Computer Security Symposium 2021
26 - 29 October 2021

マニフェスト署名検証に基づくKubernetesリソースのインテグリティ保護

工藤 瑠璃子[1,a)]　北原 啓州[1]　ガジャーナ クガムーテ[1]　渡邊 裕治[1]

概要：政府や金融機関向けの高い保護レベルが要求される環境では, クラウド上のインテグリティ維持は重要な要件であり, 米国のセキュリティ基準である NIST SP 800-53 では, 電子署名の無いリソース作成は防がなければいけないと定められている. クラウドのプラットフォームである Kubernetes では, クラスタやアプリケーションの設定は Kubernetes リソースで定義される. この Kubernetes リソースは YAML マニフェストで表現される API リソースであるため, マニフェストに署名をつけて, Kubernetes API の呼び出し時にその署名を検証すれば, 強力なクラウドのインテグリティ保護になる. このような検証処理は admission controller という機構を用いることで差し込むことができるが, 実際にこの仕組みを実クラスタ上で実現する際には解決しなければならない技術課題が存在する. 本稿では, それらの課題を解き, admission controller における署名検証に基づいた Kubernetes リソースのインテグリティ保護手法を提案し, 実クラスタ上の評価実験から手法の有効性を示す.

キーワード：インテグリティ, クラウド, Kubernetes, 署名

Integrity Protection for Kubernetes Resource based on Manifest Signature Verification

Ruriko Kudo[1,a)]　Hirokuni Kitahara[1]　Kugamoorthy Gajananan[1]　Yuji Watanabe[1]

Abstract: Integrity of the cloud is the most important requirement for mission-critical enterprise workloads. NIST SP 800-53 states that information systems must prevent the installation of any components that have not been verified digitally with a signed certificate that is recognized and approved by the organization's information system. On a Kubernetes cluster, the admission controller can control requests for application installation, and it would be a powerful protection tool if it could control requests for Kubernetes resources based on signature verification. However, there are various technical challenges when it comes to verifying the signature for a Kubernetes resource at the admission controller because a signed resource is rewritten automatically by internal cluster work and many requests that include internal mutation without a signature are generated. In this work, we propose an approach to protect the integrity of a Kubernetes resource with signature verification at the admission controller.

Keywords: Integrity, Cloud, Kubernetes, Signature

1. はじめに

政府や金融機関の高い保護レベルが要求される環境では, クラウド上のセキュリティ, コンプライアンス, インテグリティ保護は重要な要件である. 政府が導入するクラウドサービスのセキュリティ管理基準である NIST SP 800-53 [9] では,「組織によって受け入れられ承認された証明書を用いてデジタル署名されていることが検証されていないコンポーネントのインストールを防止する」ことが要件の一つに挙げられている.

[1] IBM 東京基礎研究所
IBM Research - Tokyo
[a)] rurikudo@ibm.com

クラウドのプラットフォームである Kubernetes [7] では, クラスタやアプリケーションの設定は Kubernetes リソースで定義される. この Kubernetes リソースは YAML マニフェスト [11] で表現される API リソースであるため, マニフェストに署名をつけて, Kubernetes API の呼び出し時にその署名を検証すれば, 強力なクラウドのインテグリティ保護になる. 上記のような処理は, Kubernetes リソースに対する操作 (Create, Update, Delete) のためのリクエストを制御できる admission controller [8] という機構を用いて実現できるため, 本稿ではこの機構を使用して電子署名に基づいた Kubernetes 上のアプリケーションのインテグリティ保護を目指す.

Kubernetes 上のアプリケーションは, アプリケーションのコンフィグレーションを定義する Deployment, Configmap といった Kubernetes リソースとコンテナイメージで構成される. コンテナイメージに関しては, すでに admission controller で署名検証を行い, 不正なコンテナイメージがクラスタ上にインストールされることを防ぐ技術 [12–14] が開発されているが, Kubernetes リソースに対してはそのような技術は実現されていない. アプリケーション全体を保護するためには Kubernetes リソースの保護が不可欠であるため, 本稿では Kubernetes リソースの署名による保護を実現する.

しかしながら, admission controller において署名検証に基づいた Kubernetes リソースの保護を実クラスタ上で実現するのは容易ではなく, 様々な課題存在する. まず, admission controller が受け取った Kubernetes リソースに対して署名検証を直接行うことができないという点である. Kubernetes リソースは YAML マニフェストで表現される. Kubernetes リソースを署名により保護するためには, YAML 全体をメッセージとして署名を作成し, admission controller では, Kubernetes リソースとそれに対する署名を含む admission リクエストを受け取るが, admission リクエスト内の Kubernetes リソースは, クラスター内部の動作によって署名作成時のものとは違う内容に書き換わっている. つまり, admission controller で署名検証を行うには, admission リクエスト内に含まれる kubernetes リソースをそのまま検証するのではなく, クラスタの内部的な動作を考慮した上で, 署名作成時の Kubernetes リソースに基づいて検証しなければならない. また, クラスター内部の動作によって書き変えられる部分は様々でリソースによって異なるため, 事前に変更される箇所を全て把握しホワイトリストするような手法は現実的ではない. コンテナイメージは一度 build されると不変であり, クラスタの内部的な動作でコンテナイメージの中身を変更されることはないため, このような問題は起こらないが, Kubernetes リソースでは, admission リクエストに含まれる署名検証対象のオブジェクトと, 署名作成時のオブジェクトに差異を解消した上で署名検証する必要がある.

2 つ目に, admission controller が受け取るリクエスト全てに署名がつけられる訳ではないという問題がある. Kubernetes 上ではクラスタの正常な運用の一環として様々な内部処理が行われ, それに関連した様々な admission リクエストが発生する. 当然, この内部的な動作に伴って発生するリクエストに署名をつけることは難しい. クラスタ内部から発生する署名のない admission リクエストが, 署名検証ができないために admission controller でブロックされてしまうとクラスタの正常の動作に影響を与えてしまうため, クラスタ外部から発生するリクエストを署名検証の対象にすべきであるが, そのためには, admissioni controller が受け取る大量のリクエストの中からクラスタの内部動作に起因するものを適切に判別できなければならない.

本稿では, admission controller において署名を用いて Kubernetes リソースのインテグリティを保護する手法を提案する. 本稿の貢献は以下の通りである.

(1) Admission リクエストに含まれる Kubernetes リソースの署名を, 署名生成時のマニフェストに基づいて検証する admission controller の実現
(2) 上記の Admission controller の実現に必要不可欠なクラスター内部の動作に基づく変更を署名検証対象から効率的に除外するための柔軟なプロファイルの構築とそのプロファイルに基づく admission リクエストの制御の実現

提案手法では, Kubernetes の DryRun [10] 機能を利用して, admission リクエストに含まれる Kubernetes リソースと署名生成時の YAML マニフェストの整合性を求め, 適切に署名を検証できるようにした. また, クラスタの通常動作に影響を与えず提案手法を機能させるために, admission リクエストを複数の観点から柔軟に定義できるプロファイルフレームワークを作成した. Admission リクエストのフィルタリングは, admission controller において Kubernetes リソースの署名検証を実現するために不可欠であり, このプロファイルによって admission controller では, 受け取ったリソースの作成や変更リクエストが内部的に発生しているものなのかを判断可能になる.

本稿では, まず提案手法の前提となる技術である Kubernetes と Admission controller についての説明した上で, 本稿で解こうとしている課題について示す. 続いて, 提案手法の詳細について示したのち, 提案手法の有効性を確認するために実際のアプリケーションと実 Kubernetes クラスタを用いて行った評価について述べる. 最後にまとめを行う.

2. 関連技術と課題

2.1 Kubernetes

Kubernetes は, コンテナ化したアプリケーションのデプロイ, スケーリング, および管理を行うためのオープ

ンソースのコンテナオーケストレーションシステムである．Kubernetes 上で稼働する全てのアプリケーションは Kubernetes リソースで構成される．Kubernetes リソースは, オブジェクトの基本的な情報 (e.g. Name, Kind) と共に, 望ましい状態を記述したオブジェクトの仕様を YAML マニフェストに記述することで作成できる．例えば, クラスタ上で稼働するアプリケーションを定義するオブジェクトである Deployment を用いてアプリケーションをインストールする場合, ユーザは図 1 のような YAML マニフェストを用意する．

Kubernetes リソースをクラスタ上で操作（作成, 更新, 削除など）するには, Kubernetes の API を使用する必要があり, クラスタに Kubernetes リソースを作成する場合には, 図 1 のような YAML マニフェストを kubectl コマンドラインで適用する．クラスタ側では, Kubernetes の API サーバーがリクエストを受信し, 認証・認可を行う．また, Kubernetes は柔軟に構成することが可能で, Kubernetes API サーバのリクエスト制御を司る admission controller は, その拡張機能の 1 つである．Admission controller が受け取るオブジェクトである admission リクエストには, Kubernetes リソースのオブジェクトやユーザー名などの様々な情報が含まれており, これらの情報を基にリクエストを様々な側面から制御可能である．

2.2 Admission リクエストの調査と技術課題

2.2.1 署名作成時のオブジェクトと admission リクエストに含まれるオブジェクト間の差異

本節では, Kubernete リソースに署名をつける際のメッセージと, 実際に admission controller が受け取る admission リクエスト内のメッセージ間に生じる差異について説明する．図 2 は, Kubernetes リソース作成時のフローである．

ユーザが Kubernetes リソースをクラスタに適用すると, admission リクエストが admission controller に伝搬される．図 1 に示している Deployment はクラスタ適用前の Kubernetes リソースの一例である．Admission controller で Kubernetes リソースに対する署名検証を行う場合, ユーザはこの図 1 の YAML マニフェストをメッセージとする署名を作成することになる．しかし, admission controller が受け取るリクエストに含まれている Kubernetes リソースのオブジェクトは, Kubernetes の内部的な動作によって新しいフィールドや値が挿入されたものになっており, 署名作成時とは違う内容に書き換えられている．実際に, 図 1

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: test-app
  labels:
    app: test-app
spec:
  replicas: 1
  selector:
    matchLabels:
      app: test-app
  template:
    metadata:
      labels:
        app: test-app
    spec:
      imagePullSecrets:
        - name: regkey
      containers:
        - name: test-app
          image: test-image:0.0.1
          resources:
            requests:
              cpu: 100m
          command:
            - sleep
            - infinity
```

図 1 Example of Kubernetes resource (Deployment).

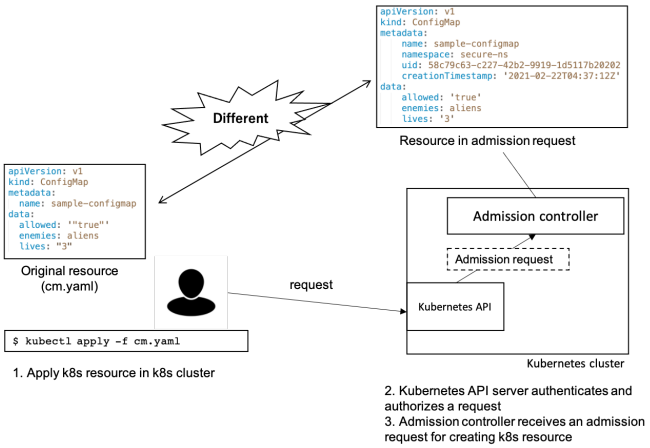

図 2 Process flow for k8s resource creation.

```yaml
kind: Deployment
apiVersion: apps/v1
metadata:
  name: test-app
  namespace: secure-ns
  uid: 9ceccc8c-4765-4266-b057-774866417734
  generation: 1
  creationTimestamp: '2021-02-22T04:33:36Z'
  labels:
    app: test-app
spec:
  replicas: 1
  selector:
    matchLabels:
      app: test-app
  template:
    metadata:
      creationTimestamp: null
      labels:
        app: test-app
    spec:
      containers:
      - name: test-app
        image: 'test-image:0.0.1'
        command:
          - sleep
          - infinity
        resources:
          requests:
            cpu: 100m
        terminationMessagePath: /dev/termination-log
        terminationMessagePolicy: File
        imagePullPolicy: IfNotPresent
      restartPolicy: Always
      terminationGracePeriodSeconds: 30
      dnsPolicy: ClusterFirst
      securityContext: {}
      imagePullSecrets:
        - name: regkey
      schedulerName: default-scheduler
  strategy:
    type: RollingUpdate
    rollingUpdate:
      maxUnavailable: 25%
      maxSurge: 25%
  revisionHistoryLimit: 10
  progressDeadlineSeconds: 600
status: {}
```

図 3 Deployment in admission request.

の Deployment を作成すると, admission リクエストに含まれているメッセージは, 図 3 のように書き換わる. また, このクラスタによって追加されるフィールドや値は, リソースによって異なる. Admission controller で Kubernetes リソースの署名検証を実現するためには, admission リクエストに含まれるリソースをそのまま署名検証するのではなく, クラスターの内部的な動作によって変更された部分を考慮した上で適切に署名検証しなければならない.

2.2.2 定常状態における admission リクエストの分布

本稿では, admission controller で Kubernetes リソースの署名検証を実現のために, admission controller がどのような admission リクエストを受け取るのか調査した. 調査データは, 2020 年 10 月 30 日から 2020 年 11 月 4 日の期間に OpenShift Container Platform [6] 上で起こった全てのリクエストである. このクラスタはインストール時の状態にほぼ等しいクラスターであるため, この期間に起きたリクエストはプラットフォームが稼働するために内部的起こしているリクエストであると言える.

図 4 は 3 時間ごとの admission controller が受け取るリクエスト数を示している. 図 4 から分かる通り, Kubernetes クラスタ上では外部からの操作がない定常状態においても, 3 時間ごとに 160,000 件以上の大量のリクエストが発生していることが分かる. Admission controller で kubernetes リソースの署名検証を実現するためには, この大量の内部リクエストを適切に処理することが不可欠である. 表 1 は, リクエストがどの Namespace でどの Kind に対して, どの ServiceAccount によって起きたかを調査した結果である. リクエスト数の多かったものを表に示している. 表からわかるように, 定常状態で起きているリクエストは様々なリソースに対して, 様々な ServiceAccount から起こされていることが分かる. また, 様々な Namespace でリクエストが起きているため, アプリケーションを署名検証により保護しながらクラスタにインストールするためには, このような定常的に起きるリクエストをブロックしないようにフィルタリングしつつ適切に処理する必要がある.

3. 提案手法

3.1 提案システムの概要

本章では, 提案手法の概要を説明する. 2.1 章で述べたように, Kubernetes には「Admission Controller」という拡張機能があり, Kubernetes リソースの作成や変更の際に, リクエストに任意の制御を加えることができる. 本稿ではこの機能を利用し, 以下のような Kubernetes リソースの署名検証に基づいたインテグリティ保護手法を提案する.

図 5 に提案手法の概要を示す. まず, アプリケーションのオーナーは, YAML マニフェスト (A) に書かれた Kubernetes リソースの署名を生成し, 署名 (B) とメッセージ (C) をファイルに添付する. メッセージは YAML マニフェスト全体である. 署名付き YAML ファイル (D) がクラスタにインストールされると, Kubernetes API は admission リクエストを作成する. Admission controller は，Kubernetes リソース (E) を含む admission リクエストを受け取り，署名検証サーバへ伝搬する. サーバでは, 提案手法のプロファイルフレームワークに基づいて admission リクエストのフィルタリングと署名検証を行い，最終的にリクエストを許可するかどうかを決定する. 署名検証およびプロファイルの詳細については, それぞれ 3.2 章, 3.3 章で説明する.

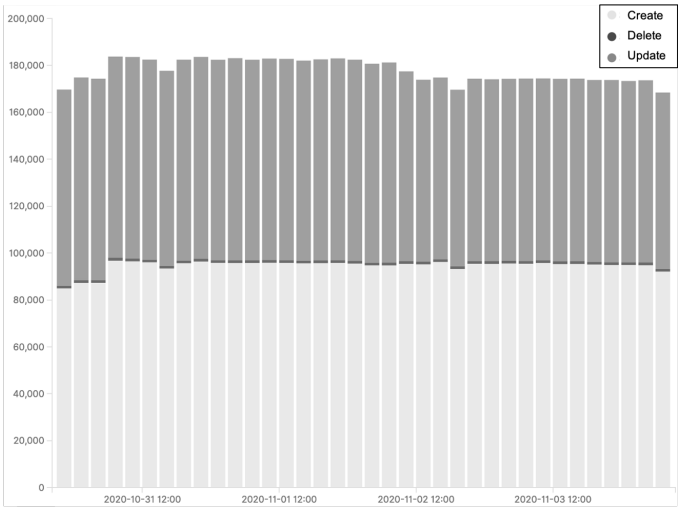

図 4 Admission requests in steady state.

3.2 Admission controller における署名検証

本節では，admission controller での署名検証をどのように実現しているか説明する. まず，図 5 に示すように，Kubernetes リソースに対する署名 (B) は, YAML ファイル (A) から生成され，添付される. また，リソース全体を符号化したメッセージ (C) も同様に添付される. 署名は, 一般的に使用されている GNU Privacy Guard(GPG) や Rivest-Shamir-Adleman(RSA) を利用する. この署名付きリソース (D) がクラスタに適用されると, admission controller はリソース (E) を含む admission リクエストを受け取るが, このオブジェクトは, 2.2 で説明したように, Kubernetes クラスタによって書き換えられており, そのまま署名検証を行うことはできない.

署名検証に基づいた admission controller でのリクエスト制御の実現のため, 提案手法の検証サーバは以下の処理を行う.

(1) Admission リクエスト (E) に含まれる Kubernetes リソースから署名 (B) とメッセージ (C) を取得する.
(2) 署名 (B) とメッセージ (C) を使用して添付の署名が正しいことを検証する.
- 署名検証が失敗した場合，その admission リクエストをブロックする.
- 署名検証が成功した場合, (3) の処理を行う.

表 1　Example of admission requests in steady state

Namespace	*Kind*	*UserName*	*Total*
openshift-kube-scheduler	ConfigMap	kube-scheduler	196,853
openshift-cloud-credential-operator	ConfigMap	openshift-cloud-credential-operator:default	196,810
kube-system	ConfigMap	kube-controller-manager	131,481
hive	ConfigMap	hive:hive-controllers	98,686
openshift-cluster-storage-operator	Lease	openshift-cluster-storage-operator:csi-snapshot-controller	79,029
openshift-monitoring	ConfigMap	openshift-monitoring:cluster-monitoring-operator	62,825
policies	PlacementBinding	rh-acm:multicluster-operators	43,464
policies	PlacementRule	rh-acm:multicluster-operators	43,464
policies	Policy	rh-acm:multicluster-operators	43,464
kube-node-lease	Lease	node:ip-xx-x-xxx-xx.xxxxx.compute.internal	39,570

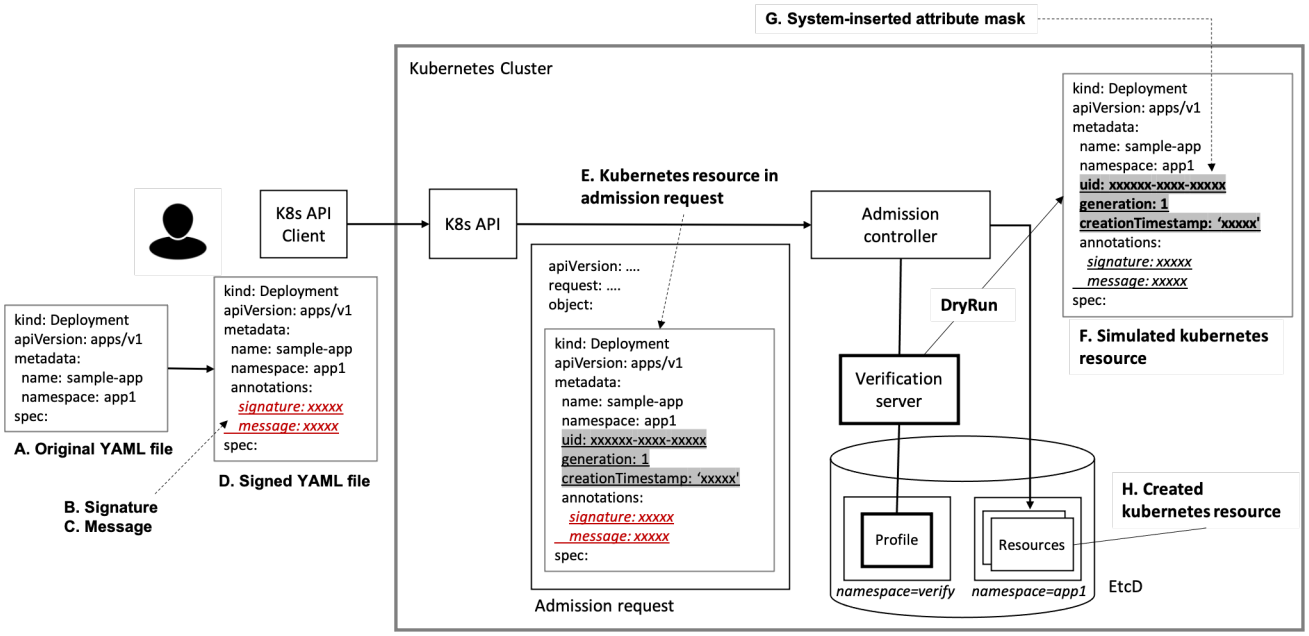

図 5　Overview of proposed approach

(3) デコードしたメッセージ (C) を使用して DryRun を行い, API サーバからシミュレートされた Kubernetes リソース (F) を受け取る. (図 6).

(4) メッセージ (C) と DryRun 結果の Kubernetes リソース (F) を比較し, クラスタによって追加されたフィールド (G) を計算する. (図 7).

(5) クラスタによって追加されたフィールド (G) でマスキングした Kubernetes リソース (E) と Kubernetes リソース (F) を比較し, 一致するか確認する.

- この 2 つのリソースが一致した場合, admission リクエストに含まれるリソース (E) はメッセージ (C) と一致していると判定できるため, admission リクエストを許可する.
- リソースが一致しない場合, admission リクエストに含まれるリソース (E) は改竄されているため, admission リクエストをブロックする.

(3) における処理では, Kubernetes に標準で備わっているオプションである DryRun モードでリクエストを作成し, Kubernetes リソースに対するクラスタの振る舞いをシミュレートする. DryRun モードでは, Kubernetes リソースをストレージに永続化しない以外は通常のリクエストと同様に処理するため, 通常のリクエストに限りなく近い結果が得られる. 図 6 に示すように, 提案手法では, クラスタによって挿入されるフィールドを取得するために, 署名生成時の YAML マニフェスト (A) を用いて DryRun を行ない, API サーバを通過する前後の状態を比較している. 図 8 は提案手法において Kubernetes リソースが処理されるフローを示している. YAML マニフェストは開発者によって署名された後, Kubernetes クラスタにデプロイされ, admission コントローラに admission リクエストとして渡される. Admission コントローラにおいてリソースが改竄されていないと検証されるとクラスタにリソースが作成される.

3.3　リクエスト制御のためのプロファイルフレームワーク

提案手法のリクエスト処理では, 署名検証に進む前に

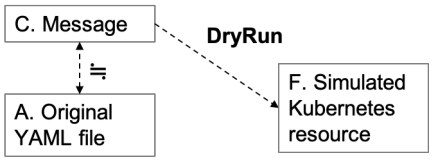

図 6 DryRun

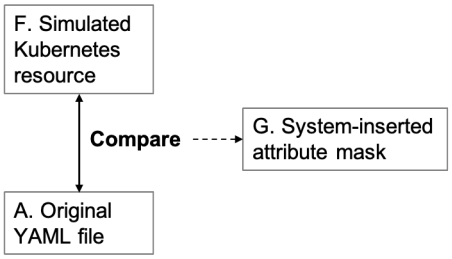

図 7 Acquire system-inserted attribute mask

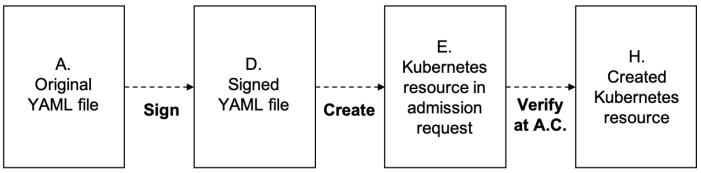

図 8 Request process flow

プロファイルによるリクエストのフィルタリングを行う. Admission リクエストには Kubernetes リソース (object) 以外にも”誰がリクエストを起こしたか (userName)”や, ”どこで起きたか (namespace)”, ”変更前の Kubernetes リソース (oldObject)”などの様々な情報が含まれている. 提案手法のプロファイルは, これらの情報を組み合わせて柔軟にリクエストをフィルタリングできるように構成されている. 提案手法の保護を有効にしつつアプリケーションをインストールする際には, ”Application Profile”を用意する. プロファイルに定義できるルールは以下である.

Protect Rules: 署名で保護したいリソースを定義する. ここで定義した条件に適合したリクエストは, admission controller において署名検証によって保護される. 以下の例は,「creds-secret」という名前の Secret を保護している. このように Kubernetes リソースの Kind や名前などの条件に基づいて保護対象を定義することができる.

```
protectRules:
- match:
  - kind: Secret
    name: creds-secret
```

Ignore Rules: クラスタの内部動作によって起こる脅威ではないリクエストを署名がなくても通過させるために使用する. また, 署名による保護から明示的に除外するリソースを定義することも可能である. 以下に無視ルールの例を示す. この例では, Kubernetes クラスタの ServiceAccount である replicaset-controller が Pod を操作することを許可しているが, これは, Replicaset が作成されると replicaset-controller によって自動的に Pod が作られるという Kubernetes クラスタの内部的な動作を許可するためである. 上記の例のようなクラスタ内部から発生する脅威ではないリクエストは多様であり Kubernetes クラスタであれば共通して起こる. 提案手法では, このような内部的な動作を許可するための定義をリストした”Common Profile”を用意し, クラスタに影響を与えずに提案手法を有効化できるようにしている.

```
ignoreRules:
- match:
  - kind: Pod
    username: kube-system:replicaset-
        controller
```

Ignore Attrs: クラスタやアプリケーションのインストーラによって内部的に書き換えるフィールドを定義する. ここで指定したフィールドに関しては, 署名検証の際に差異があっても許可される. また, スケーラビリティのためのアプリケーション設定などは, ユーザビリティの観点から署名がなくても変更を許可したいとい場合がある. このようなフィールドを定義し, リソースを保護しつつ使いやすさを損なわない設定をすることも可能である. 以下の例では, Helm Chart を使用してアプリケーションをインストールする際に自動的に追加されるフィールドが定義されている.

```
ignoreAttrs:
- attrs:
    - metadata.annotations.meta.helm.sh/
        release-namespace
    - metadata.labels.app.Kubernetes.io/
        managed-by
    - metadata.annotations.meta.helm.sh/
        release-name
    match:
    - kind: "*"
```

4. 評価

署名済みのアプリケーションを Kubernetes クラスタにインストールする実験を行い, アプリケーションのインストール中に発生したリクエストが提案手法によって適切に処理されるかどうかを評価した. 提案手法では, アプリケーションのインストール時に発生するリクエストは以下のように分類できる.

(1) 提案手法の対象外となるリクエスト (例：Event, Lease)
(2) Common Profile の条件に合致するリクエスト
(3) Application Profile の条件に合致するリクエスト
(4) 署名検証の対象となり, 有効な署名を持つリクエスト

(5) 署名検証に失敗したため, ブロックされたリクエスト
(6) プロファイルの条件に一致せず, かつ対象外でもないリクエスト

提案方式では, (2)〜(6) のリクエストについて処理を行う. しかし, 署名生成時のリソースと admission リクエスト内のリソースに差異が見つかり, 正しく署名した場合でも署名検証が失敗するため, (5) に分類される場合がある. また, (6) に分類されるリクエストの場合は, リクエストに署名が付いていない上にプロファイルで定義された条件も適合しない, つまり, 提案方法では保護されていない状態である. 実験ではアプリケーションに正しく署名をつけてインストールしているため, (5) や (6) に分類されるリクエストは発生しないはずである. 今回の実験では, 提案手法がプロファイルと署名の検証によって, どれだけ正確にリクエストを処理できるかを評価した.

4.1 実験方法

実験は, 軽量な Kubernetes クラスタである Kind クラスタ [1](Kubernetes v1.19.7) 上で行なった. アプリケーションは Kubernetes 用のアプリケーションインストーラである [2] と,Operator [4] を使用してインストールした. インストールするアプリケーションはそれぞれ Artifact Hub [3] と OperatorHub.io [5] からランダムに選択している. 上記の 2 つに加えて, GitHub で公開されているオープンソースの Operator を手動でインストールした場合も評価した.

(1) 名検証対象を指定する Protect Rule を Application Profile に設定
  (a) アプリケーションのインストールに使用する YAML マニフェストを取得
    - Helm Charts: Artifact Hub からアプリケーションパッケージを取得し, Helm template コマンドでパッケージからインストールされるリソースを取得
    - Operator: OperatorHub.io からインストールに使用する yaml を取得
    - Manual operator installation: GitHub からソースコードを取得
  (b) 取得したリソースを署名検証対象として Application Profile に定義
  (c) ServiceAccount がリソースに含まれている場合は, アプリケーションの serviceAccount による操作はその Namespace 内では許可するため, プロファイルの Ignroe Rule に定義
(2) 手元にある YAML マニフェストに署名をする
(3) アプリケーションを入れる Namespace にプロファイルをインストール
(4) 署名したアプリケーションのリソースをデプロイ

4.2 実験結果および考察

各アプリケーションのインストール時に発生したリクエストを分類した結果を表に示す. 実験では, 計 2298 件のリクエストが発生し, 署名検証に失敗したリクエストは 2 件であった. また, 39 件のリクエストは, 提案手法の保護対象であるが署名もなく, プロファイルの条件からも漏れてしまっていた. したがって, 98.3%は正しく処理されたことになる. 正しく処理されなかったリクエストについては, Helm chart による NGINX Ingress Controller のインストールでは, Service の作成で署名検証に失敗していた. これは, Port に関するフィールドでは, DryRun をする際に Kubernetes の仕様で値が埋め込まれることに起因している. そのため, 以下のように healthCheckNodePort を署名検証の際に除外するように Profile に追記することで, 想定通りの振る舞いになる.

```
ignoreAttrs:
- match:
  - kind: Service
    name: my-release-nginx-ingress
  attrs:
  - spec.healthCheckNodePort
```

また, Service リソースにおいて" Port"というフィールドで署名作成時と差分が出るのは, NGINX Ingress Controller 固有の現象ではない. そのため, Common Profile に以下のように定義することでより汎用性があるプロファイルになる.

```
ignoreAttrs:
- match:
  - kind: Service
  attrs:
  - spec.*Port
```

また, operator でインストールされた Istio と Nginx, Tektoncd では, operator の ServiceAccount 以外にも, operator から作成された ServiceAccount がリクエストを起こしていた. Operator によって作成されたアプリケーション固有の ServiceAccount がアプリケーションのリソースに対して操作することは自然な振る舞いで脅威ではないと考えられる. アプリケーション用のプロファイルに以下のように条件を追加することで提案手法のフィルタリングを適切な結果に修正可能である.

```
ignoreRules:
- match:
  - username: nginx-ingress-operator
  - username: my-nginx-ingress-controller
```

このように提案したプロファイルを使用することで, Admission controller では, 受け取ったリクエストを適切に分類できていることがわかる. また, クラスター内部の動作に

表 2 Classification results of requests

application	*installer type*	*1)out of scope*	*2)common profile*	*3)app profile*	*4)valid signature*	*5)invalid signature*	*6)unprocessed*
Kubeview	Helm Chart	31.3%	53.1%	0.0%	15.6%	0.0%	0.0%
NGINX Ingress Controller	Helm Chart	32.5%	35.0%	10.0%	20.0%	2.5%	0.0%
Jenkins	Helm Chart	44.6%	33.9%	0.0%	21.4%	0.0%	0.0%
Nutanix CSI Volume Driver	Helm Chart	51.2%	25.6%	0.0%	22.0%	1.2%	0.0%
Yourls	Helm Chart	45.1%	40.8%	0.0%	14.1%	0.0%	0.0%
Minio	Operator	17.2%	73.9%	6.7%	2.2%	0.0%	0.0%
Jenkins	Operator	14.0%	36.0%	48.3%	1.7%	0.0%	0.0%
Argocd	Operator	23.1%	64.0%	12.0%	0.9%	0.0%	0.0%
Istio	Operator	39.3%	27.5%	28.1%	0.4%	0.0%	4.7%
Grafana	Operator	25.9%	33.0%	39.8%	1.4%	0.0%	0.0%
Nginx Ingress	Operator Manual	25.0%	16.7%	37.5%	9.7%	0.0%	11.1%
Tektoncd	Operator Manual	52.6%	7.7%	36.3%	2.0%	0.0%	1.4%

基づく変更を署名検証対象から効率的に除外するための柔軟なプロファイル定義が可能であり, Admission controller での Kubernetes リソースの署名検証が機能することが評価結果からわかる.

5. 関連研究

クラウドのインテグリティ保護を目的とした研究は数多くされているが, そのほとんどはシステムの変更を監視する検出型の手法をとっている. Hai Jin et al. [16], Dragi et al. [15] はルールベースの変異検出手法を提案している. H.Kitahara et al. [17] は, 本来の振る舞いから逸脱したファイル変更やプログラム実行イベントを事前定義を用いずに効率的に検出する手法を提案している. 本稿では, 従来の検知方法とは異なり, Kubernetes リソースの認証されていない変更を署名で防ぐ方法を提案している.

6. おわりに

本稿では, admission controller における署名検証に基づいて Kubernetes リソースのインテグリティを保護する手法を提案した. 提案手法では, admission リクエストに含まれる Kubernetes リソースと署名生成時の YAML マニフェストに差異が生じるという課題を解き, admission controller においても適切に Kubernetes リソースの署名検証ができるようにした. また, クラスタが内部的に起こすリクエストを効率的にフィルタリングするための柔軟なプロファイルを提案し, そのプロファイルに基づいたリクエスト制御を実現した. さらに, 実際のアプリケーションを Kubernetes クラスタにインストールして実験を行い, 提案手法の有効性を示した. 実験は, Helm Charts や Operator などの一般的な Kubernetes アプリケーションのインストーラを使用して, 複数のアプリケーションを対象に行なったため, 一般的なアプリケーションに対しては同様の結果が得られると考えている.

参考文献

[1] “Kind”, Available: https://kind.sigs.k8s.io
[2] “Helm Charts”, Available: https://helm.sh
[3] “Find, install and publish Kubernetes packages”, Available: https://artifacthub.io
[4] “What is a Kubernetes operator?”, Available: https://www.redhat.com/en/topics/containers/what-is-a-Kubernetes-operator
[5] “Welcome to OperatorHub.io”, Available: https://operatorhub.io
[6] “Red Hat OpenShift Container Platform: Kubernetes for rapid innovation”, Available: https://www.redhat.com/en/resources/openshift-container-platform-datasheet
[7] “What is Kubernetes?”, Available: https://Kubernetes.io/docs/concepts/overview/what-is-Kubernetes/
[8] “Using Admission Controllers”, Available: https://Kubernetes.io/docs/reference/access-authn-authz/admission-controllers/
[9] “NIST Risk Management Framework”, Available: https://csrc.nist.gov/Projects/risk-management/sp800-53-controls/release-search#!/800-53
[10] “Dry-run”, Available: https://Kubernetes.io/docs/reference/using-api/api-concepts/#dry-run
[11] “YAML: YAML Ain’t Markup Language”, Available: https://yaml.org/
[12] De Benedictis, Marco & Lioy, Antonio. (2019). Integrity verification of Docker containers for a lightweight cloud environment. Future Generation Computer Systems. 97. 10.1016/j.future.2019.02.026.
[13] “Verifying Red Hat container image signatures in OpenShift Container Platform 4”, Available: https://access.redhat.com/verify-images-ocp4
[14] “Portieris”, Available: https://github.com/IBM/portieris
[15] Dragi Zlatkovski, Aleksandra Mileva, Kristina Bogatinova, and Igor Ampov. A new real-time file integrity monitoring system for windows-based environments. ICT Innovations 2018, Web Proceedings ISSN 1857-7288, pages 243–258, 2018.
[16] Hai Jin, Guofu Xiang, Deqing Zou, Feng Zhao, Min Li, and Chen Yu. A guest-transparent file integrity monitoring method in virtualization environment. Computers & Mathematics with Applications, 60(2):256 – 266, 2010. Advances in Cryptography, Security and Applications for Future Computer Science.
[17] H.Kitahara, K.Gajananan, and Y.Watanabe. Highly-scalable container integrity monitoring for large-scale Kubernetes cluster. In 2020 IEEE International Conference on Big Data (Big Data 2020), 2020.