The Transactions of the Korea Information Processing Society
TKIPS Vol.14, No.4, pp.215~223
eISSN: 3022-7011
https://doi.org/10.3745/TKIPS.2025.14.4.215

Bambda: A Framework for Preventing Function Invocation Condition-Based Attacks in Serverless Environments

Shin Chang Hee† · Lee Seung Soo††

ABSTRACT

Serverless computing is rapidly emerging as a new paradigm in cloud computing, offering automatic scalability, cost efficiency, and ease of operation. However, its two core characteristics—IAM-based privilege management and event-driven execution—can introduce security vulnerabilities. In particular, complex inter-functional call relationships make serverless applications susceptible to attacks such as privilege bypass and event trigger exploitation. Existing approaches, including static analysis and data tagging, have limitations in real-time threat response and developer productivity in dynamic serverless environments. In this paper, we propose Bambda, a dynamic security framework tailored for serverless environments. Bambda performs real-time function call verification through centralized logging and automated code injection using AWS CloudWatch. By implementing a multi-step verification process that distinguishes between direct and event-driven calls, Bambda effectively prevents bypass attacks without requiring additional security configurations from developers. Experiments in AWS Lambda environments validate its effectiveness in defending against privilege escalation and chained function call attacks while maintaining practical performance overhead.

Keywords : Serverless, Cloud Computing, Function Invocation Condition-Based Attacks

Bambda: 서버리스 환경에서의 함수 호출 조건 기반 공격 방어 프레임워크

신 창 희† · 이 승 수††

요 약

서버리스 컴퓨팅은 자동 확장성, 비용 효율성, 운영 편의성을 제공하여 클라우드 컴퓨팅의 새로운 패러다임으로 급속히 확산되고 있다. 그러나 서버리스 환경의 두 가지 핵심 특성인 IAM 기반 권한 관리와 이벤트 기반 실행 메커니즘은 보안 취약점으로 작용할 수 있다. 특히 복잡한 함수 간 호출 관계로 인해 권한 우회나 이벤트 호출 조건 악용과 같은 공격에 노출되기 쉽다. 기존 연구들은 정적 분석이나 데이터 태깅을 통한 해결책을 제시했으나, 동적인 서버리스 환경에서의 실시간 위협 대응이나 개발자의 생산성 측면에서 한계를 보인다. 본 논문에서는 서버리스 환경에 특화된 동적 보안 프레임워크인 Bambda를 제안한다. Bambda는 AWS CloudWatch를 활용한 중앙집중식 로깅과 자동화된 코드 삽입을 통해 실시간 함수 호출 검증을 수행한다. 특히 직접 호출과 이벤트 기반 호출을 구분하여 검증하는 다단계 프로세스를 도입하여, 개발자의 추가적인 보안 설정 없이도 효과적인 우회 공격 차단이 가능하다. AWS Lambda 환경에서의 실험을 통해 권한 우회와 연쇄적 함수 호출 공격에 대한 효과적인 방어 능력을 입증했으며, 성능 오버헤드 측면에서도 실용적 수준의 구현이 가능함을 확인했다.

키워드 : 서버리스, 클라우드 컴퓨팅, 함수 호출 기반 공격

※ 이 논문은 2024년 ACK 2024의 우수논문으로 “Bambda: 서버리스 환경을 위한 우회 공격 차단 프레임워크”의 제목으로 발표된 논문을 확장한 것임.
※ 이 논문은 2022년도 정부(과학기술정보통신부)의 재원으로 정보통신기획평가원의 지원을 받아 수행된 연구임 (RS-2022-II220740, 다크웹 은닉서비스 식별 및 근원지 추적기술 개발).

† 준 회 원 : 인천대학교 컴퓨터공학부 학사과정
†† 종신회원 : 인천대학교 컴퓨터공학부 조교수
Manuscript Received : January 3, 2025
Accepted : March 7, 2025
* Corresponding Author : Lee Seungsoo(seungsoo@inu.ac.kr)

1. 서 론

클라우드 컴퓨팅의 진화는 기존의 플랫폼 서비스(Platform as a Service, PaaS)와 소프트웨어 서비스(Software as a Service, SaaS) 사이에서 새로운 패러다임으로서 서버리스 컴퓨팅을 등장시켰다. 함수형 서비스(Function as a Service, FaaS)라고도 불리는 이 컴퓨팅 방식은 개발자가 복잡한 가상 머신 관리를 신경 쓰지 않고 애플리케이션 코드에 집중할 수 있게 해준다는 장점으로 인해 급격히 확산되고 있다[1]. 서버리스 환경에서는 개발자가 작성한 함수가 클라우드상에서 자동으로 실행되고 관리되며, 리소스 할당과 서버 관리를 클라우드 서비스 제공자가 담당함으로써 애플리케이션 운영의 복잡성이 크게 감소된다. 함수 요청 트래픽에 따라 리소스가 자동으로 조절되어 개발자는 실제 사용된 리소스에 대해서만 비용을 지불하게 되므로 높은 운영 효율성을 달성할 수 있다[2-4].

서버리스 컴퓨팅의 두 가지 핵심 특성인 IAM(Identity and Access Management) 기반 권한 관리와 이벤트 기반 실행 메커니즘은 운영 효율성과 보안 위험이라는 상충된 특성을 내포한다. IAM은 서버리스 플랫폼에서 사용자 인증과 권한 관리를 담당하는 핵심 보안 메커니즘으로 높은 수준의 보안 통제를 가능하게 하지만, 부적절한 IAM 구성은 우회적인 무단 데이터 접근이나 비용 증가와 같은 공격을 초래할 수 있다. 한편, 이벤트 기반 실행은 함수 요청에 따라 자동으로 리소스를 조절하여 효율성을 극대화하지만, 복잡한 함수 간 호출 관계로 인해 DoS(Denial of Service)[23], DoW(Denial of Wallet) [22], 무단 데이터 접근과 같은 보안 취약점이 발생할 수 있다.

이러한 보안 위험에도 불구하고 대응 방안은 여전히 제한적이다. 2020년 DivvyCloud 보고서[5]에 따르면, IAM의 부적절한 구성으로 인한 기업의 경제적 손실이 약 5조 달러에 달하는 것으로 추정되며, 2024년 Google Mandiant[21]의 분석에서는 클라우드 이용자의 잠재적 취약점 중 잘못된 IAM 구성이 30.3%로 높은 비중을 차지한다. AWS는 현재 사전 IAM 정책 검증 도구만을 제공할 뿐, 배포 이후 동적으로 발생하는 보안 위협을 감지하고 차단하는 서비스는 제공하지 않는다. 최근 연구들에서도 정적 권한 정책 분석이나 데이터 태깅 기반의 접근 제어를 제안하고 있으나[6-12], 동적인 서버리스 환경에 적합하지 않거나 사용자의 상당한 개입을 요구하여 운영 편의성을 저해한다.

이러한 한계를 해소하고자 서버리스 환경의 동적 특성을 고려한 실시간 보안 프레임워크 Bambda를 제안한다. Bambda는 AWS CloudWatch[19]를 활용한 실시간 모니터링과 자동화된 코드 삽입을 통해, 개발자의 추가적인 개입 없이도 서버리스 애플리케이션의 보안성을 강화할 수 있는 실용적인 솔루션을 제시한다.

본 논문이 기여하는 바는 다음과 같다.

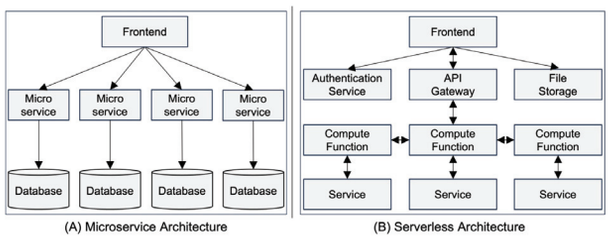


Fig. 1. The architecture of microservice and serverless application

- AWS CloudWatch를 확장한 중앙 집중식 로깅을 활용하여 실시간으로 발생하는 보안 위협을 동적으로 감지하고 차단하는 보안 프레임워크의 설계 및 구현을 제시한다.
- 서버리스 함수의 구조를 자동으로 분석하고 보안 검증 로직을 자동으로 삽입하는 코드 재구성 메커니즘을 제안하여, 개발자의 추가적인 보안 설정이나 코드 수정 없이도 즉시 프레임워크를 적용할 수 있도록 한다.
- 서버리스 함수 간의 직접 호출과 이벤트 기반 호출을 구분하여 검증하는 프로세스를 통해 권한 우회 접근과 연쇄적 함수 호출 공격을 차단하는 메커니즘을 제시한다.

2. 배경 지식 및 문제 상황

2.1 배경 지식

1) 서버리스 컴퓨팅

서버리스 컴퓨팅은 인프라스트럭처 관리에서 벗어나 순수 애플리케이션 로직에 집중할 수 있도록 설계된 클라우드 컴퓨팅 모델이다. Fig. 1은 전통적인 마이크로서비스 아키텍처와 서버리스 아키텍처의 구조적 차이를 보여준다. 마이크로서비스 아키텍처에서는 서비스 단위로 애플리케이션을 분할하여 각각 독립된 컨테이너로 구성하고, 이들의 배포와 운영을 직접 관리해야 한다. 각 서비스는 독립적인 데이터베이스를 가지며, 지속적으로 활성화된 상태를 유지한다.

반면 서버리스 컴퓨팅은 이러한 운영 부담을 획기적으로 경감시킨다. 비즈니스 로직을 무상태(Stateless) 함수 단위로 정의하여 배포하며, 이후의 인프라스트럭처 관리는 클라우드 서비스 제공자(Cloud Service Provider, CSP)가 전적으로 담당한다. 서버리스 아키텍처의 핵심 특징은 이벤트 기반 실행 모델과 자동 확장성(Autoscaling)에 있다. 함수 실행은 HTTP 요청, 데이터베이스 변경, 스토리지 업데이트 등 다양한 이벤트에 의해 트리거될 수 있으며, 이러한 이벤트들은 API Gateway를 통한 라우팅이나 클라우드 서비스에서의 직접 호출 방식으로 함수를 실행시킨다. 모든 함수는 실행이 완료되면 함수가 비활성화되며, 플랫폼은 함수 호출 빈도와 워크로드에 따라 컨테이너를 자동으로 프로비저닝하고 회수하는 탄력적인 리소스 관리를 수행한다. 이를 통해 개발자는 인프라 확장성을 고민할 필요 없이 비용 효율적인 운영이 가능하다.

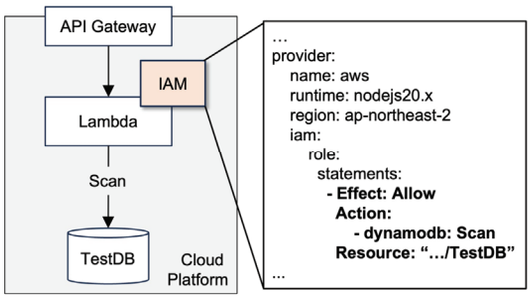


Fig. 2. The example of IAM role permissions in the Lambda serverless function

2) 서버리스 함수

서버리스 함수는 서버리스 컴퓨팅 아키텍처에서 비즈니스 로직을 실행하는 기본 단위로, 특정 이벤트나 API 호출에 응답하는 독립적인 코드 모듈이다. AWS Lambda[16], Google Cloud Functions[25], Azure Functions[26] 등이 대표적인 서버리스 함수 서비스로, Fig. 2는 AWS Lambda 환경에서의 서버리스 함수 실행 흐름을 보여준다. 클라이언트의 요청은 API Gateway를 통해 Lambda 함수로 전달되며, 이 과정에서 IAM(Identity and Access Manage-ment)을 통한 권한 검증이 수행된다. 권한이 확인된 함수는 필요한 클라우드 서비스(예: DynamoDB[20], S3[27] 등)와 상호작용하여 요청된 작업을 처리한다.

서버리스 함수의 주요 특징은 권한 기반의 실행 제어에 있다. Fig. 2에서 볼 수 있듯이, Lambda 함수가 DynamoDB의 'TestDB'에 대해 'Scan' 작업을 수행하기 위해서는 해당 권한이 명시적으로 부여되어야 한다. 이러한 세밀한 권한 제어는 보안성을 강화하지만, 동시에 잘못된 구성은 보안 취약점으로 이어질 수 있다. 특히 AWS Lambda, Goo-gle Cloud Functions 등 서비스 제공자마다 권한 관리 방식이 상이하여, 각 플랫폼에 최적화된 보안 설정이 요구된다.

2.2 문제 상황

서버리스 함수의 권한 부여 특성과 이벤트 호출로 인해 서버리스 컴퓨팅 아키텍처의 애플리케이션은 여러 보안 위협에 노출될 수 있다. 본 연구에서는 AWS Lambda 기반의 데모 애플리케이션인 Hello Retail![13]을 분석하여 발생 가능한 우회 공격 패턴을 식별했다. Hello Retail!은 제품 등록, 검색, 구매와 같은 전자상거래의 핵심 기능들이 독립적인 서버리스 함수들로 구현되어 있어, 서버리스 환경의 보안 취약점을 분석하기에 적합한 테스트베드를 제공한다. 본 섹션에서는 Fig. 3을 통해 식별된 두 가지 주요 공격 시나리오와 이로 인한 보안 위협을 분석한다.

C1. 서버리스 함수 권한 남용 기반 리소스 우회 접근. 서버리스 함수는 다른 서비스와의 상호작용을 위해 필요한 권한을 부여받아야 하며, 이는 최소 권한 원칙에 따라 엄격하게 제한되어야 한다. 그러나 개발 편의성을 위해 와일드카드(*)를 사용

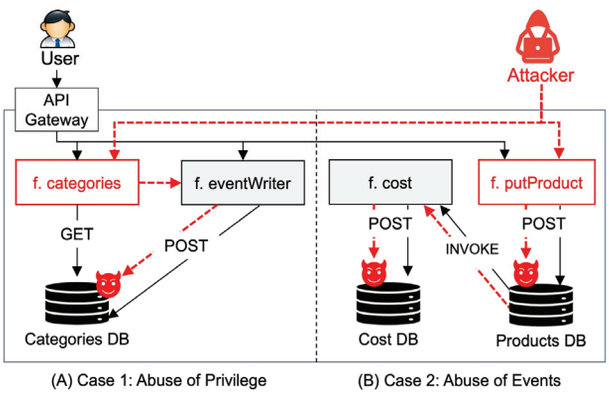


Fig. 3. An example of serverless function abuse using a bypass attack scenario

하여 과도한 권한을 부여하는 경우가 빈번하고, 실제로 사용되는 권한은 2%로 측정되었다[24]. Fig. 3 (A)는 이러한 부적절한 설정이 권한 남용으로 이어지는 우회 접근 시나리오를 보여준다. 정상적인 사용자는 API Gateway를 통해 'categories' 함수를 GET 요청으로 호출하여 Categories DB에 접근하지만, 공격자는 와일드카드로 과도한 권한이 부여된 'categories' 함수를 이용하여 'eventWriter' 함수를 호출함으로써 본래 접근 권한이 없는 Categories DB에 우회적으로 접근할 수 있다. 더욱이 이러한 우회 접근은 함수의 반복 호출을 통해 서비스 거부(Denial of Service, DoS) 공격이나 과도한 비용을 유발하는 DoW(Denial of Wallet) 공격으로 확장될 수 있다. 이를 방지하기 위한 클라우드 벤더의 보안 서비스들이 존재하지만, 이들은 정적 분석에 기반하여 동적인 서버리스 환경에서는 실효성이 떨어지거나 비용 부담이 크다는 한계가 있다.

C2. 서버리스 함수 호출 조건 이벤트 악용. 서버리스 함수의 이벤트 기반 실행 특성은 또 다른 형태의 우회 공격을 가능하게 한다. Fig. 3 (B)는 데이터베이스 업데이트 이벤트를 악용한 공격 시나리오를 보여준다. 'cost' 함수는 Products DB에 새로운 데이터가 업로드될 때마다 자동으로 실행되도록 설계되어 있다. 공격자는 이러한 이벤트 호출 조건을 악용하여 'putProduct' 함수를 통해 의미 없는 데이터를 Products DB에 반복적으로 POST 요청으로 업로드한다. 이는 연쇄적으로 'cost' 함수의 불필요한 호출을 유발하여 과도한 컴퓨팅 리소스 소비와 비용 발생을 초래할 뿐만 아니라, 데이터베이스의 오염까지 야기할 수 있다. 특히 이러한 공격은 정상적인 이벤트 트리거를 활용하기 때문에 탐지와 차단이 어려우며, 현재 사용 가능한 기존 서비스 설정만으로는 이러한 이벤트 기반 위협을 충분히 감지하고 차단하는 데 한계가 있다.

3. 관련 연구

서버리스 함수 권한 구성 및 흐름 탐지의 중요성이 증가하면서, 권한 기반 함수 경로 탐지에 대한 연구가 활발히 이루어

지고 있다. 서버리스 컴퓨팅 환경에서의 보안 위협에 대응하기 위한 연구는 크게 정적 분석, 동적 모니터링, 하드웨어 지원 기반의 세 가지 접근법으로 분류할 수 있다. 첫째, 정적 분석 기반 접근법은 서버리스 함수의 권한 구성과 호출 관계를 배포 전에 분석하여 잠재적 취약점을 식별한다. Grasp[6]는 서버리스 함수의 IAM 정책을 분석하여 함수 간 허용 가능한 호출 경로를 그래프로 시각화하고, 과도한 권한이나 잠재적 우회 경로를 탐지한다. SCIFFS[7]는 민감 데이터에 대한 라벨링 기법을 도입하여 함수 간 데이터 흐름을 제어한다. 그러나 정적 분석 기반 접근법들은 런타임에서 발생하는 동적인 함수 호출 관계를 포착하지 못하며, 실제 공격 발생 시 차단이 불가능하다는 근본적인 한계를 가진다.

둘째, 동적 모니터링 기반 접근법은 런타임에서 함수 호출과 데이터 흐름을 실시간으로 추적한다. Kalium[8]은 함수 호출 패턴을 지속적으로 학습하여 비정상적인 호출을 탐지하고, 데이터 태깅을 통해 정책 위반을 검증한다. SecLambda[9]는 로컬 함수 상태와 글로벌 애플리케이션 상태를 활용하여 제어 흐름 무결성을 보장하고 자격 증명을 보호하는 확장 가능한 프레임워크를 제시했다. Trapeze[10]는 동적 정보 흐름 제어를 도입하여 민감한 데이터의 유출을 방지하고 함수 간 안전한 데이터 흐름을 보장한다. Valve[11]는 함수 워크플로우의 연쇄 효과를 분석하여 DoS나 DoW와 같은 이벤트 기반 공격을 효과적으로 차단하는 시스템을 제안했다. 하지만 이러한 접근법들은 지속적인 모니터링으로 인한 성능 오버헤드가 크고, 복잡한 상태 관리와 설정이 요구되어 서버리스 컴퓨팅의 핵심 가치인 운영 편의성을 저해한다는 한계가 있다.

셋째, 하드웨어 지원 기반 접근법은 신뢰 실행 환경(TEE)과 같은 하드웨어 보안 기술을 활용한다. SeSeMi[12]는 신뢰 가능한 하드웨어를 활용하여 서버리스 환경과 통합 가능한 비침입적(seamless) 방식의 보안 솔루션을 제시했다. 이 접근법은 하드웨어 수준의 보안을 제공하면서도 서버리스 환경과의 자연스러운 통합을 가능케 한다. 그러나 특수한 하드웨어 지원이 필요하고 클라우드 제공자의 인프라 수정이 요구되어, 실제 서버리스 환경에 적용하기 어렵다는 한계를 가진다.

본 연구에서 제안하는 Bambda는 AWS CloudWatch의 로깅 시스템을 활용하여 실시간으로 함수 호출 관계를 모니터링하며, 자동화된 코드 삽입 기능을 통해 개발자가 복잡한 보안 설정에 시간을 들이지 않아도 된다. 특히 플러그인 방식으로 설계되어 기존 서버리스 인프라를 그대로 활용할 수 있어, 추가적인 하드웨어나 인프라 수정 없이도 즉시 적용이 가능하다. 또한, 호출 조건에 맞춘 검증을 통해 동적으로 악성 호출을 탐지하고, 차단이 가능하다. 이를 통해 Bambda는 서버리스 환경의 보안성 강화와 운영 편의성을 동시에 달성한다.

4. Bambda 설계

본 섹션에서는 서버리스 환경에서의 함수 호출 조건 기반 공격을 방지하기 위한 Bambda의 설계 원칙과 시스템 아키텍처를 소개한다. Bambda는 AWS Lambda 환경에서 플러그인 형태로 동작하며, AWS CloudWatch를 활용한 실시간 모니터링과 자동화된 코드 삽입을 통해 우회 공격을 차단한다.

4.1 설계 고려사항

R1: 실시간 서버리스 함수 호출 추적. 서버리스 애플리케이션 내 모든 함수의 호출 관계를 실시간으로 추적하고 중앙 집중화된 로그 시스템을 통해 효율적으로 관리해야 한다. 특히 서버리스 특성상 함수들이 분산되어 있고 호출 관계가 복잡하므로, 모든 함수의 호출 정보를 체계적으로 추적할 수 있는 자동화된 메커니즘이 필요하다. 따라서 AWS CloudWatch와 같은 실시간 로깅 서비스를 활용하여 애플리케이션별 단일 로그 그룹을 생성하고, 호출 정보의 분산을 최소화하는 동시에 실시간 모니터링이 가능해야 한다.

R2: 서버리스 함수 코드 삽입 자동화. 개발자의 생산성을 저해하지 않으면서도 보안 기능을 효과적으로 통합하기 위해, 기존 서버리스 함수 코드에 대한 자동화된 코드 분석 및 삽입 메커니즘이 필요하다. 함수 코드의 구조와 리소스 접근 패턴을 자동으로 분석하여 최적의 삽입 위치를 식별하고, 문법적 오류 없이 검증 로직을 통합해야 한다. 이를 통해 개발자는 복잡한 AWS 서비스 설정이나 보안 로직 구현에 시간을 들이지 않고도 안전한 서버리스 애플리케이션을 구축할 수 있어야 한다.

R3: 다단계 권한 남용 탐지. 서버리스 함수의 권한 남용과 우회적 접근을 실시간으로 탐지하고 차단하기 위한 다단계 검증 시스템이 필요하다. 먼저 함수 호출의 출처를 정확히 식별하고, 해당 함수가 접근 가능한 서비스 리스트를 기반으로 권한 범위를 검증해야 한다. 또한 이벤트 기반 호출의 경우, 이벤트 조건의 적절성과 연관 데이터의 유효성을 검사하여 악의적인 호출 패턴을 차단해야 한다. 이러한 다단계 검증은 실시간으로 이루어져야 하며, 정상적인 함수 실행에 대한 지연을 최소화해야 한다.

4.2 Bambda 아키텍처 및 워크플로우

Fig. 4에 나타난 바와 같이, Bambda는 세 가지 핵심 컴포넌트로 구성된다: Injector, Verifier, CloudWatch Logger. Bambda는 설계 고려사항을 충족하기 위해 함수 재구성 단계와 런타임 검증 단계의 두 단계로 동작한다.

첫째, 함수 재구성 단계는 서버리스 함수 코드에 보안 검증 로직을 자동으로 삽입하는 작업을 담당한다. 사용자가 제공한 서버리스 함수 코드는 Injector에 의해 분석되는데, 이 과정에서 코드 엔티티를 파싱하여 함수의 유형(예: 이벤트 기반, 직접 호출 등)과 구조(예: 인자 패턴, 호출 관계)를 판단한다. 분석된 결과를 바탕으로 Injector는 최적화된 위치에 Bambda 검증 로직을 삽입한다. 삽입된 코드는 기존 함수의 동작을 방해하지 않으면서도 호출 관계와 리소스 접근을 실시간으로 모

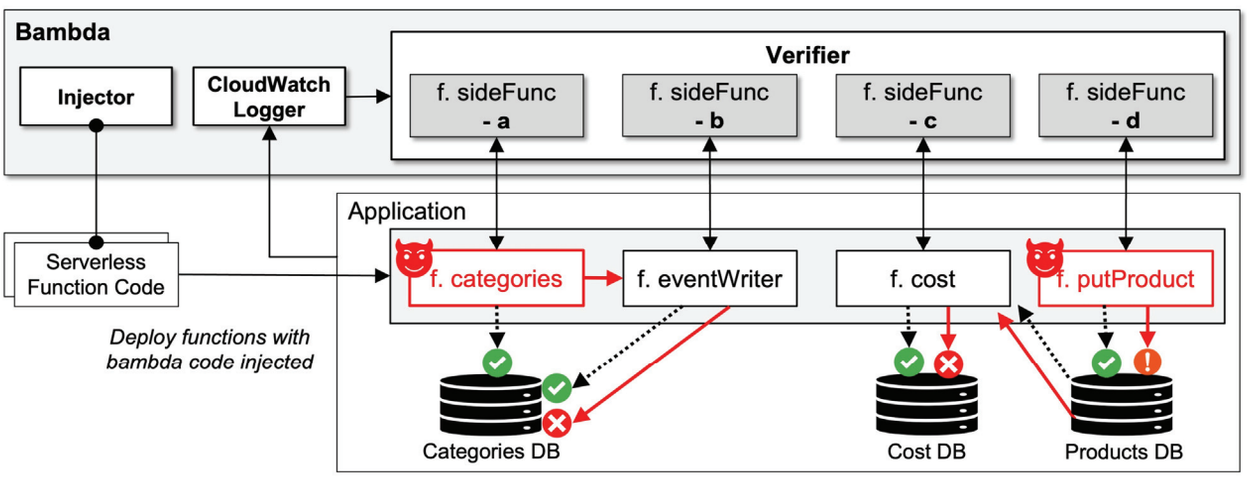


Fig. 4. Overall architecture and its workflow of Bambda with three components: (i) Injector, (ii) Verifier, (iii) CloudWatch Logger

니터링할 수 있도록 한다. 자동화된 삽입 프로세스는 사용자 개입을 최소화하며, 복잡한 AWS 서비스 설정이나 함수 간 상호작용의 세부 사항을 사용자가 직접 이해할 필요 없이 진행된다. 최종적으로 재구성된 함수 코드는 재배포 과정을 통해 AWS Lambda에 적용된다.

둘째, 런타임 검증 단계는 호출된 함수의 유효성을 확인하고 적절한 리소스 접근을 보장한다. Verifier는 함수 호출의 출처와 이벤트 조건을 두 단계에 걸쳐 검증한다. 첫 번째 단계에서는 CloudWatch Logger를 통해 호출한 함수의 이름과 고유 ID를 확인하여 호출 상태의 신뢰성을 검증한다. 두 번째 단계에서는 피호출 함수에 접근 가능한 함수 리스트를 구성하여 일치하는지 확인한다. 이벤트 기반 호출의 경우 이벤트 조건과 근원지를 평가한다. 이러한 검증 결과는 호출된 함수로 전달되며, 함수는 이를 기반으로 리소스 접근 여부를 결정한다. 이와 같은 검증 흐름은 모든 호출 관계와 리소스 접근 시도를 실시간으로 추적하여, 우회 접근과 권한 남용을 효과적으로 차단한다. 예를 들어, Fig. 4에서 'categories' 함수와 'putP-roduct' 함수 같은 주요 함수가 비정상적인 접근을 시도할 경우, Bambda는 이를 즉시 차단하며, 검증 결과는 Cl-oudWatch Logger에 기록되어 보안 감사 및 추적을 가능하게 한다.

5. Bambda 시스템 세부사항

Bambda의 핵심은 자동화된 코드 삽입과 실시간 함수 호출 검증을 결합하여, 서버리스 함수 간 우회 공격을 효과적으로 차단하고 권한 남용을 방지하는 데 있다. 이를 위해 본 시스템은 서버리스 함수의 자동 분석 및 재구성과 함께 중앙집중식 모니터링을 통한 실시간 검증을 제안한다.

5.1 코드 삽입 자동화 기반 함수 재구성

Bambda의 코드 재구성은 함수 구조 분석과 검증 코드 삽입이라는 두 가지 핵심 단계로 구성된다. 함수 구조 분석 단계는 입력된 서버리스 함수 코드를 파싱하여 AWS 서비스와 관련된 주요 엔티티들을 식별한다. 이 과정에서 함수의 진입점과 종료점을 정확히 파악하고, 리소스 접근 패턴을 분석하여 보안 검증의 수준을 결정한다. Fig. 5(A)에서 예시로 든 'eventWriter' 함수의 경우, 'DynamoDB'와 'module.exports' 같은 핵심 엔티티들이 추출되며, 이를 통해 함수의 동작 특성과 구조적 속성이 파악된다. 특히 'DynamoDB' 엔티티가 발견되면 해당 함수는 데이터베이스 작업 함수로 분류되어 권한 검증, 데이터 유효성 검사, 접근 로그 기록과 같은 추가적인 보안 단계가 요구된다. 반면 AWS 서비스 관련 엔티티가 없는 경우에는 단순 이벤트 처리 함수로 분류되어 기본적인 호출 추적만이 수행된다.

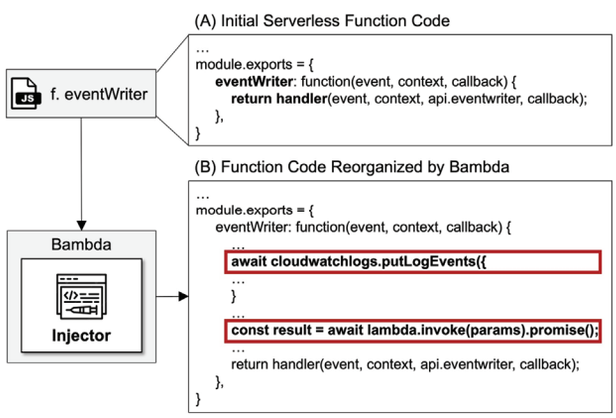


Fig. 5. The example of injector for recontructing lambda code

검증 코드 삽입 단계는 분석된 함수의 특성에 따라 차별화된 보안 로직을 통합한다. 이 과정에서는 함수의 원래 동작을 방해하지 않으면서도 효과적인 보안 검증이 가능하도록 코드 삽입 위치와 방식을 최적화한다. Fig. 5(B)와 같이, 데이터베이스 작업 함수의 경우 함수 실행의 주요 시점마다 보안 검증 로직이 삽입된다. 예를 들어 'eventWriter' 함수에 DynamoDB

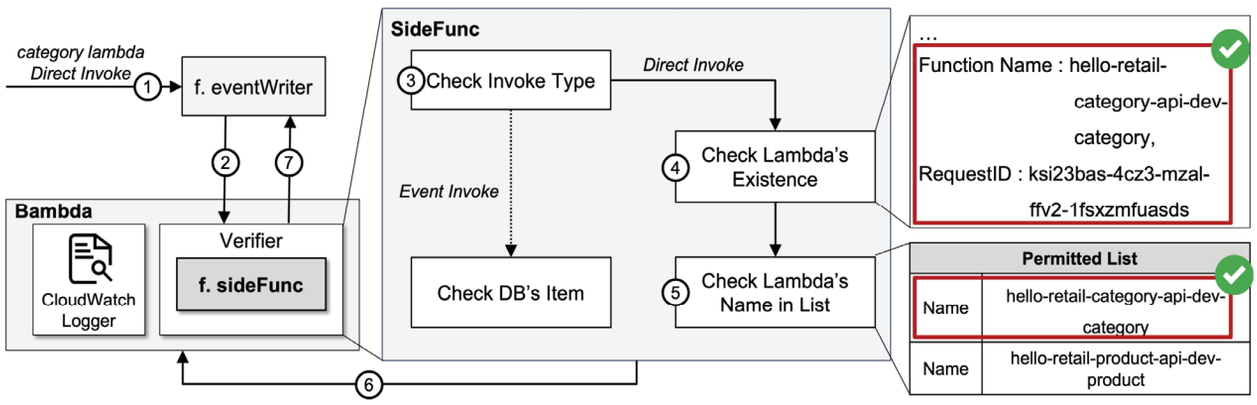


Fig. 6. The example of verifier for validating lambda invocation

접근이 감지되면, 함수 진입 시점에 호출 적합성 검증 로직이 배치되어 함수의 호출 유형에 맞춰 검증 로직이 수행되고, 실행 과정에서 발생하는 모든 이벤트에 대해 실시간 모니터링이 가능하도록 추적 코드가 삽입된다. 이러한 검증 로직은 비동기 처리 방식으로 구현되어 함수의 기존 실행 흐름에 미치는 영향을 최소화하며, 권한 위반이나 비정상적인 접근이 감지될 경우 즉각적인 차단과 함께 상세한 위반 정보를 기록한다.

5.2 로깅 서비스 기반 함수 호출 추적

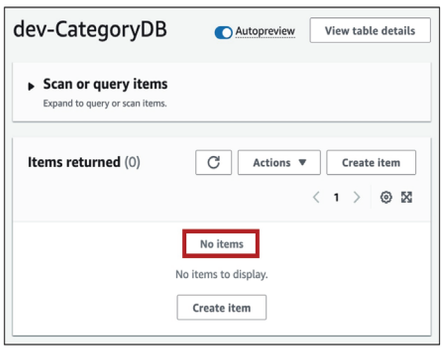
AWS Lambda의 기본 로깅 방식은 함수가 완전히 종료된 이후에만 로그를 기록하는 구조적 한계를 가진다. 이는 함수 실행 중간에 발생하는 이벤트나 상태 변화를 실시간으로 확인할 수 없게 만들며, 여러 함수가 연속적으로 호출되는 상황에서 전체 실행 흐름을 파악하기 어렵게 만든다. 이를 해결하기 위해 Bambda는 각 서버리스 애플리케이션마다 독립된 하나의 로그 그룹을 생성하여 모든 함수의 호출 정보를 중앙 집중적으로 관리한다. 예를 들어 'Hello Retail!'이 배포되면 'Hello-Retail-Log-Group'이라는 전용 로그 그룹이 생성하고, 이 그룹 내에서 모든 함수의 호출 정보가 통합한다.

각 로그 그룹 내에는 함수별로 구분된 로그 스트림이 생성되며, 여기에는 함수의 이름과 고유 ID를 포함한 상세한 실행 정보가 기록된다. 이 로그 스트림은 함수 실행의 전체 생명주기를 추적하는데, 함수 초기화 시점부터 종료 시점까지 발생하는 모든 중요 이벤트를 포함한다. 예를 들어 'eventWriter' 함수가 호출되면 해당 함수의 이름과 고유 ID를 포함한 새로운 로그 스트림이 즉시 생성되며, 함수의 호출 시간, 호출 유형(POST/GET), 데이터베이스 작업 유형 등이 시계열 순서로 기록된다. 특히 데이터베이스 작업의 경우, 쿼리 실행 시점, 영향받은 레코드 수, 작업 소요 시간과 같은 상세 정보까지 포함하여 비정상적인 데이터베이스 접근을 탐지할 수 있다. 또한 함수 간 호출이 발생할 때마다 호출자와 피호출자의 정보를 연결하여 기록함으로써, 복잡한 함수 호출 체인도 명확하게 추적할 수 있다. 이러한 실시간 로깅 구조는 분산된 서버리스 환경에서도 함수 간의 호출 관계와 실행 흐름을 즉각적으로 파악할 수 있게 하며, 특히 'putProduct' 함수에서 'cost' 함수로 이어지는 것과 같은 연쇄적인 함수 호출이 발생하는 경우에도 전체 실행 흐름을 누락 없이 추적할 수 있다.

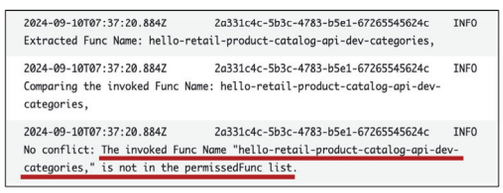
5.3 로그 기반 우회 접근 검증

Bambda의 함수 호출 검증은 직접 호출과 이벤트 기반 호출에 대해 차별화된 검증 프로세스를 적용한다. 직접 호출 검증의 경우, 호출된 함수의 실존 여부 확인을 통해 애플리케이션의 의도된 호출인지를 우선적으로 검증한다. 이어서 실시간 로그를 통해 호출의 시간적 유효성과 권한 적합성을 평가하는데, 호출 함수의 고유 ID와 이름을 확인하여 최근 5초 이내의 실행 기록을 검증하고 이를 사전 정의된 호출 허용 리스트와 대조한다. 이벤트 기반 호출의 경우에는 이벤트의 근원지와 함께 전달되는 데이터의 유효성을 중점적으로 검증한다. 예를 들어, 데이터베이스 업데이트와 같은 이벤트에 대해서는 전달된 데이터가 예상된 스키마와 일치하는지, 그리고 과도한 빈도의 호출, 무의미한 반복 데이터, 비정상적인 시간과 같은 패턴이 없는지를 상세히 검사한다. 이러한 호출 유형 별 차별화된 검증 프로세스를 통해 부적절한 함수 호출을 판단하고, 민감한 정보 접근을 차단할 수 있다.

Fig. 6에서 보여지는 검증 프로세스의 실제 흐름은 다음과 같다. 먼저 'category' 함수가 'eventWriter'함수를 호출하면(①), 검증을 위해 Verifier로 전달되어 'si-deFunc'가 실행된다(②). 'sideFunc'는 우선적으로 호출 유형을 확인하는데(③), 직접 호출인 경우 Lambda 함수의 실제 존재 여부를 검증한다. 이는 애플리케이션의 기능적 의도로 발생한 정상적인 호출인지 확인하는 필수적인 단계로, 함수의 이름과 RequestID (예: ksi23bas-4cz3-mza-l-flv2-1fsxzmfusds)를 통해 'hello-retail-category-api-dev-category'와 같은 실제 함수인지 확인한다(④). 함수가 확인되면 해당 함수가 허용 리스트에 포함되어 있는지 검사하는데(⑤), 이는 관리자가 사전에 구성한 승인된 함수 목록과 대조된다. 이러한 허용 리스트 기반 검증은 의도적인 연쇄 호출이 필요한 경우에도 보안을 유지할 수 있게 한다. 반면 이벤트 기반 호출로 확인된 경우에는 데이터베



(A) Status of the Categories DB after the attack



(B) The Result of the Bypass Attack Prevention

Fig. 7. The results of evasion attacks in a serverless application with the bambda system

이스 아이템의 속성과 구조를 검사하여 테이블 스키마와의 일치 여부를 확인하고, 무의미한 데이터의 반복적 삽입을 차단한다. 모든 검증 과정의 결과는 로그로 기록되어(⑥) 실시간 모니터링이 가능하며, 이 정보는 다시 Verifier로 전달되어(⑦) 최종적인 호출 허용 여부를 결정하는 데 사용된다.

6. 우회 공격 차단 성능 분석

6.1 실험 목적 및 방법

본 연구의 실험은 서버리스 환경에서 Bambda의 우회 접근 차단 성능을 종합적으로 평가하기 위해 설계되었다. 서버리스 플랫폼으로 AWS Lambda[16]를 사용하였으며, 배포에는 Serverless Framework[15] 4.25 버전을 통해 진행하였다. 테스트 애플리케이션으로는 제품 등록, 검색, 구매와 같은 전자상거래의 핵심 기능들이 9개의 독립적인 서버리스 함수로 구현된 Hello Retail![13]을 선정하였다. 이 애플리케이션은 함수 간 복잡한 호출 관계와 데이터베이스 상호작용을 포함하고 있어, 서버리스 환경의 우회 공격 시나리오를 테스트하기에 적합한 특성을 가진다.

Bambda의 실현 가능성과 효과성을 검증하기 위해 Node.js와 YAML을 활용하여 약 500줄의 코드로 시스템을 구현하였다. 현재 Bambda는 AWS 플랫폼에서만 동작하도록 설계되어 있다. 본 실험에서는 Fig. 3에서 제시된 두 가지 우회 공격 시나리오를 기반으로, 권한이 남용된 함수를 통한 리소스 접근 시도와 이벤트 호출 조건을 악용한 연쇄 호출 공격을 중점적으로 테스트하였으며, 각 실험은 통계적 유의성을 확보하기 위해 충분한 횟수로 반복 수행하였다.

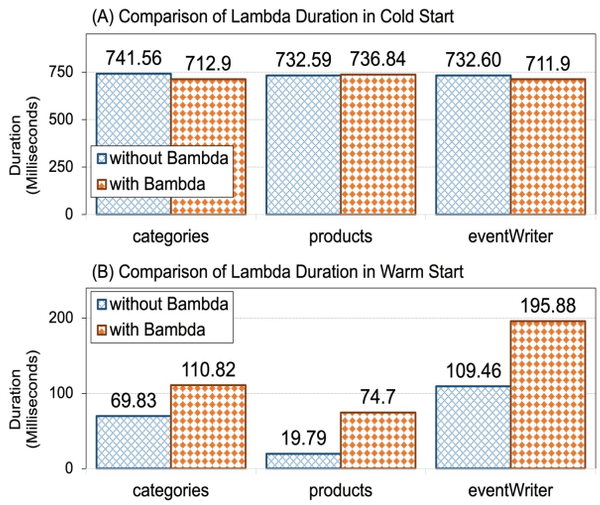


Fig. 8. The results of execution in serverless application with and without the bambda system applied

6.2 공격 탐지 기능적 정확도

본 실험은 Bambda가 서버리스 환경에서 발생하는 비정상적인 함수 호출과 권한 우회 시도를 얼마나 정확하게 탐지하고 차단할 수 있는지를 검증하기 위해 설계되었다. 실험을 위해 'categories' 함수를 대상으로 AWS Lambda의 'aws-sdk' 메소드를 활용하여 'eventWriter' 함수를 호출할 수 있는 권한을 부여하고, 이를 통해 비정상적인 데이터베이스 접근을 시도하는 공격 환경을 구축하였다.

Fig. 7을 통해 Bambda의 공격 차단 효과를 두 가지 측면에서 확인할 수 있다. 먼저 Fig. 7(A)는 공격 시도 직후의 Categories DB 상태를 보여주는데, "No items"라는 메시지와 함께 데이터베이스가 비어있는 것을 확인할 수 있다. 이는 Bambda가 비정상적인 데이터 삽입 시도를 성공적으로 차단했음을 입증한다. Fig. 7(B)는 Bambda의 검증 로그로, "hello-retail-product-catalog-api-dev-categories" 함수가 "No used func Name"으로 표시된 것을 보여준다. 이는 해당 함수가 Bambda의 허용 함수 리스트에 포함되지 않아 접근이 거부되었음을 의미한다.

이러한 결과는 Bambda의 실시간 함수 호출 검증 메커니즘이 효과적으로 작동하여 허용되지 않은 함수의 접근을 즉각적으로 탐지하고 차단할 수 있음을 보여준다. 또한, Categories DB의 빈 상태는 Bambda의 차단 조치가 단순히 로깅 수준에 그치지 않고 실제 데이터베이스 보호에도 효과적임을 증명한다. 이는 Bambda가 서버리스 환경에서 발생할 수 있는 권한 우회 공격에 대해 실질적인 방어 능력을 갖추고 있고, 결과에 대한 로깅을 통해 지속적인 보안성 강화가 가능함을 시사한다.

6.3 플러그인 Lambda 적용의 성능 영향 분석

본 실험에서는 Bambda의 통합이 서버리스 함수의 실행 성

능에 미치는 영향을 정량적으로 평가하였다. 이를 위해 함수의 두 가지 주요 실행 상태인 Cold Start[17]와 Warm Start [18] 환경에서의 지연을 측정하고 비교 분석하였다.

Fig. 8(A)에서 보이듯이, ‘categories’, ‘products’, ‘event Writer’ 함수를 대상으로 Cold Start 환경에서의 실행 시간을 측정한 결과, 세 함수 모두 710-740ms 범위의 실행 시간을 보이며 Bambda 통합으로 인한 실행 지연은 20-30ms 수준으로 제한적인 것으로 나타났다. 특히 ‘categories’와 ‘event Writer’ 함수에서는 오히려 실행 시간이 소폭 감소하였다. 이는 Cold Start 상황에서 함수 초기화와 컨테이너 프로비저닝에 소요되는 시간이 지배적 요인으로 작용하여, Bambda의 검증 로직이 전체 실행 시간에 미치는 영향이 상대적으로 미미한 것으로 해석된다.

반면, Fig. 8(B)에서 제시된 Warm Start 환경의 측정 결과는 상당한 성능 차이를 보여준다. 최소 영향을 받은 ‘categories’ 함수의 경우 약 41ms의 실행 시간 증가가 관찰되었으며, 가장 큰 영향을 받은 ‘eventWriter’ 함수는 109.46ms에서 195.88ms로, 약 86.42ms(79.0%)의 실행 시간 증가를 나타냈다. 이러한 현저한 차이는 Warm Start 환경에서 함수 초기화 부하가 제거되어 Bambda의 실시간 검증 메커니즘이 실행 시간에 직접적인 영향을 미치기 때문으로 분석된다. 특히 ‘eventWriter’ 함수의 경우, 데이터베이스 작업과 연계된 다단계 검증 프로세스가 요구되어 상대적으로 큰 성능 저하가 발생한 것으로 판단된다.

이러한 실험 결과는 보안 강화와 성능 최적화 사이의 필연적인 트레이드오프 관계를 보인다. 그러나 Bambda가 제공하는 실시간 우회 공격 차단과 포괄적인 보안 모니터링의 효용성을 고려할 때, 관찰된 성능 저하는 수용 가능한 수준으로 평가된다. 특히 실시간 함수 호출 검증과 세분화된 데이터베이스 작업 모니터링을 통한 다각적 우회 공격 방어 능력은, 서버리스 컴퓨팅 환경에서 요구되는 보안 요건을 충족하는 데 실질적인 기여를 할 것으로 기대된다.

7. 결 론

본 연구는 실제 서버리스 컴퓨팅 환경에서 서버리스 함수의 호출 조건 및 IAM 구성을 악용하는 위협을 감지하고 차단하기 위한 동적 프레임워크인 Bambda를 제안하였다. Bambda는 복잡하고 동적인 서버리스 환경에 적합하도록 설계되었으며, 기존의 정적 분석 방식이나 과도한 사용자의 개입을 요구하는 방법과 차별화된다. Bambda는 우회적인 함수 호출을 이용한 접근 차단뿐만 아니라, 차단된 함수 워크플로우를 로그로 기록하여 개발자가 추후 위협을 보완할 수 있도록 지원한다. 실험을 통해 Bambda는 우회 공격에 대한 효과적인 차단 능력을 입증하였으며, 실행 시간 측면에서도 미비한 수준의 성능 오버헤드는 보안 강화 효과를 고려할 때 수용 가능한 수준으로 평가된다. 본 연구는 호출 조건 기반 위협 차단이 취약한 서버리스 함수의 보안성을 강화하기 위한 귀중한 참조 구현을 제공하며, 서버리스 환경에서의 보안 관리에 대한 새로운 방향을 제시한다.

향후 연구에서는 API Gateway를 통해 유입되는 악의적인 함수 호출에 대한 보안 취약점을 보완하고자 한다. 현재 프레임워크는 API Gateway를 통한 외부 공격을 효과적으로 차단하는 데 한계가 있으므로, 이를 개선하여 시스템의 보안성을 강화하고 전체적인 보안 아키텍처를 최적화하는 방안을 확장 연구할 예정이다.

References

[1] DATADOG, The state of serverless [Internet], https://www.datadoghq.com/state-of-serverless

[2] S. Eismann et al., “Serverless applications: Why, when, and how?,” *IEEE Software*, Vol.38, No.1, pp.32-39, 2020.

[3] H. B. Hassan, S. A. Barakat, and Q. I. Sarhan, “Survey on serverless computing,” *Journal of Cloud Computing*, Vol.10, pp.1-29, 2021.

[4] Y. Li, Y. Lin, Y. Wang, K. Ye, and C. Xu, “Serverless computing: state-of-the-art, challenges and opportunities,” *IEEE Transactions on Services Computing*, Vol.16, No.2, pp.1522-1539, 2022.

[5] DivvyCloud, 2020 Cloud Misconfiguration Report [Internet], https://www.bankinfosecurity.com/whitepapers/2020-cloud-misconfigurations-report-w-6009

[6] I. Polinsky, P. Datta, A. Bates, and W. Enck, “GRASP: Hardening Serverless Applications through Graph Reachability Analysis of Security Policies,” *Proceedings of the ACM on Web Conference*, pp.1644-1655, 2024.

[7] I. Polinsky, P. Datta, A. Bates, and W. Enck, “SCIFFS: Enabling secure third-party security analytics using serverless computing,” *Proceedings of the 26th ACM Symposium on Access Control Models and Technologies*, pp.175-186, 2021.

[8] D. S. Jegan, L. Wang, S. Bhagat, and M. Swift, “Guarding serverless applications with Kalium,” *USENIX Security Symposium*, pp.4087-4104, 2023.

[9] D. S. Jegan, L. Wang, S. Bhagat, T. Ristenpart, and M. Swift, “Guarding serverless applications with seclambda,” *arXiv preprint*, arXiv:2011.05322, 2020.

[10] K. Alpernas et al., “Secure serverless computing using dynamic information flow control,” *Proceedings of the ACM on Programming Languages*, Vol.2(OOPSLA), pp.1-26, 2018.

[11] P. Datta, P. Kumar, T. Morris, M. Grace, A. Rahmati, and A. Bates, "Valve: Securing function workflows on serverless computing platforms," *Proceedings of The Web Conference*, pp.939-950, 2020.
[12] G. Hu, Y. Wu, G. Chen, T. T. A. Dinh, and B. C. Ooi, "SeSeMI: Secure Serverless Model Inference on Sensitive Data," *arXiv preprint*, arXiv:2412.11640, 2024.
[13] SimonEismann, Hello Retail!s Application [Internet], https://github.com/SimonEismann/hello-retail
[14] C. Ngo, P. Wang, T. Tran, and S. Chung, "Serverless computing architecture security and quality analysis for back-end development," *Journal of The Colloquium for Information Systems Security Education*, Vol.7, No.1, p.8, 2020.
[15] Serverless Framework, Easy Serverless Apps on AWS Lambda [Internet], https://www.serverless.com
[16] AWS, AWS Lambda [Internet], https://aws.amazon.com/ko/lambda/
[17] S. Ristov, C. Hollaus, and M. Hautz, "Colder than the warm start and warmer than the cold start! experience the spawn start in faas providers," *Proceedings of the 2022 Workshop on Advanced tools, programming languages, and PLatforms for Implementing and Evaluating algorithms for Distributed systems*, pp.35-39, 2022.
[18] J. Carreira, S. Kohli, R. Bruno, and P. Fonseca, "From warm to hot starts: Leveraging runtimes for the serverless era," *Proceedings of the Workshop on Hot Topics in Operating Systems*, pp.58-64, 2021.
[19] AWS, Amazon CloudWatch [Internet], https://aws.amazon.com/cloudwatch/
[20] AWS, Amazon DynamoDB [Internet], https://aws.amazon.com/dynamodb/
[21] J. Yang, Cyber attackers constantly change their tactics… Cloud hacking persists [Internet], ZDNET Korea. https://zdnet.co.kr/view/?no=20240808150629
[22] D. Kelly, F. G. Glavin, and E. Barrett, "Denial of wallet—defining a looming threat to serverless computing," *Journal of Information Security and Applications*, Vol.60, 102843, 2021.
[23] J. Xiong, M. Wei, Z. Lu, and Y. Liu, "Warmonger: inflicting denial-of-service via serverless functions in the cloud," *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, pp.955-969, 2021.
[24] sysdig, Sysdig 2024 Cloud-Native Security and Usage Report [Internet], https://sysdig.com/2024-cloud-native-security-and-usage-report/
[25] Google Cloud, Cloud Run Functions [Internet], https://cloud.google.com/functions
[26] Azure, Azure Functions [Internet], https://azure.microsoft.com/ko-kr/products/functions
[27] AWS, Amazon S3 [Internet], https://aws.amazon.com/ko/s3/
[28] C. Shin and S. Lee, "Bambda: A Framework for Preventing Evasion Attacks in Serverless Environments," *Proceedings of the Annual Conference of KIPS*, pp.76-77, 2024.

신 창 희
https://orcid.org/0009-0001-3229-2664
e-mail : changhee9149@inu.ac.kr
2022년~현재 인천대학교 컴퓨터공학부(학사)
관심분야 : 클라우드 보안, 네트워크 보안

이 승 수
https://orcid.org/0000-0002-6883-1869
e-mail : seungsoo@inu.ac.kr
2014년, 숭실대학교 컴퓨터학부 학사
2016년 KAIST 정보보호대학원 석사
2020년 KAIST 정보보호대학원 박사
2021년 9월~현재 인천대학교 컴퓨터공학부 조교수
관심분야 : 클라우드 보안, 네트워크 보안