Kube-apiserver

Informacja:

1. Komponent pozwala na walidację oraz konfigurację obiektów API, przechowując tym samym informacje w magazynie danych Etcd.
2. Jednym z zadań elementu jest implementacja funkcjonalności autoryzacji, uwierzytelnienia oraz Admission Control w celu kontroli obsługiwanych żądań.
3. W przypadku kompromitacji elementu, atakujący w dużej liczbie przypadków przejmie całkowitą kontrolę nad klastrem
4. Każdy nieuwierzytelniony użytkownik Kube-apiserver jest umieszczony w grupie system:anonymous.
5. Kube-apiserver domyślnie używa wielu obiektów Admission Control w postaci w wtyczek, które umożliwiają walidację żądań API lub ich modyfikację. (Przykładowo: Limit Ranger – dodanie ograniczenia zasobów obiektu Pod oraz NodeRestriction – ograniczenie dostępów dla Kubelet do obiektów Node/Pod). Często domyślnymi wtyczkami są NamespaceLifecycle, NodeRestriction, LimitRanger, ServiceAccount, DefaultStorageClass, ResourceQuota.

Rekomendacja:

1. Należy zadbać o to, aby żaden z portów udostępnianych przez ten element nie był typu insecure.
2. Sekrety powinny być szyfrowane „at rest” w komponencie Etcd i/lub zostać przechowane w Secret Store takim jak Hashicorp Vault, KMS lub za pomocą szyfrowanych plików.
3. Ruch wchodzący oraz wychodzący dla Kube-apiserver powinien być w pełni szyfrowany przy pomocy protokołu TLS.
4. Zaleca się wyłączenie anonimowego uwierzytelnienia i rozpoczęcie wymuszania uwierzytelnienia dla wszystkich klientów API.
5. Zaleca się wyłączenie anonimowego uwierzytelnienia i wymuszanie uwierzytelnienia dla wszystkich klientów API. Należy dodatkowo zadbać o autoryzację użytkowników API przy pomocy RBAC. Poprzez wykorzystanie punktów końcowych grupa system:anonymous umożliwia przykładowo odczyt wersji Kube-apiserver.
6. Biorąc pod uwagę pozostałe opcje konfiguracyjne dla Kube-apiserver, warto rozważyć dodanie flagi audit-webhook-config-file, w celu eksportowania logów do zewnętrznej instancji centralizującej i przechowywującej informacje.
7. Przygotowanie pliku konfiguracyjnego dla Audit Policy jest niezbędne, aby określić interesujący stopień szczegółowości zgłaszanych zdarzeń.
8. W przypadku trybu logowania RequestResponse, oprócz istotnych informacji, mogą zostać zgłoszone również dane wrażliwe w tym zakodowane sekrety. Naraża to infrastrukturę na wyciek sekretów, dlatego zalecanym rozwiązaniem może być zastosowanie trybu Metadata

9. Dodatkowe opcje konfiguracyjne:
    - Ustawienie opcji anonymous-auth na wartość false (opcja ustawiania na wartość true podczas instalacji)
    - Ustawienie opcji admission-control (zastosowanie wartości takich jak: AlwaysPullImages, NamespaceLifecycle, ServiceAccount, NodeRestricion, EventRateLimit)
    - Zaplanowanie sposobu logowania informacji w klastrze, retencji logów oraz ich maksymalnej wielkości (ustawienie opcji względem możliwych ataków na infrastrukturę)
    - Ustawienie opcji service-account-lookup na wartość true
    - Ustawienie opcji profiling na wartość false
    - Upewnienie się, iż wartość opcji admission-control nie zawiera wartości AlwaysAdmit

Etcd

Informacja:

1. Komponent odpowiada za zarządzanie danymi konfiguracyjnymi, stanem oraz metadanymi klastra.
2. Narzędziem linii komend używanym do interakcji z Etcd jest Etcdctl. Narzędzie umożliwia pełne zarządzanie magazynem przy wykorzystaniu adresu ip instancji Etcd, pod warunkiem posiadania odpowiedniego certyfikatu, w tym klucza prywatnego.
3. Właściwością komponentu jest brak wsparcia dla plików lub danych o dużych rozmiarach. Limit dla wysyłanych żądań to 1.5 MB, limit magazynu danych to 2GB, a limit dla logów audytowych dla pojedynczego żądania to 256 KB.
4. Dla opcji dedykowanego rozwiązania Etcd, ewentualna kompromitacja instancji Control Plane nie umożliwia bezpośrednio uzyskania wszystkich wrażliwych danych klastra.
5. Wystarczającą czynnością jest uzyskanie przez atakującego możliwości odczytu i zapisu do magazynu danych, co będzie skutkować pełną kontrolą nad sposobem działania klastra.

Rekomendacja:

1. Dostęp komponentu do Etcd powinien być ograniczony wyłącznie dla elementu Kube-apiserver, gdzie dostęp będzie ściśle monitorowany.
2. Przechowywane dane powinny być szyfrowane, zgodnie z aktualnie obowiązującymi standardami. Wymagane jest stosowanie szyfrowania typu At-rest, jak i In-transit, a przy wykorzystaniu szyfrowania At-rest sekrety można dodatkowo przechowywać w Secret Store takim jak KMS (ang. Key Management Services) lub bezpośrednio w szyfrowanych plikach.
3. W przypadku lokalnego przechowywania sekretów należy ograniczyć odpowiednio dostępy do pliku konfiguracyjnego EncryptionConfiguration, tak aby dostęp do odczytania był możliwy tylko dla komponentu Kube-apiserver. Zaleca się:
    - Ustawienie opcji encryption-provider-config
    - Zastosowanie szyfrowania aescbc dla konfiguracji
4. Przekroczenie limitu logów audytowych spowoduje niezapisanie danych, co potencjalny atakujący może wykorzystać do własnych celów.
5. Zalecane jest wdrożenie etapu uwierzytelnienia podczas łączenia się z komponentem Etcd.
6. Zalecane jest utworzenie dedykowanego urzędu certyfikacyjnego (CA) dla komponentu Etcd (domyślne ustawienie przy instalacji przez Kubeadm).

Kubelet

Informacja:

1. Jest agentem węzła wspierającym komunikację Worker Node z Control Plane.
2. Kubelet jest umieszczany na każdej instancji Worker Node klastra, w celu zarządzania i uruchamiania kontenerów dla aplikacji. Funkcjonalności Kubelet obejmują również zarządzanie logami na każdym z węzłów oraz port-forwarding.
3. Domyślnie Kube-apiserver nie uwierzytelnia się do punktów końcowych Https Kubelet. Żądania z Kube-apiserver są traktowane jako anonimowe.
4. Kubelet odpowiada za wysyłanie żądań adresów IP dla zasobów oraz publikuje punkty końcowe Https, które umożliwiają kontrolę nad kontenerami oraz węzłami.
5. Kubelet wykorzystuje odpowiedni gniazdo (ang. socket) do interakcji ze środowiskiem uruchomieniowym. W przypadku containerd CRI plug-in, następuje uruchomienie serwera GRPC, który nasłuchuje na Unix Socket /run/containerd/containerd.sock.
6. Jednym z zadań Kubelet jest montowanie wolumenów do obiektu Pod podczas działania, gdzie wolumen zawiera najczęściej sekrety w formie tekstowej z danymi w postaci niezaszyfrowanej.
7. Do poprawnego odbierania żądań przez komponent wymagane jest umożliwienie komunikacji wejściowej dla skonfigurowanego portu, gdzie domyślnie jest to port 10250. Dodatkowo często zezwala się na ruch wejściowych dla uruchamianych serwisów z zakresu 30000-32767.
8. Komponent odpowiada za przechowywanie oraz rotację logów, umożliwiając dostęp do nich dostęp za pośrednictwem polecenia kubectl logs. Funkcjonalność ta ogranicza się jednak do czasu życia kontenera, obiektu Pod lub węzła. Jeżeli nastąpi usunięcie jednego z wymienionych obiektów, logi zostaną utracone.
9. Domyślnie Kubernetes zapisuje logi Kubelet oraz środowiska uruchomieniowego do usługi systemowej Journald.
10. Dostęp dla Kubelet jest ograniczany przy pomocy NodeRestriction dla Admission Controller. W takim przypadku konto serwisowe Kubelet jest przypisywane do roli system:node, co powoduje iż w przypadku kompromitacji pojedynczego węzła, atakujący jest w stanie pobrać sekrety tylko i wyłącznie ze skompromitowanego węzła. Plugin pozwala również na ochronę przed atakami polegającymi na zmianie etykiet dla krytycznych obiektów klastra.

Rekomendacja:

1. Zalecane jest wyłączenie anonimowego uwierzytelnienia (punktów końcowych Https Kubelet) oraz ustawienie odpowiedniej konfiguracji opcji authorization-mode. Autoryzacja domyślnie jest ustawiona jako AlwaysAllow, co powoduje iż domyślnie proces autoryzacji nie istnieje. Zalecana jest implementacja bezpieczniejszej opcji, takiej jak Webhook.
2. Zasoby klastra powinny być pod ścisłą kontrolą, gdzie możliwość kontroli sprowadza się do implementacji reguł za pomocą LimitRanges, ResourceQuotas oraz limitów Process ID. Przykładowo bardzo łatwo może dojść do wyczerpania zasobów Process ID, bez jednoczesnego naruszenia pozostałych limitów. W takim przypadku może spowodować to niedostępność komponentów Kubelet oraz Kube-proxy.
3. Każdy obiekt Kubelet ma własny lokalny port API, który domyślnie umożliwia nieautoryzowany dostęp do odczytu Node-local Pods. Jeżeli klaster nie posiada nałożonych reguł sieciowych ograniczających dostęp do sieci węzła, atakujący jest w stanie z poziomu obiektu Pod dokonać ataku na komponent Kubelet. Należy nałożyć wymagane reguły sieciowe.

4. Zaleca się wyłączenie read-only port poprzez ustawienie flagi read-only-port na wartość 0, z uwagą na fakt zastosowania portu do uzyskiwania odpowiednich metryk.

5. Kube-apiserver powinien weryfikować certyfikat Kubelet poprzez zastosowanie flagi --kubelet-certificate-authority. Domyślnie nie jest to weryfikowane.
6. Należy ograniczyć dostęp dla Kubelet przy pomocy NodeRestriction dla Admission Controller.
7. Należy rozważyć wdrożenie Node Authorizer przy zastosowaniu flagi --authorization-mode=Node, co pozwala na ograniczenie dostępów Kubelet do wyłącznie odczytu wymaganych przez węzeł obiektów.
8. Należy rozważyć implementację rozwiązań eksportujących logi emitowane przez CRI (np. containerd) przy użyciu przykładowo Fluent Bit.
9. Należy rozważyć ustawienie opcji:

    - Ustawienie opcji kubelet-certificate-authority z ścieżką do certyfikatu
    - Ustawienie opcji RotateKubeletClientCertificate na wartość true
    - Ustawienie opcji protect-kernel-defaults na wartość true (szczególnie istotne, jeśli administrator systemu przeprowadzał proces Hardenig jądra Kernel)
    - Upewnienie się, iż wartość flagi keep-terminated-pod-volumes jest ustawiona na wartość false

Admission Webhook

Informacja:

1. Pojęcie Webhook odnosi się do wysłania przez obiekt odpowiedzi zwrotnej Http podczas pewnego określonego zdarzenia. W klastrze tryb Webhook powoduje, iż na etapie ustalania uprawnień nastąpi przygotowanie zapytania do zewnętrznej usługi REST. Wyróżnia się następujące typy:

    - **Validating Admission Webhook** – wywołuje elementy walidujące, które odpowiadają danemu żądaniu. Elementy są wywoływane równolegle. W przypadku niespełnienia warunku walidacji jednego z elementów weryfikujących, żądanie zostanie odrzucone.
    - **Mutating Admission Webhook** – wywołuje elementy modyfikujące, które odpowiadają danemu żądaniu. Działanie elementu mutującego powodującego zmianę na obiekcie zanim zostanie on przechowany. Sposób działania może wpłynąć na wykonanie pozostałych elementów mutujących lub walidujących.

2. Dzięki zastosowaniu funkcjonalności Webhook w klastrze Kubernetes, istnieje możliwość wprowadzenia dodatkowych metod detekcji oraz prewencji, które ściśle określą możliwe czynności do wykonania, zmniejszając jednocześnie ryzyko kompromitacji.
3. Wymienione rodzaje Admission Webhook są częścią Admission Controller. Komponent Admission Controller jest obiektem będącym pośrednikiem żądań do Kube-apiserver. Element ten jest uruchamiany po uwierzytelnieniu oraz autoryzacji żądania. Oprócz dwóch wymienionych kontrolerów istnieją również inne kontrolery, przykładowo AlwaysPullImages, AlwaysAdmit czy też ImagePolicyWebhook.
4. W niektórych przypadkach istnieją dwa różne podejścia do kontroli wykorzystywanych przez środowisko obrazów, gdzie jednym rozwiązaniem będzie implementacja kolejnego kontrolera

typu Admission Control, a drugim implementacja Validating lub Mutating Admission Webhook.
5. Podczas stosowania ValidatingAdmissionWebhook w przeciwieństwie do ImagePolicyWebhook, nie jest wymagana tak duża ilość pracy do wdrożenia ograniczeń oraz podejście nie wymaga od administratora niepotrzebnego nadawania dostępów do klastra. W takim przypadku jest to dodatkowy zasób, gdzie kontrolę dostępu wdraża się poprzez RBAC.
6. Rozwiązanie ValidatingAdmissionWebhook wprowadza ryzyko - w przypadku niedostępności obiektu Pod lub serwisu, ograniczenia są znoszone, co umożliwia pobranie dowolnego obrazu w klastrze.

Rekomendacja:

1. Należy rozważyć implementację Validating / Mutating Admission Webhook, przy pomocy gotowych rozwiązań lub implementując własny mechanizm w przypadku wymagania kontroli wybranego procesu.

Service Mesh

Informacja:

1. Service Mesh jest rozwiązaniem, które dostarcza funkcjonalności umożliwiające zarządzanie ruchem, dostępem, szyfrowaniem połączeń klastra oraz zbieraniem metryk.
2. Rozwiązanie pozwala zoptymalizować komunikację oraz uniknąć przestojów w przypadku skomplikowanej architektury systemu. Jest to element wspomagający pracę trasowania pomiędzy serwisami, dbając przy tym o optymalizację.
3. Istio pozwala na wdrożenie dodatkowych narzędzi, gdzie do najważniejszych z nich można zaliczyć Kiali dla wygodnej wizualizacji zależności obiektów klastra oraz Prometheus z Grafana do wizualizacji otrzymywanych metryk.
4. Narzędzia takie jak Istio pozwalają na określenie przestrzeni nazw dla których Service Mesh ma zostać wdrożony, co można dokonać poprzez zastosowanie etykiet dla obiektów.

Rekomendacja:

1. Należy rozważyć implementację Service Mesh w celu uzyskania szyfrowania ruchu pomiędzy Workloads klastra (ruch wewnętrzny) oraz monitoringu wdrożonych aplikacji i ich ruchu.
2. Należy przygotować dashboard'y wizualizujące sposób komunikacji serwisów, wskazując elementy wymagające poprawy.

Kube-proxy

Informacja:

1. Kube-proxy odpowiada za adresację IP, wykorzystując do tego Iptables ze złożonością obliczeniową O(n) lub Ipvs ze złożonością obliczeniową O(1).
2. Element w dużej mierze decyduje o dostępności serwisów dla Pod. Komponent programuje Kernel NAT Table w celu przeprowadzenia dynamicznej translacji adresów.
3. Istnieją dwa główne sposoby instalacji Kube-proxy: instalacja bezpośrednio na węzłach oraz instalacja jako DaemonSet. W przypadku instalacji bezpośredniej jest to Daemon instalowany bezpośrednio w systemie węzła, korzystający z certyfikatów TLS. Podejście DaemonSet opiera

się na obiektach typu Pod uruchomionych na każdym węźle oraz zamiast certyfikatów TLS wykorzystywane jest konto serwisowe.

Rekomendacja:

1. Istotne jest, aby komunikacja z Kube-apiserver odbywała się przy użyciu szyfrowanej komunikacji.
2. Komponent umożliwia zbieranie metryk, które pozwalają określić opóźnienia oraz kody błędów dla żądań Http klastra. Należy skonfigurować eksport metryk dla komponentu.

Service

Informacja:

1. Do działania serwisu wymagana jest współpraca komponentów takich jak Endpoints Controller oraz Kube-proxy.

Rekomendacja:

1. Należy być świadomym (odpowiednio udokumentować) jakie serwisy są udostępnianie przez klaster i na tej podstawie określić ryzyko, które może występować dla poszczególnych usług. Jeżeli dane serwisy są udostępnione na zewnątrz klastra, należy upewnić się czy są to niezbędne serwisy do dostarczenia rozwiązań dla danego klienta.
2. Warto rozważyć wdrożenie Quota, Limit Ranges oraz implementację konfiguracji ról RBAC dla osób użytkujących zasoby klastra, tak aby ograniczyć możliwości związane z tworzeniem serwisów dostępnych na zewnątrz klastra.
3. W celu zapewnienia szyfrowanej komunikacji, uzyskania większej liczby informacji oraz metryk dla komunikacji pomiędzy serwisami w klastrze, można rozważyć implementację rozwiązania Service Mesh.

CoreDNS

Informacja:

1. Element CoreDNS jest serwerem DNS (ang. Domain Name Server), który składa się z wielu wtyczek określających funkcjonalności takie jak buforowanie, metryki lub strefy podstawowe. Rozwiązanie jest domyślnie instalowane przez narzędzie Kubeadm. Serwer DNS w Kubernetes ma na celu umożliwienie zasobom w klastrze lokalizacji względem siebie. Stosowanym wcześniej rozwiązaniem było Kube-dns, jednak ze względu na występujące podatności w dnsmasq oraz problemy wydajnościowe, zostało ono zastąpione poprzez komponent CoreDNS.
2. Zapytanie jest sprawdzane przez zdefiniowane wtyczki, określając czy powinno ono zostać przetworzone. Jeżeli wtyczka zdecyduje się na przeprocesowanie zapytania, wtyczka zwróci właściwą odpowiedź do klienta.
3. Najczęściej dla każdego zapytania można otrzymać cztery stany wyjściowe: zapytanie zostało przetworzone, zapytanie nie zostało przetworzone, zapytanie zostało przetworzone z wymogiem przejścia przez dodatkowe wtyczki, zapytanie zostało przetworzone z podpowiedzią.

Rekomendacja:

1. W celu zwiększenia bezpieczeństwa komponentu podczas wdrożenia, należy upewnić się, iż wtyczka dla parametru health jest włączona. Można dokonać tego poprzez dodanie wtyczki health do listy wtyczek w Corefile.
2. Kluczowe dla bezpieczeństwa jest zbieranie metryk dla serwera DNS poprzez implementację wtyczek dla Prometheus.
3. Istotną kwestią jest optymalizacja wydajności serwera DNS poprzez implementację funkcjonalności Nodelocaldns działającej w postaci DaemonSet, która umożliwia wykorzystanie pamięci tymczasowej dla przyszłych zapytań.
4. Niektóre z wtyczek, które są nie używane, mogą narazić klaster na kompromitację. (Przykładowo: Atakujący może spróbować wykorzystać różnego rodzaju luki, takie jak DNS

Cache Poisoning dla starszych wersji oprogramowania CoreDNS, pod warunkiem wykorzystania wtyczki Cache przez serwer DNS). Należy zweryfikować wykorzystywane wtyczki przez komponent i usunąć zbędne.

Kube Controller Manager

Informacja:

1. Każdy z kontrolerów posiada utworzone konto serwisowe z przypisanymi uprawnieniami RBAC.
2. Uczestnictwo w grupie kube-controller-manager pozwala między innymi na dostęp do wrażliwych informacji w postaci sekretów, umożliwiając jednocześnie tworzenie obiektów takich jak konta serwisowe.
3. Kube Controller Manager domyślnie wykorzystuje wbudowaną grupę System:kube-controller-manager, a do procesu uwierzytelnienia wykorzystywana jest metoda X509 Client Certs.

Rekomendacja:

1. Istotne jest upewnienie się, iż dane uwierzytelniające kont serwisowych są skonfigurowane oddzielnie dla każdego kontrolera, co można osiągnąć przy zastosowaniu flagi --use-service-account-credentials.

Kube-scheduler

Informacja:

1. To jaką decyzję podejmie Kube-scheduler zależy od wielu czynników takich jak wymagania zasobów, ograniczenia reguł, Affinity/Anti-affinity, tolerancja żądań, priorytet oraz lokalizacja danych.
2. Etap filtrowania oraz punktacji podczas procesu planowania można sterować przy użyciu stosownych polityk oraz profili.
3. Informacje potrzebne do podjęcia decyzji są dostępne w magazynie danych Etcd. Kube-scheduler może być sterowany poprzez Etcd lub Kube-apiserver.
4. Obiekty Pod mogą wybrać z jakiego rodzaju Kube-scheduler skorzystają (Configure Multiple Schedulers).
5. W przypadku kompromitacji komponentu Kube-scheduler istnieje szansa, iż atakujący będzie w stanie kontrolować wydajnością klastra oraz jego dostępnością.
6. Scheduler domyślnie nie uwierzytelnia oraz nie autoryzuje dostępu do metryk, gdzie powiązanie komponentu z interfejsem lokalnym jest jednym z rozwiązań.
7. Po instalacji domyślnie Kube-scheduler wykorzystuje wbudowaną grupę System:kube-scheduler, a do procesu uwierzytelnienia wykorzystywana jest metoda X509 Client Certs.

Rekomendacja:

1. Domyślnie planowane zasoby są weryfikowane przez Kube-apiserver. Jeżeli atakującemu uda się dokonać wpisu bezpośrednio do Etcd, ominie on wtedy część zabezpieczeń takich jak reguły PodSecurityPolicy. Należy więc upewnić się, iż komponent Etcd jest właściwie zabezpieczony.
2. Sterując konfiguracją Kube-scheduler, istnieje możliwość przypisania zasobów tak, by były one odseparowane względem węzła. W przypadku kompromitacji Pod, nadal jest uzyskiwana

separacja instancji maszyny, co wprowadza dodatkowe zabezpieczenie przed np. niedostępnością usług (pod to node scheduling - nodeSelector, node affinity, inter-pod affinity). Można rozważyć zastosowanie opcji dla niektórych przypadków.
3. Zalecane jest stosowanie dedykowanego konta serwisowego do pobierania metryk dla komponentu.
4. Należy zweryfikować ustawienie Profiling dla wartości false (właściwa wartość), które jest zalecane tylko w przypadkach rozwiązywania problemów z wydajnością klastra.

Ingress Controller

Informacja:

1. Rozwiązanie może zostać wdrożone jako Deployment lub DaemonSet.
2. Ingress Controller uruchamia w klastrze obiekty Pod w trybie uprzywilejowanym oraz posiada dostępy do obiektów typu sekret.
3. Ingress Controller wymaga konta serwisowego, którego uprawnienia można modyfikować przy użyciu RBAC.

Rekomendacja:

1. Jeżeli wdrożenie jest typu Deployment, zaleca się implementację Ingress Controller w połączeniu z Anti-affinity rule. Zastosowanie opcji Anti-affinity zapobiega umieszczenia replik na tych samych węzłach, co w przypadku awarii węzła zapobiega niedostępności usług.
2. Istnieje możliwość wdrożenie silnika Web Application Firewall (WAF) (Przykładowo: Domyślnie dla Nginx Ingress Controller jest to Modsecurity). Zaleca się wdrożenie WAF zgodnie z podejściem Defense in Depth na poziomie Ingress Controller lub Reverse Proxy.
3. W przypadku włączenia metryk Prometheus, zaleca się stosowanie szyfrowanej wersji protokołu http.
4. W przypadku stosowania opcji HostNetwork, wymagane jest również zadbanie o zasady DnsPolicy.

Storage

Informacja:

1. Sposobem na ustandaryzowanie sposobu dostarczania wolumenów może być nałożenie reguł przy pomocy Pod Security Admission.

Rekomendacja:

1. Zalecane jest brak stosowania typów hostPath, a w przypadku braku wykluczenia tego typu, należy ograniczyć się do trybu tylko do odczytu. Stosowanie typu hostPath może skutkować nieautoryzowanym dostępem do danych uwierzytelniających klastra. Zamiast hostPath może zostać wykorzystany Local Persistent Volume. Podejście pozwala znacząco poprawić bezpieczeństwo oraz niezawodność montowanych wolumenów dla zasobów.
2. Należy zadbać o szyfrowanie komunikacji przy pomocy TLS pomiędzy Storage Plugin, a Storage Infrastructure, gdzie weryfikacja certyfikatu CA jest konieczna. Jeżeli istnieje możliwość dla sterownika CSI zastosowanie opcji transportEncryption, należy tą opcję włączyć.
3. W przypadku sekretów można rozważyć stosowanie External Secret Operator, który jest zewnętrznym systemem do zarządzania sekretami.

4. Należy zadbać o wykonywanie regularnej kopii zapasowej. Do tego celu można skorzystać z rozwiązań zewnętrznych dostawców oprogramowania takich jak Velero.

Instalacja oraz zarządzanie klastrem Kubernetes

Rekomendacja:

1. Przed instalacją klastra należy dokładnie zaplanować proces przyszłej aktualizacji wersji komponentów klastra.
2. Przed instalacją należy rozważyć problemy dotyczące kopii zapasowych oraz procesu Disaster Recovery.
3. Należy określić sposób skalowalności klastra oraz wymaganych przez przyszłe aplikacje zasobów w celu następnego nałożenia właściwych Quotas oraz LimitRanges.
4. Należy odseparować logiczne (przestrzeń nazw) zasobów w klastrze względem kategorii ryzyka jakie ze sobą dane zasoby wnoszą. Pozwoli to w przyszłości na łatwiejsze nakładanie reguł Network Policy oraz RBAC.
5. Należy skonfigurować import logów systemu do centralnego systemu logowania:

    - logów aplikacji (dostarczane przez uruchomione w klastrze aplikacje),
    - logów klastra (logi komponentów Kube-apiserver, Kube-scheduler, Etcd,…),
    - zdarzeń klastra (informacje na temat stanu obiektów lub występujących błędów w klastrze – posiadają metadane dotyczące zdarzeń przechowywane w Etcd)
    - logów audytowych (pozwalających na chronologiczne określenie sekwencji wykonywanych w klastrze, umożliwiając badanie aktywności użytkowników i wdrożonych aplikacji. Proces rozpoczyna się od wystąpienia zdarzenia dla komponentu Kube-apiserver).

Kontrola dostępu instancji

Informacja:

1. Standardowym podejściem połączenia poprzez Jumphost (Bastion) jest utworzenie maszyny pośredniej Jump Host (Bastion), która istnieje w co najmniej dwóch sieciach – sieci prywatnej oraz sieci wewnętrznej poprzez które administratorzy uzyskują odpowiedni dostęp. W takim przypadku poszczególne węzły klastra znajdują się w sieci prywatnej. W sytuacji, kiedy administrator potrzebuje dostać się do klastra, musi początkowo uzyskać dostęp do maszyny Jump Host poprzez sieć wewnętrzną, a następnie wykorzystując połączenie sieci prywatnej uzyskać dostęp do wybranego węzła klastra.
2. Umożliwienie dostępu do klastra dla uprawnień RBAC administratora, jak i również zezwolenie użytkownikowi na uzyskanie dostępu do konta systemu Host o uprawnieniach Root dla danego węzła, jest umożliwieniem pełnej kontroli użytkownikowi nad klastrem z aplikacjami.

Rekomendacja:

1. Należy określić metodę dostępu do węzłów klastra. Można rozważyć zastosowanie rozwiązań takich jak Teleport, w celu ograniczenia ilości przyznawanych uprawnień dla administratora, co stanowi dobrą podstawę dla PAM (ang. Privileged Access Management).
2. Podmiot nie powinien posiadać możliwości połączenia się po protokole SSH bezpośrednio do instancji klastra Kubernetes. Należy korzystać ze stacji pośredniczących takich jak Bastion.
3. Zalecane jest zwrócić uwagę, aby ściśle określić dostęp do klastra tylko dla poszczególnych użytkowników, bez możliwości stosowania kont współdzielonych lub kont gościa.

4. Dostęp powinien być ściśle monitorowany, gdzie informacja powinna być wysyłana do centralnego systemu logowania, tak aby zapobiec możliwości zatarcia śladów przez atakującego.
5. Wymagane jest przygotowanie procedury na sytuację odebrania uprawnień w przypadku zwolnienia pracownika, rotacji stanowiska pracownika lub jego zawieszenia wobec długotrwałej nieobecności w pracy.
6. Logowanie informacji oraz zbieranie metryk z węzłów jest kluczowym elementem pozwalającym na dokładniejszą kontrolę dostępów do instancji Host. Przechowywanie logów również jest czynnością konieczną, nie tylko ze względu na potrzebę spełnienia wymagań konkretnego klienta, ale jak i również w celu umożliwienia pełnego zbadania wykonanych czynności na klastrze.
7. Węzły klastra oraz instancje Jump Host powinny zostać dodatkowo sprawdzone pod kątem zaleceń CIS Benchmark. Dla klastra Kubernetes można rozważyć zastosowania narzędzia Kube-bech.
8. Przed implementacją pełnego rozwiązania kontroli dostępów, zalecane jest rozważenie przyszłych wymagań bezpieczeństwa, jakie organizacja będzie musiała spełnić.

Przestrzeń nazw

Informacja:

1. Jest to izolacja logiczna zasobów polegająca na naniesieniu granic między zasobami w klastrze – umieszczenie w dedykowanych przestrzeniach nazw. Naniesienie granic ułatwia wdrożenie mechanizmów kontroli dostępu. Klaster staje się łatwiejszy w zarządzaniu i utrzymaniu.
2. Nie każdy obiekt umożliwia przypisanie do przestrzeni nazw (Cluster-wide objects / Persistent Volume).
3. Używane w połączeniu z: RBAC, Network Policy, Admission Control, Webhooks, Resource Quotas, LimitRanges, Pod Security Admission, Pod Anti-Affinity oraz NodeTaints.

Rekomendacja:

1. Nie należy tworzyć nowych zasobów w domyślnej przestrzeni nazw.

Kubernetes Service/User Accounts

Informacja:

1. W klastrze istnieją dwa podstawowe rodzaje kont: konta serwisowe (ang. service account) oraz konta użytkowników (ang. user account), gdzie kontrolę dostępów dla kont ustanawia się najczęściej przy pomocy mechanizmu RBAC.
2. ServiceAccount Admission Controller jest elementem będącym częścią komponentu Kube-apiserver, którego zadaniem jest kontrola stanu konta serwisowego przypisanego do danego obiektu Pod. (Zastosowanie przykładowo: Brak podanego konta serwisowego powoduje odrzucenie próby uruchomienia lub wykorzystanie w procesie uwierzytelniania kont serwisowych przy zastosowaniu Bearer Tokens).
3. Tokeny są domyślnie przechowywane w komponencie Etcd w postaci niezaszyfrowanej.

4. W przypadku potrzeby stosowania tokenu dla konta serwisowego, warto rozważyć podejście Bound Service Account Tokens, które są tokenami ułatwiającymi ograniczenie uprawnień oraz o ograniczonym czasie aktywności.
5. Jeżeli dla klastra zostaje wdrożony proces CI/CD, może być wymagane wstrzyknięcie tokenu, tak aby nie był on bezpośrednio przechowywany w pliku konfiguracyjnym wygenerowanym dla konta serwisowego. Czynność ta ma na celu zapobiegnięciu wycieku wrażliwych informacji do logów obsługującej proces aplikacji.
6. Konta użytkownika wygenerowanie poprzez podpisanie wniosku o certyfikat, wprowadzają problem odwołania wystawionego certyfikatu (brak wsparcia dla CRL – Certificate Revocation List).

Rekomendacja:

1. W klastrze Kubernetes występują konta domyślne – należy upewnić się iż nie są one aktywnie używane przez użytkowników klastra oraz ich uprawnienia są ściśle ograniczone.
2. Zadbanie o właściwy sposób udostępniania oraz przechowywania tokenów kont jest kluczowe, gdyż ich uzyskanie może być jednym z pierwszych celów atakującego. Zaleca się regularną wymianę tokenów dla kont serwisowych poprzez ich usunięcie przy pomocy Kubernetes Secrets API, gdzie kontroler automatycznie odnowi usunięty token.
3. Należy zaimplementować system monitoringu dla tokenów kont serwisowych.
4. Istnieje możliwość montowania dodatkowych sekretów w kontekście danych funkcjonalności dla kont serwisowych (Przykładowo: ImagePullSecrets – umożliwia wykorzystanie danych logowania dla repozytorium z obrazami). W przypadku montowania sekretów dla pobierania obrazów z repozytoriów, należy wykorzystać tą funkcjonalność.
5. Należy rozważyć możliwość skorzystania z zewnętrznych magazynu dla obiektów typu sekret.
6. Domyślnie dla każdej instancji obiektu Pod w klastrze dla wdrożenia, token konta serwisowego jest montowany do ścieżki kontenera, dostępnej z poziomu użytkownika. Należy wyłączyć domyślne montowanie tokenów poprzez zastosowanie opcji konfiguracyjnej automountServiceAccountToken z wartością false w ustawieniach konta serwisowego lub obiektu Pod.
7. Domyślne konto użytkownika w klastrze po instalacji przy użyciu Kubeadm, należy do OU system:masters, z którego nie powinno się korzystać do wykonywania codziennych czynności administratorskich (break glass only) oraz dostęp do tokenu tego konta powinien być ściśle ograniczony.
8. Kube-apiserver umożliwia zarządzanie sekretami oraz ich odczyt pod warunkiem posiadania uprawnień RBAC.
9. Zalecane jest utworzenie osobnego konta serwisowego dla każdej części danej aplikacji/rozwiązania (Przykładowo: Osobne konto serwisowe dla Frontend oraz osobne dla Backend) – szczególnie jeśli konta mają pewne uprawnienia RBAC i są montowane do kontenerów.
10. Do tworzenia reguł RBAC dla kont, należy robić to zgodnie z podejściem least privilege. Narzędzie audit2rbac znacząco ułatwia przypisywanie odpowiednich uprawnień RBAC dla utworzonych kont, zgodnie z tym podejściem.

RBAC (ang. Role Based Access Control)

Informacja:

1. RBAC to mechanizm kontroli dostępu, który za pomocą przypisanych ról do utworzonych kont, pozwala na jednoznaczne określenie uprawnień do wykonywania ściśle określonych czynności.
2. Przed implementacją reguł RBAC dla systemu docelowego, częstą praktyką jest przygotowanie macierzy dostępów.
3. W klastrze Kubernetes istnieje możliwość utworzenia czterech typów obiektów: Role, ClusterRole, RoleBinding oraz ClusterRoleBinding.
4. Grupa API rbac.authorization.k8s.io w klastrze Kubernetes, umożliwia dynamiczne konfigurowanie reguł.
5. Komponentem odpowiadającym w głównej części za autoryzację żądań jest Kube-apiserver.
6. W celu włączenia mechanizmu RBAC wymagane jest ustawienie wartości RBAC dla flagi authorization-mode w konfiguracji Kube-apiserver.
7. Po przypisaniu uprawnienia dla obiektu, nie ma możliwości zmiany roli obiektu w klastrze. W tym celu wymagane jest usunięcie RoleBinding dla obiektu. Dla automatyzacji czynności zmiany roli można wykorzystać gotowe polecenie: kubectl auth reconcile.
8. Dodawane reguły RBAC mają charakter addytywny oraz są one nakładane na użytkowników, grupy lub konta serwisowe.
9. W przypadku braku ustawienia konta serwisowego dla obiektu Pod, zostanie przypisanie konto typu Default o domyślnej roli, uniemożliwiającej wykonania jakiejkolwiek czynności na zasobach. Uprawnienia konta domyślnego mogą zostać zmienione.
10. Istnieje możliwość weryfikacji przydzielonych roli dla danego konta poprzez użycie polecenia: kubectl auth can-i.
11. Należy zwrócić uwagę na opcję Auto-reconciliation, która powoduje automatyczne uzupełnienie Role oraz RoleBinding w klastrze podczas startu Kube-apiserver. Ma to na celu ułatwienie procesu aktualizacji klastra do nowszych wersji, gdyż w przypadku braku wprowadzenia wymaganej zmiany, proces aktualizacji może spowodować jego niedostępność. Opcja jest domyślnie włączona, gdzie istnieje możliwość wyłączenia flagi poprzez ustawienie adnotacji rbac.authorization.kubernetes.io/autoupdate na wartość false.
12. Polecenie: kubectl get clusterroles system:discovery -o yaml jest wykorzystywane do sprawdzenia przyznawania anonimowego dostępu do wykrywania nonResourceURL.

Rekomendacja:

1. Podczas nadawania roli użytkownikom w klastrze należy przestrzegać zasady najmniejszego uprzywilejowania (ang. least privilege). Zasada ta polega na ustawieniu takich uprawnień dla konta, jakie są wymagane do wykonania zadań powiązanych z tym kontem. Istnieje możliwość sprecyzowania dla jakiego dokładnie typu oraz nazwy zasobu zostaną nadane uprawnienia.
2. Zagrożeniem dla klastra może okazać się możliwość zastosowania flagi --insecure-port dla Kube-apiserver. Opcja ta pozwala określić port dla którego pomijany jest etap uwierzytelnienia oraz autoryzacji, umożliwiając wykonywanie żądań API bez wymagania posiadania odpowiednich uprawnień. Należy upewnić się, iż flaga nie jest zastosowana.
3. W celu uproszczenia procesu wdrażania odpowiednich Roles oraz RolesBinding, można posłużyć się narzędziem audit2rbac. Jest to narzędzie, które na podstawie dokonanych prób odrzuconych żądań zapisanych w plikach audytowych komponentu Kube-apiserver, określa

wymagane uprawnienia oraz generuje konfigurację, stosując tym samym podejście least privilege.

Network Policy

Informacja:

1. W celu umożliwienia kontroli ruchu na poziomie warstwy trzeciej lub czwartej modelu ISO/OSI, często stosowanym rozwiązaniem są zasady sieciowe. Poprawnie zaimplementowane zasady sieciowe pozwalają na określenie dostępów poszczególnych obiektów Pod w klastrze dla konkretnych podmiotów sieciowych (Przykładowo: W przypadku kompromitacji aplikacji Frontend w klastrze, podczas gdy zasady sieciowe zostały wdrożone, atakujący nie ma możliwości bezpośredniego połączenia się z systemem bazy danych). Zasady sieciowe mają charakter addytywny. Implementacja zasad sieciowych pozwala na wdrożenie podejścia Implicit Deny, polegającego na zezwolenie tylko na takich ruch, jaki jest potrzebny do poprawnego działania aplikacji.
2. Stosowanie zasad sieciowych ma na celu ograniczenie możliwych do zastosowania przez atakującego technik, takich jak Lateral Movement lub Pivoting.
3. Zasady dzielą się na ograniczające ruch wejściowy oraz ograniczające ruch wyjściowy.
4. W celu implementacji zasad, wymagane jest przygotowanie pliku konfiguracyjnego, który utworzy obiekt typu NetworkPolicy.
5. W celu identyfikacji podmiotów z którymi obiekt Pod może się komunikować, wykorzystuje się selektory lub bloki adresów IP.
6. Domyślnie reguły ograniczające zezwalają w pełni na ruch wejściowy oraz wyjściowy sieci.
7. W celu skutecznego nałożenia zasad sieciowych, wymagane jest wcześniejsze wdrożenie mechanizmu CNI Plugin (np. Calico), który wspiera zasady sieciowe.
8. Podczas tworzenia zasad istnieje możliwość skorzystania z narzędzi do wizualizacji, takich jak networkpolicy.io.
9. W ramach testów, można skorzystać z wtyczek takich jak Krew-net-forward, co pozwala usprawnić proces rozwiązywania problemów sieciowych oraz zweryfikować poprawność komunikacji pomiędzy komponentami. Przydatnym narzędziem do weryfikacji poprawności nałożonych zasad sieciowych może okazać się również Netassert.

Rekomendacja:

1. Należy wdrożyć reguły sieciowe dla zasobów klastra.

Pod Security Admission

Informacja:

1. Bazując na zestawach profili najlepszych praktyk Pod Security Standards, Pod Security Admission umożliwia implementację profili wymuszających zastosowanie właściwych standardów bezpieczeństwa. Jest to obszar konfiguracji, który pozwala zastąpić implementację przestarzałych reguł Pod Security Policy. Pod Security Admission jest rozwiązaniem, które może zapobiec wdrożeniu konfiguracji niezgodnych z normami bezpieczeństwa, jednocześnie standaryzując podejście tworzenia obiektów takich jak Pod. (Przykładowo: Atakujący jest w stanie uruchomić Pod typu uprzywilejowanego, który następnie umożliwi dostęp do plików maszyny hosta).
2. Polega na tworzeniu ograniczeń dla obiektów Pod. Dzięki implementacji, istnieje możliwość przykładowo zablokowania możliwości tworzenia obiektu Pod o typie uprzywilejowanym

przez użytkowników klastra lub uniemożliwienie tworzenia wolumenów w oparciu o typ HostPath.

3. Wdrożona wersja 1.23 klastra umożliwia korzystanie z funkcjonalności Pod Security Admission w formie beta i jest domyślnie włączona.

4. Implementacja Pod Security Admission umożliwia przypisanie dla obiektów Pod profilu:
    - **Privileged** – brak ograniczeń, umożliwiając większość możliwych eskalacji. Profil jest najczęściej wdrażany dla infrastruktury samego systemu o zaufanych użytkownikach.
    - **Baseline** – chroni przed podstawową ilością możliwych eskalacji. Profil jest przeznaczony głównie dla aplikacji o mniejszym poziomie ryzyka.
    - **Restricted** – wysoki poziom ochrony przed możliwymi eskalacjami. Profil jest przeznaczony dla aplikacji o wysokim poziomie ryzyka.
5. Profile mogą być implementowane poprzez ustawienie etykiety dla przestrzeni nazw lub AdmissionConfiguration. W przypadku AdmissionConfiguration etykieta jest ustawiana w kontekście klastra.
6. Występują trzy podstawowe poziomy Pod Security Admission: enforce, audit oraz warn. Poziom enforce powoduje odrzucenie danego Pod w przypadku niezgodności. Dla poziomu audit nastąpi odpowiednie zalogowanie informacji do logów audytowych, bez ingerencji w tworzenie Pod, a warn wyświetlenie stosownego komunikatu.
7. Poziomy takie jak warn oraz audit posiadają dodatkowe zastosowanie umożlwiające sprawdzenie czy nowsza wersja klastra Kubernetes zmieni sposób działania reguł, bez realnej ingerencji w sposób działania aplikacji. Należy więc zwrócić uwagę, iż zmiana wersji klastra może wpływać na sposób działania reguł, co należy rozważyć podczas przeprowadzania procesu aktualizacji.
8. Pod security admission umożliwia również wdrożenie wyjątków w zakresie użytkowników, klas typu Runtime oraz przestrzeni nazw. Opcja może być zastosowana w przypadku sytuacji wyjątkowych, kiedy przestrzeń nazw narzuca pewne ograniczenia, których konkretny obiekt Pod nie jest w stanie spełnić.
9. Implementacja Pod Security Admission wymaga stopniowego nakładania reguł dla każdej z przestrzeni nazw oraz sprawdzania możliwego do wdrożenia profilu, tak aby wdrożony profil ograniczał możliwe funkcjonalności w jak największym stopniu.
10. Istnieje możliwość zastosowania dla danego Pod kilku poziomów profili jednocześnie, co pozwala dodatkowo określić możliwość zmiany profilu, bez ingerencji w sposób działania obiektu.
11. Dla niektórych starszych wersji klastra nie ma możliwości skorzystania domyślnie z Pod Security Admission Plugin. W takim przypadku istnieje możliwość implementacji Pod Security Admission poprzez Validating Admission Webhook.
12. Istnieją implementacje podobnych rozwiązań dla zewnętrznego oprogramowania. Jeżeli pojawi się przypadek zastosowania, w którym Pod Security Admission dostarczane do Kubernetes domyślnie nie będzie zadawalające pod kątem oferowanych funkcjonalności, wykorzystanie zewnętrznego oprogramowania może okazać się najlepszym z możliwych rozwiązań. Do powszechnie stosowanych zewnętrznych dostawców można przykładowo zaliczyć Kyverno lub Gatekeeper.
13. Pod Security Admission lub Pod Security Policy to nie to samo co Pod Security Context. Pod Security Admission pozwala tylko ograniczyć wartości jakie są ustawiane w Pod Security Context.

Rekomendacja:

1. Należy wdrożyć mechanizm standaryzujący sposób tworzenia lub/i monitorowania obiektów w klastrze Kubernetes (wykorzystując przykładowo rozwiązania takie jak Pod Security Admission).

Security Context

Informacja:

1. Kontekst bezpieczeństwa stosuje się w celu wdrożenia kontroli dostępów oraz ograniczeń dla obiektów typu Pod (Kontenerów). Kontekst bezpieczeństwa pozwala na zapewnienie izolacji pomiędzy uruchamianym kontenerem w obiekcie Pod, a maszyną Host (przykładowo: uruchomienie kontenera z użytkownikiem jako root).
2. Kontekst zabezpieczeń jest implementowany poprzez konfigurację opcji securityContext dla Pod wybranej aplikacji. Ustawienie wybranej opcji dla Pod powoduje nałożenie konfiguracji dla wszystkich kontenerów związanych z Pod.
3. Do najważniejszych opcji konfiguracyjnych dla SecurityContext zalicza się:

    - **Uznaniowa metoda kontroli dostępu** (ang. Discretionary Access Control) – pozwala na określenie dostępów dla obiektów bazując na parametrach uid oraz gid. Kontekst zabezpieczeń implementuje podstawy podejścia DAC w celu kontroli uprawnień jakie są przydzielane do poszczególnych plików oraz procesów uruchamianych w kontenerach. Przykładowo, jeżeli dla utworzonego obrazu aplikacji nie został zdefiniowany parametr uid użytkownika, dla którego zostanie uruchomiony proces, użytkownikiem dla procesu może być użytkownik Root. W celu ograniczenia wystąpienia takiego ryzyka można skorzystać z takich opcji jak runAsUser, runAsGroup oraz fsGroup.
    - **Ustawienie eskalacji uprawnień** – przy pomocy konfiguracji flagi allowPrivilegeEscalation istnieje możliwość kontroli, czy uruchomiony proces ma możliwość otrzymania większej liczby uprawnień niż jego rodzic. Flaga ta jest ustawiona na wartość prawdziwą, jeśli kontener jest uruchamiany jako uprzywilejowany lub posiada funkcjonalność CAP_SYS_ADMIN.
    - **Ustawienie montowania systemu plików Root kontenera tylko do odczytu** – przy pomocy flagi readOnlyRootFilesystem istnieje możliwość określenia czy kontener ma odpowiednie uprawnienia do zapisu dla systemu plików Root kontenera.
    - **Selinux** – umożliwia ustawienie etykiety dla obiektów. Kontekst dla Selinux jest nakładany dla kontenera. W przypadku braku określenia wartości, środowisko uruchomieniowe automatycznie przypisze kontekst Selinux dla każdego z kontenerów. Ustawienie Selinux jest bardzo ważne, gdyż w dużej liczbie przypadków uniemożliwia atakującemu ucieczkę z kontenera. Równie popularnym odpowiednikiem dla Selinux jest Apparmor, który jest często stosowany na dystrybucjach Debian. Warto zwrócić uwagę, iż do skutecznego działania tej funkcjonalności wymagane jest zainstalowanie modułu na maszynie Host.
    - **Kontrola Linux Capability** – umożliwia kontrolę uprawnień dla uruchamianych procesów poprzez określenie możliwych dla nich funkcjonalności. Domyślnie procesy nie posiadają żadnych dodatkowych funkcjonalności, jednak ustawienie to nadal zależy od konfiguracji środowiska uruchomieniowego. Zaleca się wyłączenie wszystkich funkcjonalności poprzez ustawienie opcji drop: - all. Dodatkowe uprawnienia są wymagane tylko w przypadku zadań o poziomie systemowym. Warto podkreślić, iż każdy z procesów w celu otrzymania dostępów do pamięci, dysku lub urządzeń musi skomunikować się z jądrem Kernel, gdzie całość jest kontrolowana głównie przy pomocy dostępów do plików. Ustawione funkcjonalności można zweryfikować poprzez wykonanie polecenia cat /proc/1/status wewnątrz kontenera.
    - **Seccomp** – umożliwia filtrowanie wywołań systemowych. Zastosowanie ustawienia ma na celu ograniczenie dostępu z poziomu przestrzeni użytkownika do jądra Kernel systemu operacyjnego. Opcję wdraża się poprzez ustawienie pola SeccompProfile. Zaleca się, aby

ustawienie nie było wdrożone z wartością unconfined. W przypadku zastosowania Docker, opcja ta przykładowo domyślnie ogranicza około 44 wywołania systemowe z ponad 300. W przypadku dużej części aplikacji taka konfiguracja może być niewystarczająca, gdyż duża część dostępnych wywołań systemowych jest nie używana. Podejście to doskonale odzwierciedla kierunek rozwoju bezpieczeństwa środowisk opartych na piaskownicy (ang. Sandbox). Dodatkowe narzędzia takie jak Strace pozwalają na dokładne zbadanie z jakich wywołań systemowych korzystają konkretne aplikacje. Zastosowanie wartości runtime/default może być wystarczające dla większości przypadków. Istnieje możliwość generowania własnych profili Seccomp, jednak w takim przypadku należy wziąć pod uwagę ewentualny zmienny charakter dostarczanych aplikacji.

4. Kontekst bezpieczeństwa wymaga dodatkowej kontroli, w szczególności podczas procesu aktualizacji wersji klastra. Należy zwrócić uwagę, iż stosowanie niektórych opcji takich jak Selinux lub AppArmor, może wymagać spójności rodzajów oraz wersji systemów operacyjnych na poszczególnych węzłach.

Rekomendacja:

1. Należy wdrożyć ustawienia lub zweryfikować ustawienia:

    - **DAC** – Należy skonfigurować opcje runAsUser, runAsGroup oraz fsGroup.
    - **Eskalacja uprawnień** – należy upewnić się, iż opcja eskalacji uprawnień jest wyłączona
    - **Ustawienie montowania systemu plików Root kontenera tylko do odczytu** – należy uniemożliwić zapis do systemu plików Root kontenera.
    - **Selinux** – Należy skonfigurować Selinux/Apparmor na instancjach host oraz upewnić się, iż CRE (ang. Container Runtime Environment) w klastrze ma aktywną opcję – selinux-enabled.
    - **Kontrola Linux Capabilit**y – Zaleca się ustawienie opcji drop: - all
    - **Seccomp** – należy upewnić się, iż stosowana jest wartość runtime/default (domyślna) lub bardziej restrykcyjna (polecenie crictl inspect).

Image Signature Verification

Informacja:

1. Dla atakującego prostszą czynnością jest umieszczenie podrobionego obrazu w repozytorium, tak aby został on pobrany bez dodatkowej weryfikacji przez środowisko docelowe niż aby próbował dostać się do systemu bezpośrednio. Funkcjonalności rozwiązań takich jak Kubernetes, opierają się w dużej mierze na automatyzacji czynności związanych z pobieraniem obrazów oraz uruchamianiem kontenerów. Powoduje to, iż potencjalny atakujący ma ułatwione zadanie, szczególnie jeśli dostęp do repozytorium nie jest ściśle ograniczony. W celu zachowania integralności oraz uzyskania poświadczenia o wiarygodności danego obiektu wygenerowanego przez programistę, należy skorzystać z podpisu cyfrowego obrazów lub artefaktów.
2. Sigstore jest zbiorem narzędzi dostarczającym ustandaryzowane podejście dla bezpieczeństwa łańcuchu dostaw oprogramowania. Rozwiązanie w stosunku do wersji pierwszej Notary posiada między innymi odmienny sposób przechowywania wygenerowanych sygnatur, poprzez ich bezpośrednie umieszczenie w repozytorium. Do

przygotowania oraz zarządzania mechanizmem podpisów cyfrowych przy użyciu Sigstore, wykorzystuje się najczęściej następujące aplikacje:

- **Cosign** – obejmuje czynności takie jak: generowanie klucza prywatnego oraz publicznego, podpisywanie cyfrowe obrazów oraz publikacja wygenerowanej sygnatury w docelowym repozytorium, wyszukiwanie sygnatur dla obrazu oraz ich weryfikacja przy użyciu klucza publicznego.
- **Rekor** – obsługuje funkcjonalności związane z prowadzeniem bazy danych metadanych, umożliwiając użytkownikom zapisanie do tylko inkrementowanego systemu, podpisanych metadanych w niezmiennym rekordzie. Narzędzie ma na celu zapewnienie publicznego dostępu do informacji o utworzonych sygnaturach, co może zostać wykorzystane do ich monitoringu. Pozwala ono również na wprowadzenie dwustopniowej weryfikacji wskazanego obrazu, poprzez sprawdzenie z sygnaturą w repozytorium oraz niezmiennym rekordem Rekor.
- **Fulcio** – narzędzie pełniące rolę Root CA (ang. Certificate Authority), pozwalające na wdrożenie podejścia KeyLess. Dostarcza możliwość podpisywania certyfikatów x509, przy zastosowaniu rozwiązania OpenID Connect. Cechą ustanawianych certyfikatów jest ich krótki czas ważności. Podejście pozwala zaimplementować uproszczony system, który rozwiązuje problemy związane z zarządzaniem wygenerowanymi certyfikatami, nie wymagając tym samym od osoby podpisującej dodatkowego nakładu pracy związanego z przechowywaniem wygenerowanych kluczy.

3. Dla środowiska opartego o klaster Kubernetes, w celu implementacji automatycznej weryfikacji podpisów cyfrowych, istnieje możliwość wykorzystania gotowych rozwiązań. Do jednych z nich zalicza się Kyverno oraz Connaisseur. W przypadku Kyverno implementacja polega na wdrożeniu reguły, która przed utworzeniem obiektów zweryfikuje zgodność przy pomocy narzędzia Cosign. Connaisseur opiera się na implementacji Admission Controller w celu weryfikacji sygnatur obrazów. Narzędzie również uniemożliwi utworzenie obiektów, w przypadku wykrycia niezgodności podczas weryfikacji przy pomocy Notary lub Cosign.

Rekomendacja:

1. Należy rozważyć zaimplementowanie jednego z dostępnych rozwiązań takich jak Sigstore lub Notary (w wersji 2 – obecnie niedostępnej) dla procesu Image Signature Verification.
2. Jeśli jest to możliwe, dla podejścia Sigstore zaleca się wykorzystanie metody KeyLess.

Zastosowanie technologii Sandbox

Informacja:

1. Piaskownica (ang. Sandbox) jest mechanizmem służącym do izolacji uruchamianych zasobów od pozostałej części środowiska. Izolacja danego zasobu pozwala na ochronę pozostałej części środowiska przed zagrożeniem występującym wewnątrz izolowanego obiektu (Przykładowo: 0-day).
2. Zastosowanie modelu piaskownicy pozwala na izolację uruchamianej aplikacji od interfejsu wywołań systemowych Host.
3. Piaskownica jest implementowana poprzez wykorzystanie mechanizmów takich jak ograniczenie funkcjonalności systemu, wirtualizację części sprzętowej oraz aplikacji, konteneryzację, wdrożenie odpowiednich modułów Kernel, wdrożenie LSM, jak i również poprzez zastosowanie dedykowanych sterowników sprzętowych.

4. W porównaniu do procesu konteneryzacji, piaskownica wykorzystuje emulację własnych funkcjonalności Kernel, minimalizując tym samym ilość możliwych do wykonania wywołań systemowych w kierunku Kernel Host.
5. Rynek technologii dostarcza wiele darmowych rozwiązań umożlwiających implementację piaskownicy dla klastra Kubernetes. Są to przeważnie rozwiązania, które implementują specyfikację OCI (ang. Open Container Initiative Runtimes), co umożliwia ich łatwą integrację z istniejącymi już środowiskami.
6. Można wyróżnić dwa przykładowe rozwiązania implementujące model piaskownicy:

    - **GVisor** – jest rozwiązaniem korzystającym z trybu KVM lub Ptrace. Mechanizm bazuje na uruchomieniu własnego jądra Kernel w przestrzeni użytkownika, które następnie przesyła wywołania systemowe do Kernel Host poprzez proces Sentry. Rozwiązanie jest również popularnie wykorzystywane na platformie GCP (ang. Google Cloud Platform).
    - **Kata containers** – jest to środowisko uruchomieniowe w którym uruchamiana jest osobna zminimalizowana oraz zoptymalizowana wersja maszyny wirtualnej, a wewnątrz niej docelowy kontener. Kontener jest uruchamiany przy wykorzystaniu jądra Kernel systemu gościa. Do wirtualizacji wykorzystywany jest tryb KVM, gdzie urządzenia emulowane są przy pomocy QEMU, QEMU-lite, Firecracker lub NEMU. Bezpośredni dostęp do sieci Host dla działających kontenerów jest nie możliwy. Dla rozwiązania implementacja SELinux oraz AppArmor może nie być wspierana.

7. Wymienione rozwiązania mogą być zaimplementowane w klastrze Kubernetes poprzez konfigurację środowiska uruchomieniowego oraz utworzenie obiektu RuntimeClass.
8. Oprócz zmniejszenia ryzyka kompromitacji, implementacja może powodować zmniejszenie wydajności aplikacji. Dodatkowo, jeżeli wybrana piaskownica nie wspiera wywołań systemowych wykorzystywanych przez aplikację, może to spowodować niestabilność działania usług.
9. Klaster Kubernetes wspiera możliwość stosowania różnych typów RuntimeClass jednocześnie na pojedynczym środowisku. Pozwala to na odpowiednie pogrupowanie zasobów ze względu na występujące ryzyko oraz zastosowanie odrębnego podejścia dla każdej z kategorii.

Rekomendacja:

1. Należy rozważyć wdrożenie modelu piaskownicy (ang. Sandbox) dla działających aplikacji klastra (przykładowo dla aplikacji szczególnie narażających klaster na kompromitację).

Łańcuch dostaw oprogramowania

Informacja:

1. W kontekście łańcucha dostaw dla oprogramowania najtrudniejszą czynnością jest identyfikacja wystąpień zagrożeń dla danego środowiska. Ze względu na wysoki stopień złożoności większości architektury systemów oraz powiązań w zależnościach aplikacji, w momencie wystąpienia zagrożenia ciężko jest manualnie stwierdzić, czy środowisko jest realnie zagrożone. Do tego celu wdrażane są narzędzia umożliwiające automatyczne śledzenie zależności występujących w aplikacji oraz przeprowadzanie skanów podatności.
2. Skany podatności dla klastra w poszukiwaniu podatności w łańcucha dostaw można przeprowadzać przy użyciu narzędzi takich jak Dependency Check, Trivy lub Clair.

Rekomendacja:

1. Należy rozważyć implementację rozwiązań do detekcji występujących podatności w procesie CI/CD, uniemożliwiając wdrożenie aplikacji w przypadku wykrycia potencjalnych nieakceptowalnego ryzyka podatności.
2. Należy rozważyć wdrożenie rozwiązania Renovate do automatycznego aktualizowania zależności występujących w projektach.

System kontroli wersji

Informacja:

1. W dużej mierze bezpieczeństwo klastra zależy od sposobu przechowywania plików konfiguracyjnych oraz ich synchronizacji,
2. Podejście GitOps jest powiązane z systemem kontroli wersji. Implementacja GitOps wprowadza zestaw dobrych praktyk poprawiających bezpieczeństwo klastra, a w tym ograniczenie wymagania logowania się przez osoby wdrażające aplikacje bezpośrednio na klastrze. W podejściu tym zamiast przyznawać uprawnienia dla poszczególnych użytkowników, wykorzystywane są uprawnienia automatycznego mechanizmu do naniesienia zmian na klastrze. (Podejście można łatwo wdrożyć przez narzędzia takie jak Flux lub Argo CD).
3. Kontrola dostępu do projektu – zagadnienie obejmuje weryfikację uprawnień widoczności projektu oraz jego sekcji, tak aby ograniczyć uprawnienia osób powiązanych z danym projektem.
4. Kontrola dostępu do zmiennych (w zależności od gałęzi i użytkowników) – dla projektów często niezbędną czynnością jest inicjowanie procesów uwierzytelnienia, do którego potrzebują właściwych danych. Dane muszą być przechowywane w taki sposób, aby uniemożliwiły osobom nieuprawnionym odczyt informacji.
5. Właściwy sposób logowania informacji – w przypadku uruchamiania procesów CI/CD, należy zwrócić uwagę na logi wygenerowane przez proces. Odpowiednia konfiguracja powinna zapobiec logowaniu informacji takich jak dane wymagane do uwierzytelnienia oraz wszelkie inne kluczowe informacje o kategorii niepublicznej.
6. Wymaganie uwierzytelnienia wieloskładnikowego – dodatkowym zabezpieczeniem dla procesu logowania jest wprowadzenie wieloskładnikowego uwierzytelnienia, co pozwala zapobiec różnym rodzajom ataków.
7. Monitoring zdarzeń dla wdrożeń – informacje dotyczące wykonywanych czynności dla konkretnych gałęzi projektu są odpowiednio monitorowane, gdzie gałęzie środowisk z danymi produkcyjnymi, są ściśle kontrolowane, a każda nieprawidłowość zgłaszana.
8. Kontrola wersji obrazów wykorzystywanych do wdrożeń – klaster wykorzystując wybrane repozytorium pobiera odpowiednie obrazy dla kontenerów, gdzie każda nieautoryzowana czynność związana z umieszczaniem kolejnych wersji obrazów w repozytorium, zostaje automatycznie wykryta oraz zgłoszona.
9. Checkov dla testów bezpieczeństwa typu SAST, które może być sukcesywnie wdrożone do analizy konfiguracji w systemie kontroli wersji. Dla domyślnej konfiguracji klastra oraz obrazów, narzędzie pozwala wykryć problemy z nazewnictwem obrazów, polityką pobierania obrazów, brakiem zdefiniowania wymaganych zasobów oraz limitów, brak ograniczenia funkcjonalności NET_RAW, wyłączenia automatycznego montowania tokenu konta serwisowego, brak konfiguracji Liveness Probe, nieodpowiednie ustawienie dla UID oraz GID użytkownika wewnątrz kontenera, brak ustawienia kontekstu bezpieczeństwa oraz profilu Seccomp, wysoki poziom entropii dla ciągu znaków BASE64 oraz kilka dodatkowych uwag stanowiących wskazówki dotyczące konfiguracji. Skaner poprawnie wskazuje problemy występujące w konfiguracji przygotowanej dla przykładowych aplikacji wdrożonych w klastrze Kubernetes.

Rekomendacja:

1. Proces zarządzania zmianami (ang. Patch management) powinien zostać wdrożony dla projektu tak, aby łańcuch dostaw dla oprogramowania był ściśle kontrolowany.
2. Zalecane jest zadbanie o poprawną obsługę/wdrożenie kroków od Info 3. do Info 8..
3. Dostarczane środowiska powinny być ściśle kontrolowane pod względem dokonywanych zmian, poprzez wdrożenie procesu zarządzania zmianami (ang. Change management). (Przykładowo: Atakujący może przykładowo podmienić obraz w repozytorium z obrazami, gdzie, jeśli administrator nie używa funkcji skrótu do wygenerowania nazwy obrazu bazując na jego zawartości, może to spowodować dodanie przez atakującego szkodliwej zawartości do obrazu).
4. Należy zadbać o separację obowiązków (ang. separation of duties), tak aby zapobiec możliwości dokonania zmiany przez pojedynczą osobę.
5. Należy rozważyć wdrożenie podejścia GitOps dla wdrażania aplikacji do klastra.
6. Należy wdrożyć testy bezpieczeństwa dla procesu CI/CD (w tym testy typu SAST, przy użyciu przykładowo Checkov).

Runtime Security

Informacja:

1. Runtime Security jest pojęciem odnoszącym się do bezpieczeństwa uruchomionych zasobów w danym środowisku. Zagadnienie w przypadku klastra Kubernetes sprowadza się między innymi do przeprowadzania testów bezpieczeństwa bezpośrednio na działających węzłach, włączając tym samym w zakres kontenery aplikacji. Testy bezpieczeństwa mogą obejmować narzędzia kategorii SAST, DAST, IAST oraz RASP.
2. Niektóre rozwiązania implementujące idee monitoringu zasobów w ramach Runtime Security, pozwalają na implementację dodatkowej warstwy detekcji lub prewencji przed możliwymi zagrożeniami, poprzez odpowiednie monitorowanie występujących sygnałów na działającym środowisku.
3. Rozwiązaniem umożliwiającym detekcję incydentów bezpieczeństwa zgodnych z podejściem Runtime Security jest Falco. Rozwiązanie instaluje się na poszczególnych węzłach klastra bezpośrednio na maszynie Host lub jako typ Daemonset w klastrze. Instalacja jako Daemonset wymaga, aby kontenery zostały uruchomione w trybie uprzywilejowanym – co wnosi dodatkowe ryzyko. Oprogramowanie to składa się z trzech części: programu przestrzeni użytkownika, konfiguracji oraz sterownika. Narzędzie analizuje wywołania systemowe w celu zapewnienia odpowiedniego poziomu monitorowania zdarzeń Host, klastra Kubernetes oraz działających w klastrze kontenerów. W podejściu rozwiązań chmurowych można skorzystać z gotowych rozwiązań takich jak Snyk lub Aqua Platform.

Rekomendacja:

1. Należy wdrożyć rozwiązanie pozwalających na monitorowanie wywołań systemowych dla środowisk skonteneryzowanych (przykładowo poprzez implementacja narzędzia Falco, Snyk lub Aqua Platform).