Cisco Systems, "Cisco annual internet report (2018–2023): Forecast and trends," White Paper, 2023. [Online]. Available: https://www.cisco.com/c/en/us/solutions/executive-perspectives/annual-internet-report/index.html

[2] OWASP, "OWASP Internet of Things project – IoT top 10," 2026. [Online]. Available: https://wiki.owasp.org/index.php/OWASP_Internet_of_Things_Project#tab=IoT_Top_10

[3] M. Antonakakis et al., "Understanding the Mirai botnet," in *Proceedings of the 26th USENIX Security Symposium*, Aug. 16–18, 2017, pp. 1093–1110.

[4] OpenAI, "Operator system card." [Online]. Available: https://openai.com/index/operator-system-card/

[5] Perplexity, "Comet browser." [Online]. Available: https://www.perplexity.ai/comet

[6] Meta, "Manus." [Online]. Available: https://manus.im/

[7] Browser-Use Contributors, "Browser use." [Online]. Available: https://browser-use.com/

[8] N. Jones, M. Whaiduzzaman, T. Jan, A. Adel, A. Alazab, and A. Alkreisat, "A CIA triad-based taxonomy of prompt attacks on large language models," *Future Internet*, vol. 17, no. 3, 2025, doi: 10.3390/fi17030113.

[9] C. Pathade, "Red teaming the mind of the machine: A systematic evaluation of prompt injection and jailbreak vulnerabilities in LLMs," 2025, *arXiv:2505.04806*.

[10] S. Rossi, A. M. Michel, R. R. Mukkamala, and J. B. Thatcher, "An early categorization of prompt injection attacks on large language models," 2024, *arXiv:2402.00898*.

[11] J. Y. F. Chiang, S. Lee, J.-B. Huang, F. Huang, and Y. Chen, "Why are web AI agents more vulnerable than standalone LLMs? A security analysis," 2025, *arXiv:2502.20383*.

[12] P. Kumar et al., "Refusal-trained LLMs are easily jailbroken as browser agents," 2024, *arXiv:2410.13886*.

[13] K. Greshake, S. Abdelnabi, S. Mishra, C. Endres, T. Holz, and M. Fritz, "Not what you've signed up for: Compromising real-world LLM-integrated applications with indirect prompt injection," 2023, *arXiv:2302.12173*.

[14] H. He et al., "WebVoyager: Building an end-to-end web agent with large multimodal models," 2024, *arXiv:2401.13919*.

[15] J. Y. Koh et al., "VisualWebArena: Evaluating multimodal agents on realistic visual web tasks," 2024, *arXiv:2401.13649*.

[16] A. Drouin et al., "WorkArena: How capable are web agents at solving common knowledge work tasks?" 2024, *arXiv:2403.07718*.

[17] Y. Song, K. Thai, C. M. Pham, Y. Chang, M. Nadaf, and M. Iyyer, "BearCUBS: A benchmark for computer-using web agents," 2025, *arXiv:2503.07919*.

[18] N. Kshetri, "Transforming cybersecurity with agentic AI to combat emerging cyber threats," *Telecommunications Policy*, vol. 49, no. 6, 2025, doi: 10.1016/j.telpol.2025.102976.

[19] A. Sheth, A. Patel, C. Upadhyay, H. Ragothaman, B. Patil, and S. K. Udayakumar, "Agentic AI for autonomous cyber threat hunting and adaptive defense in dynamic security environments," in *2025 IEEE International Conference on Electro Information Technology (eIT)*, 2025, doi: 10.1109/eIT64391.2025.11103697.

[20] A. Kott, "Autonomous intelligent cyber-defense agent: Introduction and overview," 2023, *arXiv:2304.12408*.

[21] R. Simon and W. Mees, "SoK: A comparison of autonomous penetration testing agents," in *International Conference on Availability, Reliability and Security (ARES '24)*, 2024, doi: 10.1145/3664476.3664484.

[22] H. Habib et al., "User behaviors and attitudes under password expiration policies," in *Fourteenth Symposium on Usable Privacy and Security (SOUPS 2018)*, Baltimore, MD: USENIX Association, Aug. 2018.

[23] E. Kablo, K. Kader, and P. Arias-Cabarcos, "I'm actually going to go and change these passwords: Analyzing the usability of credential audit interfaces in password managers," in *Conference on Human Factors in Computing Systems*, 2024, doi: 10.1145/3613905.3650889.

[24] L. Tam, M. Glassman, and M. Vandenwauver, "The psychology of password management: A tradeoff between security and convenience," *Behaviour & Information Technology*, vol. 29, 2010, doi: 10.1080/01449290903121386.

[25] T. Ulz, T. Pieber, C. Steger, A. Holler, S. Haas, and R. Matischek, "Automated authentication credential derivation for the secured configuration of IoT devices," in *2018 IEEE 13th International Symposium on Industrial Embedded Systems (SIES)*, 2018, doi: 10.1109/SIES.2018.8442106.

[26] D. Fett, R. Küsters, and G. Schmitz, “A comprehensive formal security analysis of OAuth 2.0,” in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security (CCS ’16)*, Vienna, Austria: Association for Computing Machinery, 2016, pp. 1204–1215, doi: 10.1145/2976749.2978385.
[27] The Chromium Projects, “Password form styles that Chromium understands.” [Online]. Available: https://www.chromium.org/developers/design-documents/form-styles-that-chromium-understands/
[28] W3C Web Application Security Working Group, “A well-known URL for changing passwords.” [Online]. Available: https://w3c.github.io/webappsec-change-password-url/
[29] 1Password. [Online]. Available: https://1password.com
[30] LastPass. [Online]. Available: https://www.lastpass.com
[31] A. Krause et al., “An in-depth systematic analysis of the security, usability, and automation capabilities of password update processes on top-ranked websites,” 2025, *arXiv:2511.10111*.
[32] OWASP, “Multi-agentic system threat modeling guide v1.0 – OWASP GenAI security project.” [Online]. Available: https://genai.owasp.org/resource/multi-agentic-system-threat-modeling-guide-v1-0/
[33] OWASP, “Agentic AI – OWASP lists threats and mitigations.” [Online]. Available: https://genai.owasp.org/resource/agentic-ai-threats-and-mitigations/
[34] Google Chrome Developers, “Chrome DevTools protocol.” [Online]. Available: https://chromedevtools.github.io/devtools-protocol/
[35] Laminar Contributors, “Laminar.” [Online]. Available: https://laminar.sh/
[36] K. Zhang, M. Tenenholtz, K. Polley, J. Ma, D. Yarats, and N. Li, “BrowseSafe: Understanding and preventing prompt injection within AI browser agents,” 2025, *arXiv:2511.20597*.
[37] K. Huang and C. Hughes, *Securing AI agents: Foundations, frameworks, and real-world deployment*, Cham: Springer Nature Switzerland, 2025.
[38] OpenAI, “OpenAI o3 and o4-mini system card,” 2025. [Online]. Available: https://openai.com/de-DE/index/o3-o4-mini-system-card/
[39] Meta, “Meta-llama/llama-4-maverick-17b-128e.” [Online]. Available: https://huggingface.co/meta-llama/Llama-4-Maverick-17B-128E
[40] Anthropic, “System card: Claude Opus 4 & Claude Sonnet 4,” 2025. [Online]. Available: https://www-cdn.anthropic.com/6d8a8055020700718b0c49369f60816ba2a7c285.pdf
[41] Center for Internet Security, “CIS benchmarks,” 2026. [Online]. Available: https://www.cisecurity.org/cis-benchmarks
[42] European Commission, “NIS2 directive: Securing network and information systems,” 2025. [Online]. Available: https://digital-strategy.ec.europa.eu/en/policies/nis2-directive
[43] NATO CCDCOE, “Locked Shields.” [Online]. Available: https://ccdcoe.org/locked-shields/