A Self-Synchronizing Cyber Deception Framework via Infrastructure as Code Reflection

Junyeong Park
Dept. of Computer and Information Security
Sejong University
Seoul, Republic of Korea
pjy010218@sju.ac.kr

Sayeon Kim
Cyber Electronic Warfare R&D
LIG Nex1
Pangyo, Republic of Korea
sayeon.kim@lignex1.com

Woohyun Jang
Cyber Electronic Warfare R&D
LIG Nex1
Pangyo, Republic of Korea
woohyun.jang2@lignex1.com

Yeon-Jae Kim
Cyber Electronic Warfare R&D
LIG Nex1
Pangyo, Republic of Korea
yeonjae.kim@lignex1.com

Shinwoo Shim
Cyber Electronic Warfare R&D
LIG Nex1
Pangyo, Republic of Korea
shimshinwoo@lignex1.com

Olmi Lee
Cyber Electronic Warfare R&D
LIG Nex1
Pangyo, Republic of Korea
olmi.lee@lignex1.com

Ki-Woong Park*
Dept. of Computer and Information Security
Sejong University
Seoul, Republic of Korea
woongbak@sejong.ac.kr

***Abstract*—As cyberattacks become more sophisticated, the use of honeypots has emerged as an alternative to proactively collect attackers' exploit strategies. However, conventional honeypots often lack realism and require much human effort to deploy and maintain in modern IT infrastructures. This excessive cost in resources creates a major obstacle to their widespread use. To address these challenges, this paper proposes a self-synchronizing cyber deception framework that upholds the declarative approach of Infrastructure as Code (IaC), maintaining idempotency and consistency while automatically generating a deceptive environment. Our framework treats the target system's IaC files as a blueprint to automatically generate and deploy a high-fidelity, "digital twin" deception environment. Our framework uses an automated pipeline to analyze the IaC file, apply the predefined transformation rules, and dynamically build new container images - replicating structural elements while replacing the core application logic with a honeypot. After deployment, the framework keeps the deceptive environment synchronized with the original system by automatically re-deploying the pipeline in response to any changes in the source IaC file, increasing fidelity and reducing management costs. The implementation of this prototype successfully shows a reduction in the manual effort required for deploying and maintaining deception environments, presenting a scalable and sustainable framework for active defense. This provides a strong foundation for building the next-generation defense mechanisms that can adapt to both evolving cyberattacks and changing infrastructure.**

Keywords—deception, honeypot, IaC, automation

I. Introduction

As the cybersecurity landscape evolves, cyberattacks have become equally sophisticated and persistent. Now the paradigm of cyberattacks has shifted to persistence, where attackers target specific systems and routes, invest more time and effort, and collect more information about the target system through a long period of time. Such persistence has led attackers to successfully bypass traditional defense mechanisms such as firewalls and intrusion detection systems (IDS), regardless of user caution. This paradigm has shed a light upon an active defense mechanism, a proactive approach where defenders engage with, mislead, and or study attackers to generate cyber threat intelligence. Within this context, cyber deception, particularly through the use of honeypots, has emerged as a powerful strategy to achieve these objectives.

However, despite its theoretical promise, the application of cyber deception in real-world enterprise environments is considered significantly difficult. First, the fidelity of conventional honeypots offer a dilemma; low-interaction honeypots are easily identified and bypassed by attackers, while high-interaction honeypots, though more realistic, are complex and resource-intensive to build and manage, with the risk of using one's own real-world data; something that was intended to be protected. Second, the operational cost associated with deploying, configuring, and maintaining a deception environment is too high for a small return. The use of deceptive environments requires expertise and manual effort, a cost that rises drastically in large-scale infrastructures—ironically, the very systems most targeted by attackers.

The most critical challenge comes from rapidly changing modern IT environments. Declarative tools like Kubernetes, Terraform, and Docker Compose make modern IT environments dynamic, with services being updated and reconfigured continuously with ease. Considering this context, using a manually deployed, static deception environment, designed to emulate the real environment, seems unrealistic. As the production system evolves, the deception environment will fail to adapt, creating a fatal gap that impacts its realism and thus making it all the more useless.

To address these fundamental limitations, this paper introduces a Self-Synchronizing Cyber Deception Framework designed for modern, Infrastructure as Code (IaC) driven infrastructures. The primary objective of this research is to design and implement a framework that automates the lifecycle of high-fidelity deception environments; from creation to maintenance, ensuring that they remain synchronized with the original environment to reduce manual reconfiguration. Our framework leverages a methodology called "IaC Reflection", which involves programmatically parsing a target system's IaC file, analyzing its architecture, and transforming it into a "digital twin" honeypot. In this process, our framework replicates the original system's structure, services versions, and configurations, while replacing the core application logic with

predefined honeypot services or pre-made 'dummy logic'. After the deployment of the replicated environment, the Sync Controller monitors the original IaC file for changes. Upon said change, the controller automatically triggers the entire pipeline again, restarting the lifecycle and thus eliminating the need of human intervention.

The core achievement of this research is presenting a scalable framework that makes advanced cyber deception strategies a viable and cost-effective strategy for constantly changing real-world systems. By reducing the required manual effort and needed expertise, our framework lowers the barrier to entry for adopting deceptive strategies. By providing an automatically generated, and self-maintaining deception environment, this research lays a concrete foundation for next-generation security strategies. It is capable of evolving in sync with growing dynamic infrastructures that attackers may target. This paper details the architecture of our framework, the implementation of its modules, and demonstrates its effectiveness in creating and maintaining a realistic deception environment from an industry standard IaC file.

II. Related Work

The introduction of IaC and the integration of its principles over various fields in the cyber domain have reduced significant overhead in both the initial deployment and the implementation of infrastructural change, allowing better cost-efficiency and flexibility in organizational operations [1]. This shift of paradigm has also opened new opportunities for cybersecurity, including an automated approach in deploying dynamic security environments, like security-focused digital twins [2]. However, while automated infrastructure deployment is key contribution of IaC, the need for dynamic control and management of the entire IaC life cycle is on the rise in order to keep up with the rapidly changing modern IT infrastructure [3]. This trend is where our research lies, addressing the need for a dynamic, self-synchronizing system in the cyber deception domain.

Within this context, the concept of replicating existing architecture for active deception, such as high-fidelity honeypots in conjunction with digital twins, is an emerging area of research. The initial drive of the research in this domain was to overcome the limited use cases of honeypots; to expand from single instances to entire systems. We mainly discuss these digital twin-based approaches as it successfully aligns with our IaC-based approach, such as the use of replicate environments or an automated lifecycle of the deception environment.

Previous works have proposed various ways of using digital twins for cybersecurity. Yigit et al. proposed 'TwinPot', a digital twin-based honeypot that was designed to detect and analyze external attacks on smart ports [4], and Suhail et al. proposed 'INCEPTION', an automated digital twin-based deception platform, focusing more on the automated generation and management of the deception environment [5]. However, the two research rely mostly on pre-existing digital twins, using them as blueprints for creating the actual honeypots. This results in a three-layer architecture; the original system, digital twin, and the honeypot. Naturally, this becomes a costly prerequisite for systems that do not deploy digital twins. These early approaches, as emphasized by Heluany et al., have largely remained at a theoretical or conceptual level, highlighting the gap between the potential of digital twin deception and its automated implementation [6].

Several studies have suggested multiple ways to automate the deployment of deception environments. For example, Kahlhofer et al. proposes 'Koney', a deception orchestration framework designed for Kubernetes [7]. Koney introduces a method called "deception policy documents" to convert deception techniques into code. It also uses a Kubernetes Operator to automatically inject traps in active services or applications. Koney is very powerful as it modifies applications at runtime, without requiring source code access. However, the method of Koney differs from what we propose. Koney is a 'runtime mutation' framework, where as our research focuses on pre-deployment generation and synchronization. This approach differs in which process the framework sees valuable. Koney recognized the importance of altering a service behavior, where as the self-synchronizing framework sees importance in the redeployment of the entire structure that also serves as a honeypot.

Furthermore, new approaches in the deception strategy itself are also notable, where the core difference lies in the response of the deceptive environment according to the attacker's actions, which ultimately leads to an 'adaptive' deception strategy. The primary goal of these research is to maximize attacker engagement by varying the honeypot's behavior and or appearances. For example, Safargalieva et al. suggests bio-inspired behavior such as camouflage and playing dead, to make the honeypot more believable [8]. While these adaptive honeypots present new possibilities for honeypot utilization, the effectiveness still relies heavily on the rapid change and reconfiguration of the deception system. Any intelligent response mechanism must be set upon a quick, automated deployment and synchronization framework in order for it to function as intended.

Another approach involves creating a dynamic 'deception grid' using Software-Defined Networking (SDN). Guerra et al. proposed an architecture that uses SDN to dynamically configure and redirect traffic into a honeynet based on a pre-designed strategy [9]. This research contributes in a real-time modification of the network topology to mislead attackers. However, the deception strategy relies on only the topology, not the individual honeypots that actually make up the network. So the challenge remains: an automated, synchronized framework for fast and accurate deployment.

Our research addresses these foundational gaps, differing from the former strategies. Instead of focusing on *what* deception strategy to deploy, we focus on *how.* With this change in view, we create the underlying mechanism that creates dynamic and adaptive, synchronized deception practical. We use a method called "IaC Reflection" to create a two-layer architecture, making creation process of the deception environment inherently simpler, faster, and more efficient. Our framework suggests using the original system's IaC blueprint as the single source for desired deception environment, which enables a direct generation of a 'look-alike' that itself serves as a honeypot. This way, the generated deception environment possesses identical structure and configurations while also having different functionalities than the original, hence the term

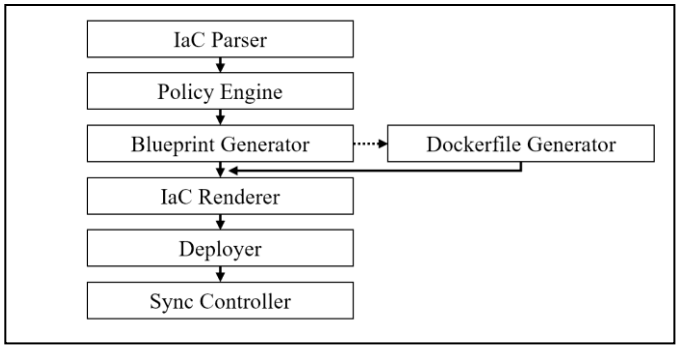

Fig. 1. Main Automated Pipeline.

'look-alike', not 'digital twin'. By eliminating the need for the intermediate, our approach not only solves the problem of continuous synchronization; by reducing the three-layer synchronization to a two-layer synchronization but also presents a more practical and efficient way for dynamic infrastructures.

III. Framework Architecture and Design

The Self-Synchronizing Cyber Deception Framework proposed in this research is designed to overcome the aforementioned limitations. The framework achieves this with three core philosophies: High-Fidelity, Full Automation, and Synchronization. First, high-fidelity is achieved through replicating the 'appearances' of the original system, such as base image, dependencies, or version that can be found in the source IaC file. Second, full automation and synchronization is achieved through creating automatically deployable modules for each step of the deception environment lifecycle; configuration, deployment, and maintenance. As a result, deploying and maintaining deception environments become cost-effective and thus, becomes a viable security option across various domains in modern infrastructures.

The framework is composed of seven independent modules as shown in Fig. 1—the IaC Parser, Policy Engine, Dockerfile Generator, Blueprint Generator, IaC Renderer, Deployer and Sync Controller. Each module is designed to perform a clearly defined responsibility, which allows the entire framework to harness sustainability and future scalability. Notably, the IaC Parser and Renderer modules are intentionally separated into multiple parsers and renderers, according to the IaC format (e.g., Kubernetes, Terraform), to support diverse IaC environments and maintain sustainability.

A. Overall Architecture

The framework is designed as a multi-stage pipeline. The entire process is intended to be automated, from making the source IaC file to transforming it into a deception environment. The pipeline, as shown in Fig 1, begins with the IaC Parser. The IaC Parser reads the source file and translates it into a standardized structure. This structure is then passed to the Policy Engine, which analyzes the translated structure and adds 'tags'; a component from a predefined set of rules that defines the relevant transformation actions.

The tagged data then moves to the Blueprint Generator, where the blueprint of the final deception environment is created. The Blueprint Generator reads these tags, then either modifies the component directly, or calls the Dockerfile Generator to create a new container image. After processing all components and inserting common configurations, the final blueprint is created and then sent to the IaC Renderer. The IaC Renderer converts the blueprint into a new and configured IaC file, and the Deployer deploys the IaC file, creating a new deception environment based solely on the source IaC file.

B. Core Concepts – IaC Reflection

The core concept in the Self-Synchronizing Cyber Deception Framework is "IaC Reflection". Unlike a text-based approach, this approach allows precision and reliability when transforming the source IaC file into desired blueprints. The IaC Parser goes beyond the basic search-and-replace tactic, instead it understands the relationships between services and volumes (in Docker Compose) or between services and pods (in Kubernetes). This makes the Policy Engine apply the predefined rules more properly and easily. For instance, a predefined rule may target the *image* of a specific service such as '*database(services.database.image)*', a tricky task for a simple string search where it might modify an *image* key elsewhere.

C. Core Concepts – Dynamic Build

In order to achieve high-fidelity in the final deception environment, this framework uses a dynamic build strategy, which is in contrast with the basic image replacement strategy. Image replacement represents the simple rule of substituting a production image with a pre-built, generic honeypot image. While the image replacement strategy is easy to implement, this approach is prone to attackers; an attacker can perform basic reconnaissance and immediately identify the generic honeypot.

Meanwhile, the dynamic build strategy aims to create a custom honeypot image, intentionally structured to be almost identical to the original service while having none of the actual application logic. The dynamic build is deployed by a specific module called 'the Dockerfile Generator', and is made up of these two operations:

- **Structure Replacement:** The foundation of the fidelity. The Dockerfile Generator parses the original service's Dockerfile, and reads the service's underlying structure. Then, the module replicates key instructions that define the service's structure. Some examples include *FROM* instructions to use the same base operating system, or *RUN* commands to install the same packages and libraries, or *COPY* commands for dependencies (e.g., *requirements.txt*). This results in a honeypot environment which is almost identical to the original environment.
- **Logic Replacement:** This is how the deceptive aspect is created. The Dockerfile Generator ignores the original *COPY* command, which normally inserts the actual application logic. Instead, it uses a new *COPY* command that places a pre-built honeypot application into the image. This honeypot application does not perform any actual functions nor use any real data from the original source. This creates a seemingly identical application from the outside, but the actual logic inside has been altered completely.

The final output is a new *Dockerfile.honeypot* which

produces a container image with the same operating system, same libraries, same dependencies but with completely different application logic.

D. Module Design

The framework consists of the seven modules listed below. This modular design is crucial for defining roles and responsibilities between each module, thus creating scalability.

- **IaC Parser:** The starting point of the pipeline. It reads the source IaC file and translates it into a standardized python data structure (e.g., dictionary for Docker Compose, list of dictionaries for Kubernetes).
- **Policy Engine:** The brains of the operation. It takes the translated IaC file, and attaches a pre-defined set of policies. The Policy Engine will find the correct objects in the translated IaC file and attach the replacement rule as an *'x-honeypot-policy'* tag. This module does not directly configure the components.
- **Dockerfile Generator:** This module is used when the dynamic build strategy is called. It takes the original service's build and build policy as input, and replicates the necessary dependencies to create a fake application. The Dockerfile Generator then outputs a new *'Dockerfile.honeypot'* to the designated directory.
- **Blueprint Generator:** The backbone of the operation. It takes the tagged information from the Policy Engine, reads the attached tags, and executes the actual transformations. Transformations may be in the form of simple image replacement, or a sophisticated dynamic build via Dockerfile Generator. This module also takes care of system-wide configurations such as logging services (e.g., Fluentd) or *healthcheck* operations.
- **IaC Renderer:** The final stage of the transformation process. It retrieves the final blueprint created by the Blueprint Generator and converts it back to the original format; the same format that the IaC Parser took as input.
- **Deployer:** This module is for the actual deployment of the final blueprint. The default state of the Deployer is an 'underlying' interface for the Sync Controller, but when necessary it also serves as a user interface for manual control. It provides crucial commands such as *up, down, status,* to control the deployment of the deception environment manually.
- **Sync Controller:** The eye that sees all. It monitors changes in the source IaC file using monitoring tools such as *watchdog*. If a change is detected, the Sync Controller reactivates the entire pipeline, re-configuring the blueprint and re-deploying the deception environment. This module enforces the "self-synchronizing" nature of the suggested framework.

IV. EXPERIMENTS

We analyzed the proposed claim and design of the Self-Synchronizing Cyber Deception Framework with three core experiments: (1) Verifying the accuracy of the constructed honeypots and evaluating the fidelity, (2) Analyzing the synchronization and efficiency to evaluate the framework's effectiveness in costs and complexity, compared to manual setup and configuration, and (3) Evaluating the scalability of the framework using bigger, more complex source files.

A. Experiment Setup

The experiment was conducted using Docker on a Windows 11 environment. The framework was executed with Python 3.10, and the source infrastructure file was defined in a Docker Compose file. As mentioned above, transformation policies were pre-defined. To compare the generated honeypot and its fidelity, an unmodified *nginx* service and a generic *httpd:alpine* (*Apache*) service were used as baselines. Basic network scanning tools such as *Nmap* and *WhatWeb* were used for scanning the deployed environment.

B. Experiment 1: Verifying Accuracy

The first core concept of this experiment is to analyze the structural fidelity of the honeypot, specifically generated by the framework's dynamic build strategy. By comparing the metadata of containers created from both an original image and the generated honeypot image, we prove that the generated honeypot container is of high-fidelity, which replicates the original service's structure, while successfully replacing the internal logic with a 'dummy' logic. We first built an original image using the source *api/Dockerfile*, in order to create a standard for comparing the honeypot image. Then we executed the framework, which automatically generated the honeypot image using the dynamic build strategy. Containers were created from both images, and inspected through the *docker inspect* command. We extracted detailed metadata of both containers and compared core structures.

The metadata comparison, as summarized in Table 1, indicates that the core characteristics of both structures are identical. The base image, environment variables, and working directory are exactly replicated in the honeypot container metadata, creating a believable, seemingly identical high-fidelity honeypot. The only meaningful difference can be found in the CMD field in Table 1, which specifies the container's startup command. The original container runs *real_app.py*, while the honeypot runs *app.py*. This difference proves that the framework is successfully working as intended; preserving the structure of the original service while only replacing the internal service logic. This experiment proves the framework's ability to achieve high-fidelity through the "IaC Reflection" method.

The second core concept is to evaluate the fidelity of the digital twin honeypot created by the dynamic build strategy. By assessing its outputs against popular reconnaissance tools like

TABLE I. COMPARISON OF ORIGINAL VS DIGITAL TWIN HONEYPOT CONTAINER METADATA

Attribute	Container	
	Original Container	***Digital Twin Honeypot***
"Env"	PYTHON_VERSION=3.9.23	PYTHON_VERSION=3.9.23
"Env"	PATH=/usr/local/bin:/usr/	PATH=/usr/local/bin:/usr/
"ExposedPorts"	5000/tcp	5000/tcp
"Cmd"	"python", "real_app.py"	"python", "app.py"

TABLE II. FINGERPRINT SCAN RESULTS

Target	Outputs		
	Ports	***Tool***	***Signature***
Digital Twin *(api)*	5000	nmap -sV	"Unrecognized"
	5000	whatweb	Python/3.923 Werkzeug/2.3.8
Original Webserver (nginx)	8080	nmap -sV	http, nginx 1.29.0
	8080	whatweb	Nginx/1.29.0
Generic (httpd)	8081	nmap -sV	http, Apache httpd 2.4.65
	8081	whatweb	Apache/2.4.65

Nmap and *WhatWeb*, we evaluate the generated honeypot's 'stealthiness'. As mentioned above, we used an original, unmodified *nginx* web application and a generic Apache service as baselines. As shown in Table 2, there are significant differences in the outputs. Both *Nginx* and *Apache* were accurately identified by the automated scanning tools, along with their service names and versions. However, the honeypot generated from the dynamic build strategy was not recognized by *Nmap* as a known service. *WhatWeb* was only able to identify the underlying *Python/Werkzeug,* without noticing any honeypot signatures. This "unrecognized" state shows the success of the dynamic build strategy in creating a high-fidelity honeypot that cannot be detected or classified as a generic honeypot.

C. Experiment 2: Self-Synchronization and Efficiency

The core concept of this experiment is to evaluate the 'self-synchronizing' ability of the framework and how efficient it is compared to a manual configuration. By measuring the average time it took from reaction to the redeployment of the changed infrastructure, we prove the automation is indeed vastly quicker and cost-effective.

We started the experiment with monitoring the source for any changes. A pre-defined change (e.g., modifying a service's port setting, changing the MySQL version, etc.) was applied to the source file, and we measured the total time took from the file save to the redeployment and reflection of the changed environment. This process was repeated 10 times in order to calculate the average time of the automation. The average response time, from detecting the change to redeploying the modified environment, was 26.326 seconds as shown in Table 3. This rapid automation shows the framework's ability to maintain sync with the original environment in almost real-time.

Furthermore, the automated framework reduces the entire modification process to a single step, making the obstacle in modification, deployment, and maintenance, trivial. Manual synchronization requires several steps; manually stopping the current environment, editing the IaC source file, attaching and reconfiguring according to policy, regenerating the deception blueprint, and manually redeploying the new environment.

While the manual approach requires multiple steps as mentioned, the automated approach need only one step (with the exception of configuring the pre-defined policies); configurint the IaC file. The proposed automation lets the user to save time and resources, while also minimizing the possibility of human error which can lead to malfunctioning deception environments.

TABLE III. AVERAGE RESPONSE TIME PER MODIFICATION

Rounds	Container	
	Modifications	***Response Time***
1	Port Settings Change (8080:80 → 8888:80)	24.63s
2	Port Settings Change (8888:80 → 8080:80)	25.87s
3	Port Settings Change (5000:5000 → 6060:6060)	26.48s
4	Port Settings Change (6060:6060 → 5000:5000)	26.71s
5	MySQL Version Change (mysql:8.0 → mysql:7.0)	27.19s
6	MySQL Version Change (mysql:7.0 → mysql:8.0)	26.05s
7	MySQL Version Change (mysql:8.0 → mysql:6.0)	26.58s
8	MySQL Version Change (mysql:6.0 → mysql:8.0)	26.78s
9	MySQL & Port Settings Change	26.78s
10	MySQL & Port Settings Change	27.19s
Average	**26.326s**	

The framework not only provides speed, but stability as well.

D. Experiment 3: Scalability

The core concept of this experiment is to evaluate the scalability of the framework's blueprint generating. In other words, the blueprint generation must be of no fault for larger, more complex infrastructures. By measuring the average 'final blueprint' creation time, we prove that the computational resources do not increase drastically, and still provides a cost-effective and efficient way to deploy and manage large complex infrastructures.

We created a complex *docker-compose.yml* file including 25 services, all of which are random. A *policy.yml* file was also created to apply pre-defined policies to 10 of the 25 services. Then, we measured the time taken from parsing the source file to creating the final blueprint for each round, and calculated the average time taken.

As shown in Table 4, for 5 rounds, the framework successfully processed 25 services and generated the final blueprint in an average time of 0.027 seconds. In comparison to the simple setup's average blueprint creation time of 0.012 seconds, this indicates that the processing time and computation did not increase linearly, showing the framework's viability for the widespread use in complex systems, with little overhead in the deployment of the deception environment.

TABLE IV. AVERAGE BLUEPRINT CREATION TIME PER SERVICE

Rounds	Complexity	
	***Simple Setup** (3 services)*	***Complex Setup** (25 services)*
1	0.013s	0.028s
2	0.011s	0.029s
3	0.013s	0.026s
4	0.012s	0.026s
5	0.011s	0.027s
Average	**0.012s**	**0.027s**

E. Discussion

The experiment results prove that the Self-Synchronizing Cyber Deception Framework successfully achieves its intended goals. The efficiency and fast processing times from experiment 2 and 3 show that this framework could be considered as a viable option against the high-cost deceptive strategies. By reducing a multi-step manual job into a single automated process, the suggested framework not only saves time and resources, but also eliminates the possibilities of human error. Also, the fidelity evaluation proves the framework's ability to create believable honeypots without using any real data or application logic. The dynamic build strategy acts as a core mechanism that creates seemingly indistinguishable honeypots; a crucial factor in deceptive environments. Overall, the proposed framework is an effective framework in solving the aforementioned gaps and limitations of current deception strategies.

However, considering that the experiments were done in controlled environments with sample IaC files, real-world use has still room for validation. Real-world enterprise IaC may contain complex logic, including user-defined components. This requires more dynamic modules, such as enhanced parsing modules or adaptable transformation engines.

V. CONCLUSION

This paper addressed a critical challenge in the active cyber deception domain: the difficulties of deploying and maintaining high-fidelity deception environments in rapidly growing modern IT infrastructures. Conventional honeypots quickly lose their effectiveness due to the configuration gap, while automated approaches often required costly prerequisites such as pre-existing digital twins. To fill these gaps, we proposed a Self-Synchronizing Cyber Deception Framework based on the "IaC Reflection" method. Our framework successfully demonstrates a fully automated pipeline that transforms the original environment's IaC blueprint into a high-fidelity self-synchronizing honeypot.

There are three core contributions of this paper. First, by treating the IaC file as a single source of input, our framework creates a two-layer honeypot architecture. Second, the viability of the dynamic build strategy achieves the creation of high-fidelity honeypots. These honeypots have the same structural dependencies but do not use real application logic, which allows users to create a more realistic, seemingly identical honeypot. Third, with the self-synchronizing mechanism, whenever a change in infrastructure is made (in the source IaC file), the framework can apply the changes automatically, solving the gap between the modified original and the unmodified deception environment in real-time. As the original system changes over time, the self-synchronization mechanism provides stability and cost reduction.

The final results of the framework also suggest a successful integration of IaC and DevSecOps. By directly connecting the source IaC file with the domain of cyber deception, cyber deception is no longer an extra step to take, especially in the lifecycle of mass infrastructures. By a completely automated, self-synchronizing framework, the cyber security space is kept in-sync with the original infrastructure, providing a strong lure for infrastructure owners to be more willing to accept a more active security strategy. With one integration from the initial planning and deployment of the infrastructure, active deceptive strategies can easily be a part of keeping the system safe.

While the framework successfully addresses the aforementioned limits in current active defense systems, the framework still has potential for expansion. First, the current framework handles simple IaC constructs. It may not support advanced, complex logic in real-world IaC examples. Also, efficiency was measured qualitatively. While the efficiency gain was evident, using a quantitative approach would be required in further research. Second, the dynamic build strategy is generic in behavior. The next step would be to enhance the 'behavioral fidelity' by dynamically generating application logic. Further work will be focused not only on increasing support for broad IaC features, but also addressing intelligent approaches in the dynamic build strategy.

ACKNOWLEDGMENT

This work was supported by Korea Research Institute for Defense Technology Planning and Advancement (KRIT) – Grant funded by Defense Acquisition Program Administration (DAPA) (KRIT-CT-22-051).

REFERENCES

[1] S. I. Abbas and A. Garg, "Integrating Emerging Technologies with Infrastructure as Code in Distributed Environments," *2024 3rd International Conference on Applied Artificial Intelligence and Computing (ICAAIC)*, Salem, India, 2024, pp. 1138-1144, doi: 10.1109/ICAAIC60222.2024.10575600.

[2] K. Hammar and R. Stadler, "Digital Twins for Security Automation," *NOMS 2023-2023 IEEE/IFIP Network Operations and Management Symposium*, Miami, FL, USA, 2023, pp. 1-6, doi: 10.1109/NOMS56928.2023.10154288.

[3] Pahl, Claus et al. (2025). Infrastructure as Code -Technology Review and Research Challenges. 10.5220/0013247700003950.

[4] Y. Yigit, O. K. Kinaci, T. Q. Duong and B. Canberk, "TwinPot: Digital Twin-assisted Honeypot for Cyber-Secure Smart Seaports," *2023 IEEE International Conference on Communications Workshops (ICC Workshops)*, Rome, Italy, 2023, pp. 740-745, doi: 10.1109/ICCWorkshops57953.2023.10283756.

[5] S. Suhail, M. Iqbal and K. McLaughlin, "Digital-Twin-Driven Deception Platform: Vision and Way Forward," in *IEEE Internet Computing*, vol. 28, no. 4, pp. 40-47, July-Aug. 2024, doi: 10.1109/MIC.2024.3406188.

[6] Heluany, J., Amro, A., Gkioulos, V., & Katsikas, S. (2024, April). Interplay of Digital Twins and Cyber Deception: Unraveling Paths for Technological Advancements. *2024 IEEE/ACM 4th International Workshop on Engineering and Cybersecurity of Critical Systems and 2024 IEEE/ACM Second International Workshop on Software Vulnerability (EnCyCriS/SVM), 20–28.*

[7] *Kahlhofer, Mario & Golinelli, Matteo & Rass, Stefan. (2025). Koney: A Cyber Deception Orchestration Framework for Kubernetes. 10.48550/arXiv.2504.02431.*

[8] Safargalieva, A., & Vasilomanolakis, E. (Accepted/In press). Towards bio-inspired cyber-deception: a case study of SSH and Telnet honeypots. In *Proceedings of 4th Workshop on Active Defense and Deception (AD&D) : Co-located with the 10th IEEE European Symposium on Security and Privacy (Euro S&P)* IEEE.

[9] Guerra, Luís Maria de Figueiredo Cruz - Proactive Cybersecurity tailoring through deception techniques. Lisboa: Instituto Superior de Engenharia de Lisboa, 2023. Dissertação de Mestrado