Salesforce Kubernetes Control Plane Vulnerability Assessment

Security Assessment Report

Prepared for Salesforce.com, Inc.
August 3, 2021 (version 1.1)

Project Team:

Technical Testing	Tom Steele and Joshua Dow
Technical Editing	Nathan Keltner and Lacey Kasten
Project Management	Sara Bettes

Table of Contents

Engagement Overview

Assessment Components and Objectives

Salesforce.com, Inc. ("Salesforce") recently engaged Atredis Partners ("Atredis") to perform a Vulnerability Assessment of the Kubernetes Control Plane.

Testing was performed from April 19 through May 14, 2021 by Tom Steele and Joshua Dow of the Atredis Partners team, with Sara Bettes providing project management and delivery oversight. For Atredis Partners' assessment methodology, please see Appendix I of this document, and for team biographies, please see Appendix II. Specific testing components and testing tasks are included below.

COMPONENT	ENGAGEMENT TASKS
Salesforce Kubernetes Control Plane Vulnerability Assessment	

Assessment Targets	• Kubernetes Control Plane
	  • Latest Kuberentes release
	  • Standard deployment via kubeadm
	  • kube-apiserver
	  • kubelet
	  • kube-proxy
	  • etcd
	  • kube-scheduler
	  • kube-controller-manager
	  • TLS and token-based authentication schemes
	  • RBAC authorization
	  • Multi-tenant simulation environment
	  • Simulated node compromise
	  • CoreDNS configuration

Assessment Tasks	• Source-Assisted Vulnerability Assessment and Penetration Testing • Control Plane Threat Modeling • DFD and Attack Surface Mapping
Reporting and Analysis	

Analysis and Deliverables	• Status Reporting and Realtime Communication
	  • Comprehensive Engagement Deliverables
	  • DFD
	  • Threat Model
	  • Final deliverable with executive summary, overview of findings, and low-level finding details
	  • Engagement Outbrief

Executive Summary

Atredis Partners conducted an assessment of the Kubernetes control plane with a focus on multi-tenancy. Multi-tenancy within the same enterprise, such as development, test, production teams sharing the same cluster has been discussed extensively in the Kubernetes space. Software as a Service (SaaS) solutions, where applications are deployed for different tenants within the same cluster has been discussed frequently as well. The conclusion of these discussions is that multi-tenancy is satisfactory solution depending on the risk appetite of the organization.

Multi-tenancy as it concerned for this project, which is sometimes referred to as Kubernetes as a Service, where each tenant shares the same control plane but is not part of the same organization, has received some discussion but has not been extensively researched.

The Kubernetes Multi-Tenancy Working Group recently published an update for common tenancy models along with emerging products that are designed to resolve potential issues. The three models discussed included:

- Namespace as a Service – tenant share a cluster and nodes and workloads are only restricted by namespace
- Clusters as a Service – each tenant is provisioned their own cluster
- Control planes as a Service – each tenant is provided a control plane but share worker nodes

The model proposed for testing does not fit any of these, as the proposed solution shares a control plane but isolates the worker nodes. This can be done using node taints. The caveat with taints is that a tenant cannot have direct access to the control plane. This fits the proposed solution as an intermediary service would communicate with the control plane on behalf of the tenant.

Atredis Partners created a threat model operating under the assumption that an attacker has compromised a node and tenants do not have direct access to the control plane. Threat scenarios drove the vulnerability assessment testing process to prove mitigations, examine impact, and document residual risk.

Key Conclusions

No significant findings were identified in the Kubernetes control plane or data plane components. Controls that would be critical to the security of cluster and multi-tenancy were tested extensively using a combination of code review and dynamic testing.

Several low severity findings were identified. None of these findings would impede a solution from moving forward. Most of these findings can be resolved by configuration changes and designing systems and processes to avoid their potential impact.

Despite these positive results, the radius of a security event that results in complete or partial compromise of the kube-apiserver or etc. may not be an acceptable risk for the Salesforce environment. Once compromised, post-exploitation, data exfiltration, and persistence across the cluster is straight-forward and may be difficult to detect. Investing time and resources into enhancing Clusters as a Service initiatives should be investigated as an alternative to the proposed solution.

Future work involves reviews of Container Network Interface (CNI) implementations and the story of services in the proposed solutions. Of particular interest will be confused deputy attacks that could allow a tenant to expose the services of another tenant. The use of namespaces and network policies adequately prevented these scenarios in the test environment, but the use of more enterprise grade solutions involving external load balancers and more complex networking is an area for additional research.

Findings Summary

In performing testing for this assessment, Atredis Partners identified **(4) low** severity findings. No medium, high, or critical severity findings were identified. As stated earlier, none of these issues constitute a potential for direct compromise.

Atredis defines vulnerability severity ranking as follows:

- **Critical:** These vulnerabilities expose systems and applications to immediate threat of compromise by a dedicated or opportunistic attacker.
- **High:** These vulnerabilities entail greater effort for attackers to exploit and may result in successful network compromise within a relatively short time.
- **Medium:** These vulnerabilities may not lead to network compromise but could be leveraged by attackers to attack other systems or applications components or be chained together with multiple medium findings to constitute a successful compromise.
- **Low:** These vulnerabilities are largely concerned with improper disclosure of information and should be resolved. They may provide attackers with important information that could lead to additional attack vectors or lower the level of effort necessary to exploit a system.

Environment Setup

Atredis Partners created a test environment using kubeadm defaults consisting of the following:

- Ubuntu Linux Operating System
- 1 control plane node
- 2 data plane nodes
- containerd container runtime
- Kubernetes version 1.21.0
- CoreDNS
- Weave CNI

Changes to configurations were made when required in order to test possible deficient configurations or prove security controls that were the result of improvements over the default configuration.

A combination of namespaces and node taints were used to separate hypothetical tenants in the cluster. Each tenant was provided a single node. Taints were created using the following:

```
$ kubectl taint nodes nodea tentant=clienta:NoSchedule
```

When creating a Pod for a tenant, tolerations were used to ensure workloads were scheduled on the appropriate node:

```
apiVersion: v1
kind: Pod
metadata:
  name: echo
spec:
  containers:
  - name: echo
    image: k8s.gcr.io/echoserver:1.4
    imagePullPolicy: IfNotPresent
    volumeMounts:
    - name: secretvolume
      mountPath: "/etc/namespace-name"
      readOnly: true
  volumes:
  - name: secretvolume
    secret:
      secretName: test-secret

  tolerations:
  - key: "tenant"
    operator: "Equal"
    value: "clienta"
    effect: "NoSchedule"
```

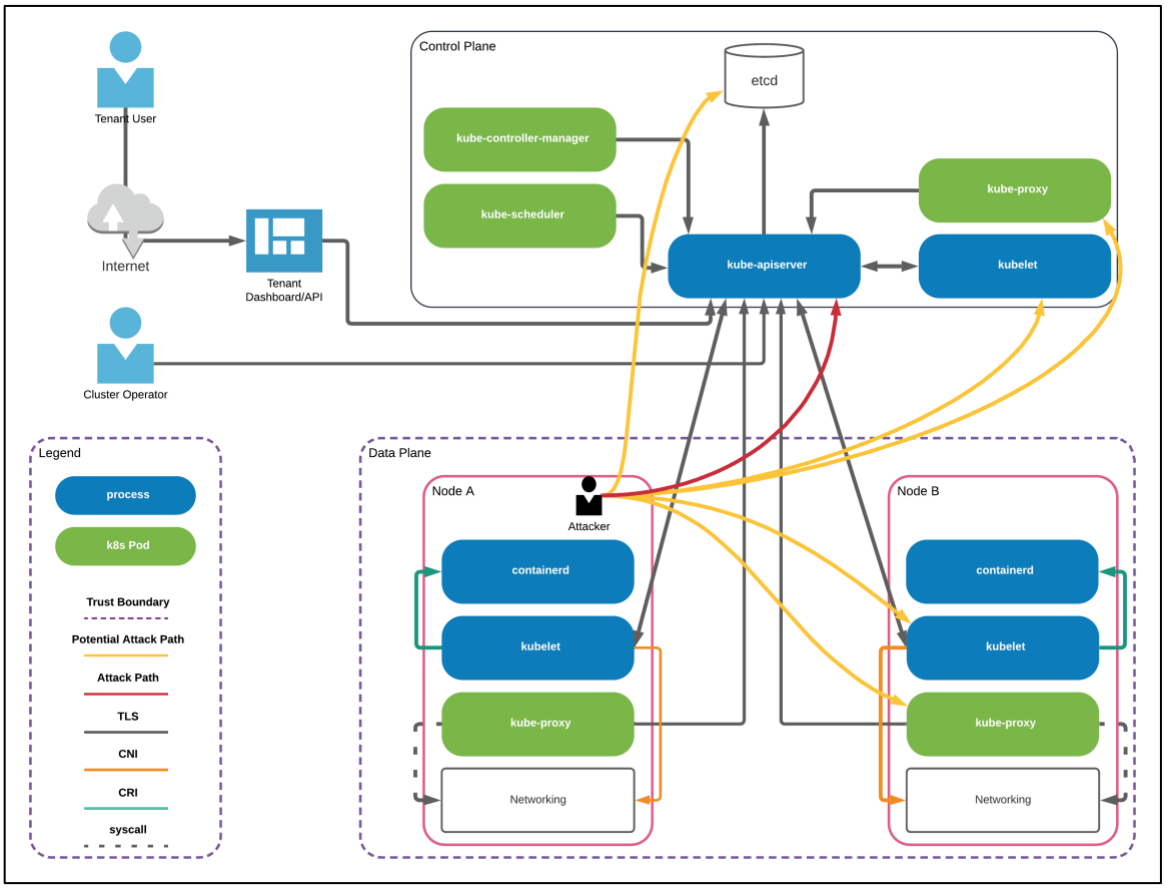
Attack Surface Review

As discussed, this focus of this assessment centered on attacks from a compromised node. An attacker with uninhibited access to a node would be able to interact and gain access to authentication material for the following components:

- kubelet
- kube-proxy
- CNI plugin(s)
- container runtime (i.e., Docker)

Container Network Interface (CNI) plugin(s) and the container runtime were not in scope for this assessment.

When discussing the attack surface of these components this assessment will separate the data flows into two major directions, Node to Node and Node to API. The Data Flow Diagram (DFD) below shows these potential attack paths. Note that when discussing Node-to-Node that control plane nodes are a potential target along with data plane nodes as they may run the same components.



Kubernetes DFD

Node to Node

kubelet

A Kubernetes node runs an agent, kubelet, which oversees managing workloads on a node. Kubelet is a Go binary that continuously communicates with kube-apiserver and serves its own HTTP API. The HTTP listener is typically bound to the local network interface as it needs to be available to kube-apiserver. It is possible to restrict access to the network and use SSH tunneling, but this is not typical. The HTTP listener uses Transport Layer Security (TLS) to secure network communications. TLS is required, as shown below:

```
# curl -ki http://nodeb:10250/pods
HTTP/1.0 400 Bad Request

Client sent an HTTP request to an HTTPS server.
```

HTTP to kubelet Request Response

The API exposes several endpoints that can be leveraged by an attacker to affect the security of a Pod on a node. These endpoints and attacks scenarios are well known and have been discussed in previous assessments, these include executing commands, reading and writing files, and exposing services on a Pod.

Authentication

By default, there is no authentication configured for kubelet and anonymous access must be explicitly disabled. To prevent attacks against kubelet, authentication must be used. There are two methods of authentication available:

- X509 client certificates
- API bearer tokens

X509 uses a certificate to identify a user. Incoming TLS connections must have a valid client certificate in order to be authenticated.

API bearer token authentication sends incoming bearer tokens to kube-apiserver, which in turn uses the `TokenReview` API. The `TokenReview` API uses a webhook to call an external service to determine the identity of a user based on the incoming token. Service account tokens can also be used for authentication.

Allowing bearer tokens to be used for authentication exposes sibling nodes and kube-apiserver to an attack that is not possible with X509. Due to inherent traits of a client certificate and associated private key it is not possible for an attacker to intercept usable authentication material from an incoming connection.

For example, if an attacker was able to decrypt TLS secured HTTP requests or use a patched kubelet server and intercept an incoming client certificate, they would not be able to use that client certificate to authenticate to kubelet running on another node, as they do not have access to the associated private key. The same scenario using a bearer token would allow an attacker to authenticate to kubelet or kube-apiserver itself.

This method of authentication is configured by default by kubeadm and does present an unnecessary risk, particularly in a multi-tenant environment. A service account should not be authenticating directly to kubelet and kube-apiserver uses X509 for authentication as specified in the `kubelet-client-certificate` and `kubelet-client-key` configuration flags. Removing this form of authentication is recommended.

Authorization

An attacker who has compromised a node would have access to kubelet's X509 certificate and key and would be able to authenticate to a sibling kubelet as that node. For that reason, authorization must be used in conjunction with authentication.

The following request shows `nodea` successfully authenticating (using an X509 certificate and key) to `nodeb` and failing the requisite authorization request.

```
nodea:~$ sudo curl -ki https://nodeb:10250/pods --cert /var/lib/kubelet/pki/kubelet-client-
current.pem --key /var/lib/kubelet/pki/kubelet-client-current.pem
HTTP/2 403
content-type: text/plain; charset=utf-8
content-length: 79
date: Fri, 07 May 2021 19:13:19 GMT

Forbidden (user=system:node:nodea, verb=get, resource=nodes, subresource=proxy)
```

Node-to-Node Authentication and Authorization Request Response

Authorization by default is set to `AlwaysAllow`. The other authorization option available is `Webhook`, which delegates authorization to kube-apiserver using a `SubjectAccessReview`. This authorization setting will be used in conjunction with `Node Authorizer` and `NodeRestriction` plugins to prevent access to a kubelet on a sibling node using kubelet's credentials.

As these authorization mechanisms are impetrative to mitigating several threats they are discussed in their own section. When authenticating as a user that is not a node or as a service account, Role Based Access Control (RBAC) rules are used by delegating to the kube-apiserver using a `SubjectAccessReview`.

kube-proxy

The Kubernetes network proxy (kube-proxy) is a Go service that runs on each node and is used for managing connections to Pods. This service runs as a Pod in the `kube-system` namespace on each node and exposes an HTTP service. The HTTP service is used for service readiness and metrics. No functionality was identified that could be used to compromise the service or node from the local network. No authentication is required to access this service.

```
root@nodea:/# curl -ki http://nodeb:10256/healthz
HTTP/1.1 200 OK
Content-Type: application/json
X-Content-Type-Options: nosniff
Date: Fri, 21 May 2021 01:29:38 GMT
Content-Length: 149

{"lastUpdated": "2021-05-21 01:29:38.775785677 +0000 UTC m=+904.346320541","currentTime":
"2021-05-21 01:29:38.775785677 +0000 UTC m=+904.346320541"}
```

Node-to-Node kube-proxy Request Response

Unlike kubelet, kube-proxy uses a service account to communicate with the kube-apiserver. The service account token would be recoverable by an attacker with root access to a node.

Accessing the service account token is documented below to aid in future work against service account tokens from the filesystem and overall understanding of the authentication and authorization scheme. In practice there are no reasons for an attacker to use this token, as the permissions are very limited.

This scenario begins by accessing the `kubeconfig` for the kube-proxy Pod, showing the location of the service account token (local to the Pod).

```
# cat ./var/lib/kubelet/pods/cf9830d3-bb1d-46dd-9c90-
3b1c5100e6be/volumes/kubernetes.io~configmap/kube-proxy/kubeconfig.conf
apiVersion: v1
kind: Config
clusters:
- cluster:
    certificate-authority: /var/run/secrets/kubernetes.io/serviceaccount/ca.crt
    server: https://192.168.7.212:6443
  name: default
contexts:
- context:
    cluster: default
    namespace: default
    user: default
  name: default
current-context: default
users:
- name: default
  user:
    tokenFile: /var/run/secrets/kubernetes.io/serviceaccount/token
```

kube-proxy kubeconfig

Next, enumerating the location from the root filesystem on the node.

```
# cat ./var/lib/kubelet/pods/cf9830d3-bb1d-46dd-9c90-
3b1c5100e6be/volumes/kubernetes.io~projected/kube-api-access-gzccq/token
eyJhbGciOiJSUzI1NiIsImtpZCI6IjFUaE9Ra2thZ1ppS3Z1aWI1blVyWW1RWVBJVzJBMEpSUkFkbmNOMzlsNmcifQ.
eyJhdWQiOlsiaHR0cHM6Ly9rdWJlcm5ldGVzLmRlZmF1bHQuc3ZjLmNsdXN0ZXIubG9jYWwiXSwiZXhwIjoxNjUzMDk
0NTM5LCJpYXQiOjE2MjE1NTg1MzksImlzcyI6Imh0dHBzOi8va3ViZXJuZXRlcy5kZWZhdWx0LnN2Yy5jbHVzdGVyLm
xvY2FsIiwia3ViZXJuZXRlcy5pbyI6eyJuYW1lc3BhY2UiOiJrdWJlLXN5c3RlbSIsInBvZCI6eyJuYW1lIjoia3ViZ
S1wcm94eS1tdGJtbCIsInVpZCI6ImNmOTgzMGQzLWJiMWQtNDZkZC05YzkwLTNiMWM1MTAwZTZiZSJ9LCJzZXJ2aWNl
YWNjb3VudCI6eyJuYW1lIjoia3ViZS1wcm94eSIsInVpZCI6ImUxYzE4NDUxLTFiYWItNGJlMS05MDIxLTcxZTc1NjE
5YjEyMCJ9LCJ3YXJuYWZ0ZXIiOjE2MjE1NjIxNDZ9LCJuYmYiOjE2MjE1NTg1MzksInN1YiI6InN5c3RlbTpzZXJ2aW
NlYWNjb3VudDprdWJlLXN5c3RlbTprdWJlLXByb3h5In0.eI9JQ6aes7nD-g9JS-
aEYYcVvOGFakcHpZj57Mh75Sa00X84_VoG1gTArY-9pezfSdVP2-yOas1r18thFvWi4eHIdByxFFHNg7d1_8_T-
2WTmcYpBOdfG79tWiw40xIclj6CLcAc_BbE-
TXYMVYJ58rUIhiyAhcNoZUpzlRvuo_ylZ0pbLcNcDnF4ieGyIU1N7XxY5-
O9YjZgNE9OXf10YYV4DkrznWjZyVNR25bCINZtJd02LdbPHvyANoq2Lh1IPdBjeRRHJ5W-98_n-
qM4AJ3SZuugNA0EyISZIxTe0P_Md_N22JWddzQTpELBZ82McR7bQmZpOlngFDlHkffeg
```

kube-proxy Service Account Token File

The JSON Web Token (JWT) contains the following information.


```
{
  "aud": [
    "https://kubernetes.default.svc.cluster.local"
  ],
  "exp": 1653094539,
  "iat": 1621558539,
  "iss": "https://kubernetes.default.svc.cluster.local",
  "kubernetes.io": {
    "namespace": "kube-system",
    "pod": {
      "name": "kube-proxy-mtbml",
      "uid": "cf9830d3-bb1d-46dd-9c90-3b1c5100e6be"
    },
    "serviceaccount": {
      "name": "kube-proxy",
      "uid": "e1c18451-1bab-4be1-9021-71e75619b120"
    },
    "warnafter": 1621562146
  },
  "nbf": 1621558539,
  "sub": "system:serviceaccount:kube-system:kube-proxy"
}
```


kube-proxy Service Account JWT

As with kubelet's X509 certificate, this JWT can be used to authenticate to a sibling node or kube-apiserver.


```
# curl -ki https://192.168.7.214:10250/pods -H 'Authorization: bearer
eyJhbGciOiJSUzI1NiIsImtpZCI6IjFUaE9Ra2thZ1ppS3Z1aWI1blVyWW1RWVBJVzJBMEpSUkFkbmNOMzlsNmcifQ.
eyJhdWQiOlsiaHR0cHM6Ly9rdWJlcm5ldGVzLmRlZmF1bHQuc3ZjLmNsdXN0ZXIubG9jYWwiXSwiZXhwIjoxNjUzMDk
0NTM5LCJpYXQiOjE2MjE1NTg1MzksImlzcyI6Imh0dHBzOi8va3ViZXJuZXRlcy5kZWZhdWx0LnN2Yy5jbHVzdGVyLm
xvY2FsIiwia3ViZXJuZXRlcy5pbyI6eyJuYW1lc3BhY2UiOiJrdWJlLXN5c3RlbSIsInBvZCI6eyJuYW1lIjoia3ViZ
S1wcm94eS1tdGJtbCIsInVpZCI6ImNmOTgzMGQzLWJiMWQtNDZkZC05YzkwLTNiMWM1MTAwZTZiZSJ9LCJzZXJ2aWNl
YWNjb3VudCI6eyJuYW1lIjoia3ViZS1wcm94eSIsInVpZCI6ImUxYzE4NDUxLTFiYWItNGJlMS05MDIxLTcxZTc1NjE
5YjEyMCJ9LCJ3YXJuYWZ0ZXIiOjE2MjE1NjIxNDZ9LCJuYmYiOjE2MjE1NTg1MzksInN1YiI6InN5c3RlbTpzZXJ2aW
NlYWNjb3VudDprdWJlLXN5c3RlbTprdWJlLXByb3h5In0.eI9JQ6aes7nD-g9JS-
aEYYcVvOGFakcHpZj57Mh75Sa00X84_VoG1gTArY-9pezfSdVP2-yOas1r18thFvWi4eHIdByxFFHNg7d1_8_T-
2WTmcYpBOdfG79tWiw40xIclj6CLcAc_BbE-
TXYMVYJ58rUIhiyAhcNoZUpzlRvuo_ylZ0pbLcNcDnF4ieGyIU1N7XxY5-
O9YjZgNE9OXf10YYV4DkrznWjZyVNR25bCINZtJd02LdbPHvyANoq2Lh1IPdBjeRRHJ5W-98_n-
qM4AJ3SZuugNA0EyISZIxTe0P_Md_N22JWddzQTpELBZ82McR7bQmZpOlngFDlHkffeg'
HTTP/2 403
content-type: text/plain; charset=utf-8
content-length: 106

Forbidden (user=system:serviceaccount:kube-system:kube-proxy, verb=get, resource=nodes,
subresource=proxy)
```


kube-proxy-to-Node Authentication and Authorization Request Response

As expected, the service account by default does not have permissions to communicate with kubelet. Further RBAC permissions for this service account are described in the Node to API section.

Node to API

An attacker who has compromised a node will have access to at least two sets of credentials that can be used to authenticate to the kube-apiserver, X509 certificate for kubelet and a service account token for kube-proxy. Other sets of credentials could be available from Pod(s) and are dependent on configuration.

kubelet

The kubelet uses an X509 certificate to authenticate itself to kube-apiserver. In Kubernetes versions prior to v1.8, a default `ClusterRole` was used for authorization. This role was overly permissive, for example, it allowed a node to read all secrets for the entire cluster. In current Kubernetes versions the default `ClusterRole` and `ClusterRoleBinding` are still created, but no subjects are present, preventing them from being used without modification.

```
$ kubectl get clusterrolebinding system:node -o json | jq .subjects
Null
```

ClusterRoleBinding system:node Subjects

These RBAC rules were replaced with the `Node Authorizer` and `NodeRestriction` plugins. A comprehensive review of these controls was performed, culminating in an attack surface mapping document attached as a separate document to this report. This document shows the efficacy of the combined controls as well as the residual risk. Each plugin is discussed below.

Node Authorizer

The `Node Authorizer` is used by kube-apiserver to perform authorization when authenticating as a node to Kubernetes components including kubelet (both local and on a sibling node) and kube-apiserver. This is a special purpose authorization plugin that operates alongside another authorization plugin, typically RBAC. The reason for this is straight-forward. Using RBAC rules it is not possible to apply fine grained control to a sensitive API. For example, RBAC controls can allow or deny access to secrets at the namespace level, but the `Node Authorizer` can control access to only secrets that are mounted in a Pod running on a node.

This plugin is critical to reducing the impact of a compromised node. Due to this, it's worth stepping through the code to further understand the controls.

First, the user from the previously processed X509 certificate is used to ensure that this is a node, checking for the `system:node` prefix.

```
func (r *NodeAuthorizer) Authorize(ctx context.Context, attrs authorizer.Attributes)
(authorizer.Decision, string, error) {
   nodeName, isNode := r.identifier.NodeIdentity(attrs.GetUser())
   if !isNode {
       // reject requests from non-nodes
       return authorizer.DecisionNoOpinion, "", nil
   }
   if len(nodeName) == 0 {
       // reject requests from unidentifiable nodes
       klog.V(2).Infof("NODE DENY: unknown node for user %q", attrs.GetUser().GetName())
       return authorizer.DecisionNoOpinion, fmt.Sprintf("unknown node for user %q",
attrs.GetUser().GetName()), nil
   }
```

Node Authorizer Node Validation

Next, if the request references an object resource in the API, it falls through functions that are designed for a resource type. Each of these functions ensures that the requested resource has some ongoing relationship to the node in addition to restricting which verbs (get, create, list, watch, etc.) are allowed. The relationship mapping is done using an in-memory graphing package that maintains the state of the cluster.

For example, when a request comes from a node for a secret, the graph is traversed from the secret to the node to ensure that relationship exists.

```
 // subdivide access to specific resources
   if attrs.IsResourceRequest() {
       requestResource := schema.GroupResource{Group: attrs.GetAPIGroup(), Resource:
attrs.GetResource()}
       switch requestResource {
       case secretResource:
           return r.authorizeReadNamespacedObject(nodeName, secretVertexType, attrs)
       case configMapResource:
           return r.authorizeReadNamespacedObject(nodeName, configMapVertexType, attrs)
       case pvcResource:
           if r.features.Enabled(features.ExpandPersistentVolumes) {
               if attrs.GetSubresource() == "status" {
                   return r.authorizeStatusUpdate(nodeName, pvcVertexType, attrs)
               }
           }
           return r.authorizeGet(nodeName, pvcVertexType, attrs)
       case pvResource:
           return r.authorizeGet(nodeName, pvVertexType, attrs)
       case vaResource:
           return r.authorizeGet(nodeName, vaVertexType, attrs)
       case svcAcctResource:
           return r.authorizeCreateToken(nodeName, serviceAccountVertexType, attrs)
       case leaseResource:
           return r.authorizeLease(nodeName, attrs)
       case csiNodeResource:
           return r.authorizeCSINode(nodeName, attrs)
       }
```

Node Authorizer Resource Request Logic

When an incoming request is not operating on `configmaps`, `secrets`, `persistentvolumeclaims`, `persistentvolumes`, `volumeattachments`, `serviceaccounts`, `leases`, or `csinodes`; or is not for an object at all, a set of statically defined RBAC rules are used.

These rules are available at in the relative code path for the Kubernetes repository at `plugin/pkg/auth/authorizer/rbac/bootstrappolicy/policy.go`.

The following table summarized the allowed actions:

RESOURCE	ACTION
tokenreviews	create
subjectaccessreviews	create
localsubjectaccessreviews	create
services	get, list, watch
nodes	create, get, list, watch, update, patch
nodes/status	update, patch
events	create, update, patch
pods	get, list, watch, create, delete
pods/status	update, patch
pods/eviction	create
secrets	get, list, watch
configmaps	get, list, watch
persistentvolumeclaims	get
persistentvolumes	get
endpoints	get
certificatesigningrequests	create, get, list, watch
leases	get, create, update, patch, delete
volumeattachments	get
serviceaccounts/token	create

Many of these rule's conflict with the previous logic discussed above. This is known and stated many times throughout the comments. For example, regarding the rules created for secrets:

```
// Use the Node authorization mode to limit a node to get secrets/configmaps referenced by
pods bound to itself.
```

Bootstrap Policy Comment

A hypothetical authorization bypass could arise if an attacker could cause a request to fail the `attrs.IsResourceRequest()` check while still being usable by the server. During testing, no scenario was identified where this was possible.

Several rules listed would allow an attacker to break the integrity of critical components related to multi-tenancy and the cluster, notably the ability to update a node and Pod. Further inspection and restrictions are handled by the `NodeRestriction` admission plugin. Other rules would allow an attacker to retrieve potentially sensitive information about resources in the cluster. This could be an acceptable risk when multi-tenancy is used in terms of different teams within the same organization, but unacceptable within the context of many organizations.

NodeRestriction

The `NodeRestriction` plugin is an admission plugin which further limits what objects Kubelet can modify. Admission plugins are executed by kube-apiserver after authentication and authorization and prior to object persistence.

This plugin is critical to reducing the impact of a compromised node, particularly in a multi-tenant environment. Due to this, it's worth stepping through the code to further understand the controls.

Similar to the `Node Authorizer`, the plugin first verifies that the request came from a node.

```
// Admit checks the admission policy and triggers corresponding actions
func (p *Plugin) Admit(ctx context.Context, a admission.Attributes, o
admission.ObjectInterfaces) error {
    nodeName, isNode := p.nodeIdentifier.NodeIdentity(a.GetUserInfo())

    // Our job is just to restrict nodes
    if !isNode {
        return nil
    }

    if len(nodeName) == 0 {
        // disallow requests we cannot match to a particular node
        return admission.NewForbidden(a, fmt.Errorf("could not determine node from user
%q", a.GetUserInfo().GetName()))
    }
```

NodeRestriction Node Validation

Next, individual functions are used to validate modifications depending on the resource type. Logic exists for `pods`, `nodes`, `persistentvolumeclaims`, `serviceaccounts`, `leases`, and `csinodes`.

```
switch a.GetResource().GroupResource() {
case podResource:
    switch a.GetSubresource() {
    case "":
        return p.admitPod(nodeName, a)
    case "status":
        return p.admitPodStatus(nodeName, a)
    case "eviction":
        return p.admitPodEviction(nodeName, a)
    default:
        return admission.NewForbidden(a, fmt.Errorf("unexpected pod subresource %q, only
'status' and 'eviction' are allowed", a.GetSubresource()))
    }

case nodeResource:
    return p.admitNode(nodeName, a)

case pvcResource:
    switch a.GetSubresource() {
    case "status":
        return p.admitPVCStatus(nodeName, a)
    default:
        return admission.NewForbidden(a, fmt.Errorf("may only update PVC status"))
    }

case svcacctResource:
    return p.admitServiceAccount(nodeName, a)

case leaseResource:
    return p.admitLease(nodeName, a)

case csiNodeResource:
    return p.admitCSINode(nodeName, a)

default:
    return nil
}
```

NodeRestriction Sub Resource Logic

Atredis Partners performed a line-by-line review of each of these functions in combination with dynamic testing in an attempt to identify lapses in controls or other issues that could allow an attacker to affect the security of the cluster. No major issues were identified. Additionally, controls critical to multi-tenancy are adequate, for example, node cannot modify its own taints or create a Pod that is not a mirror Pod.

Residual Risk

The following table summarizes permissions allowed from kubelet to kube-apiserver when the `Node Authorizer` and `NodeRestriction` plugins are used in combination with each other. The "RBAC" control is in reference to the permissions from the bootstrap policy discussed previously.

Resource	Verb	Limitation	Control
tokenreviews	create		RBAC
subjectaccessreviews	create		RBAC
localsubjectaccessreviews	create		RBAC
services	get, list, watch		RBAC
nodes	get, list, watch		RBAC
nodes	get, list, watch, update, patch, create	• A node can only be created with its own name • Nodes are only allowed to update themselves • Nodes are not allowed to update taints	NodeRestriction RBAC
nodes/status	update, patch	• Nodes can only update the status of themselves	RBAC
events	create, update, patch		RBAC
secrets	get	• Nodes cannot list secrets • Nodes are only allowed to get secrets for Pods that are scheduled on them	NodeAuthorizer RBAC
configmaps	get	• Nodes cannot list configmaps • Nodes can only get configmaps for Pods that are scheduled on them	NodeAuthorizer RBAC
persistentvolumeclaims	get	• Nodes are only allowed to view claims that are attached to a Pod that is scheduled on them	NodeAuthorizer RBAC
persistentvolumes	get	• Nodes are only allowed to view volumes that are attached to a Pod that is scheduled on them	NodeAuthorizer RBAC
endpoints	get		RBAC
certificatesigningrequests	create, get, list, watch		RBAC

Resource	Verb	Limitation	Control
leases	get, create, update, patch, delete	• Nodes can only update and view leases for themselves	NodeAuthorizer RBAC
volumeattachments	get	• Nodes can only view a volumeattachment that is attached to a Pod scheduled on them	NodeAuthorizer RBAC
serviceaccounts/token	create	• Nodes cannot create a token for a service account unless there is a relationship to a Pod that is scheduled on the node	NodeAuthorizer NodeRestriction RBAC
csinodes	get, create, update, patch, delete	• Nodes can only interact with a csinode object if the name matches the node name	NodeAuthorizer NodeRestriction
pods	get, list, watch, create, delete	• Nodes can only delete Pods scheduled on themselves • Nodes can only create mirror Pods • Nodes can only create a Pod bound to itself	RBAC NodeRestriction

Several of these API endpoints could allow an attacker to view information about the cluster and tenants and others could be used to cause confusion to other tenants and cluster administrators. Each endpoint that presents some residual risk is discussed below.

Pods

The Pods API allows Kubernetes operators to perform create, read, update, and delete (CRUD) operations against Pods in the Kubernetes cluster. An attacker that is able to leverage authentication material from a compromised node can authenticate to the Kubernetes API server and retrieve a detailed list of Pods running in the cluster, across all namespaces, via the following API endpoints:

```
/api/v1/namespaces/{namespace}/pods
/api/v1/pods
/api/v1/watch/namespaces/{namespace}/pods
/api/v1/watch/namespaces/{namespace}/pods/{name}
/api/v1/watch/pods
```

An example of cross-namespace information disclosure can be seen below, including the Pod name, the namespace it has been deployed into, and details of the manifest used to create the Pod in the first place:

```
curl -s -ki https://100.79.47.55:6443/api/v1/watch/pods --cert kubelet-client-current.pem --key kubelet-client-current.pem
```

cURL request to `api/v1/watch/pods` endpoint

```
"metadata": {
            "name": "nginx",
            "namespace": "default",
            "uid": "9e25c3bf-b382-4ecf-ab16-35058b8a9723",
            "resourceVersion": "97598",
            "creationTimestamp": "2021-04-29T17:06:27Z",
            "annotations": {
                "kubectl.kubernetes.io/last-applied-configuration":
"{\"apiVersion\":\"v1\",\"kind\":\"Pod\",\"metadata\":{\"annotations\":{},\"name\":\"nginx\",\"namespace\":\"default\"},\"spec\":{\"containers\":[{\"image\":\"nginx\",\"imagePullPolicy\":\"IfNotPresent\",\"name\":\"nginx\"}],\"nodeSelector\":{\"foo\":\"bar\"}}}\n"
            },
[…snip…]
```

Response from `api/v1/watch/pods` endpoint

The impact of this information disclosure is that an attacker could potentially identify other tenants in the cluster, along with the types of workloads that they are running.

Services

The services API allows Kubernetes operators to perform CRUD operations against services running in the Kubernetes cluster. An attacker that can leverage authentication material from a compromised node can authenticate to the Kubernetes API server and retrieve a detailed list of services running in the cluster, across all namespaces, via the following API endpoints:

```
/api/v1/namespaces/{namespace}/services
/api/v1/services
/api/v1/watch/namespaces/{namespace}/services
/api/v1/watch/namespaces/{namespace}/services/{name}
/api/v1/watch/services
```

An example of cross-namespace information disclosure for Kubernetes services can be seen below. Some of the information disclosed includes service name, the namespace the service is associated with, and any associated labels.

```
curl -s -ki https://100.79.47.55:6443/api/v1/watch/services --cert kubelet-client-
current.pem --key kubelet-client-current.pem
```

cURL Request to `api/v1/watch/services`

```
[...snip...]
{
    "type": "ADDED",
    "object": {
        "kind": "Service",
        "apiVersion": "v1",
        "metadata": {
            "name": "nginx-service",
            "namespace": "tenant1",
            "uid": "a1422774-f55b-4ef9-957e-8544b0371b41",
            "resourceVersion": "152399",
            "creationTimestamp": "2021-04-30T18:54:55Z",
            "labels": {
                "app": "tenantapp1"
            },
[...snip...]
```

Response from `api/v1/watch/services`

The impact of an attacker being able to call API endpoints that list running services is that they could potentially identify other tenants in the cluster, along with the types of services they are exposing.

Events

The events API is used to report extra information about various objects, such as warnings and errors during Pod creation. An attacker has no restrictions in the type of events they can create. This in conjunction with other APIs could be used to tie events to cluster resources or other tenant's workloads. The impact of this is dependent on how these events are used by customers and the platform provider. It is likely that events about a Pod would want to be shown to a tenant, this could cause confusion and a lack of confidence in the service.

Nodes

The node API gives Kubernetes operators insight into the nodes running in the cluster such as the versions of kubelet they are running, the Operating System (OS) being used, and the kernel version. An attacker that is able to leverage authentication material from a compromised node can authenticate to the Kubernetes API server and retrieve detailed information for all nodes running in the cluster via the following API endpoints:

```
/api/v1/nodes
/api/v1/nodes/{name}
/api/v1/watch/nodes
/api/v1/watch/nodes/{name}
```

An example of information disclosure when it comes to Kubernetes nodes can be seen below. Some of the information disclosed includes OS version, kernel version, kubelet version, kube-proxy version, and the container runtime version.

```
curl -s -ki https://100.79.47.55:6443/api/v1/watch/nodes --cert kubelet-client-current.pem
--key kubelet-client-current.pem
```

cURL Request to `api/v1/watch/nodes` Endpoint

```
[...snip...]
            "nodeInfo": {
                "machineID": "c9653659bd2a4a498ffd1ef8e2b95d9a",
                "systemUUID": "0ee84d56-6e5f-072a-a2a5-48795e2dbefc",
                "bootID": "50f1bb6f-4656-4331-adae-041e81c97409",
                "kernelVersion": "5.4.0-72-generic",
                "osImage": "Ubuntu 20.04.2 LTS",
                "containerRuntimeVersion": "containerd://1.4.4",
                "kubeletVersion": "v1.21.0",
                "kubeProxyVersion": "v1.21.0",
                "operatingSystem": "linux",
                "architecture": "amd64"
            },
[...snip...]
```

Response from `api/v1/watch/nodes` Endpoint

The impact of an attacker being able to call the node API endpoints is that they could use these endpoints to gather information about all the worker nodes in the cluster. Kernel and OS version information could be used to further refine attacks against the cluster.

nodes/status

The `nodes/status` API has potential for an attack. This endpoint allows a node to set its address, both via IP Address and hostname. The following request shows this in action, updating the IP address of `nodea` to that of `nodeb`:

```
root@nodea# curl --key /var/lib/kubelet/pki/kubelet-client-current.pem --cert
/var/lib/kubelet/pki/kubelet-client-current.pem https://kube-
apiserver:6443/api/v1/nodes/nodea/status -X PATCH -H 'Content-Type: application/json-
patch+json' -d '[{"op": "replace", "path": "/status/addresses/0/address", "value":
"nodeb"}]' -ki
```

`nodes/status` PATCH

Modifying the IP address is this way causes kube-apiserver to send requests destined for a node to an arbitrary address. Relaying requests is also possible using a generic TCP relay on the node as well. This works because kube-apiserver does not verify the TLS connection when connecting to kubelet (because connections are always made by IP address).

For example, assuming the request above, an attacker could relay requests for `nodea` to `nodeb`. If a cluster admin starts a local proxy and sends the following request to kube-apiserver, they will get a response from `nodeb`.

```
$ kubectl proxy --port 8080 &
$ curl http://127.0.0.1:8080/api/v1/nodes/nodea/proxy/pods
{"kind":"PodList","apiVersion":"v1","metadata":{},"items":[{"me --snip--
```

Node Proxy Request

At first glance, redirecting requests to an arbitrary node may seem as if it does not present much of a risk but is definitely something to keep in mind when designing components that potentially use or present data from the kubelet API. Any tooling or debugging using a non-namespaced kubelet API would not be able to depend on the integrity of the data returned.

Lateral movement using this same scenario is not possible as most requests require a namespace, for example, the exec API `/exec/<podNamespace>/<podID>/<containerName>`. As a result, an attacker cannot coerce some external component into successfully relaying an authorized request as it would result in a HTTP `404 Not Found` error. As an aside, a modified kubelet could HTTP redirect requests to another kubelet, this scenario was identified and fixed with CVE-2018-1002102[1] (Atredis Partners validated these fixes).

The scenario described above should be taken into account if there are future updates to the kubelet API.

kube-proxy

The `system:node-proxier ClusterRole` is used for RBAC when authenticating as the `system:kube-proxy` user. The role has the following permissions.

RESOURCE	ACTION
endpoints	list, watch
services	list, watch
nodes	get, list, watch

[1] https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2018-1002102

RESOURCE	ACTION
events	create, patch, update
endpointslices	list, watch

None of these resources and associated actions present any risk that hasn't been discussed previously.

Node Bootstrap

A bootstrap token is used by kubeadm to join a new node to a cluster. A special form of authentication is used when authenticating to kube-apiserver with a bootstrap token. The bootstrap token must exist in the `kube-system` namespace and not be expired. The default expiration is 23 hours, but an expiration is not required.

```
$ sudo kubeadm token list
TOKEN                     TTL         EXPIRES                USAGES
DESCRIPTION                                                EXTRA GROUPS
pl7c73.bangf249rzznj2fd   23h         2021-05-22T05:43:57Z   authentication,signing
<none>
system:bootstrappers:kubeadm:default-node-token
```

Bootstrap Token List

Bootstrap tokens are stored as secrets and can be accessed by anyone with access to the secrets API in the `kube-system` namespace.

```
$ kubectl get secret bootstrap-token-pl7c73 -n kube-system -o yaml
apiVersion: v1
data:
  auth-extra-groups: c3lzdGVtOmJvb3RzdHJhcHBlcnM6a3ViZWFkbTpkZWZhdWx0LW5vZGUtdG9rZW4=
  expiration: MjAyMS0wNS0yMlQwNTo0Mzo1N1o=
  token-id: cGw3Yzcz
  token-secret: YmFuZ2YyNDlyenpuajJmZA==
  usage-bootstrap-authentication: dHJ1ZQ==
  usage-bootstrap-signing: dHJ1ZQ==
kind: Secret
metadata:
  creationTimestamp: "2021-05-21T05:43:57Z"
  name: bootstrap-token-pl7c73
  namespace: kube-system
  resourceVersion: "285209"
  uid: 4fabc9e3-413e-4dff-b165-778ea749969d
type: bootstrap.kubernetes.io/token
```

Bootstrap Token Secret

The RBAC rules created by kubeadm during cluster creation allow a bootstrap token to generate a `CertificateSigningRequest` (CSR). A CSR originating from the group associated with a bootstrap token will be auto-approved by the kube-controller-manager.

An attacker with access to a bootstrap token would be able to create a CSR for an existing node, and once auto-approved, use the associated certificate to impersonate as that node to kube-apiserver. This would allow an attacker to gain access to any secret in any Pod in the entire cluster. This attack is fairly common, particularly with Google Kubernetes Engine, where TLS bootstrapping information is stored in instance metadata.

An attacker could also use the bootstrap token to create a new node with taints of another tenant. This would allow them to steer tenant workloads to their node and gain unauthorized access to secrets and data.

Special care must be taken to ensure that bootstrap tokens are never left over on a node. Additionally, tight controls and monitoring should be put into place to control creation and access to these tokens.

Threat Model Overview

Atredis created the following Trust Levels based on scenarios provided by Salesforce and an understanding of the overall attack surface and dataflows.

- Node level attacker: An attacker who has leveraged one or more vulnerabilities to gain root level access to a Kubernetes node. A node level attacker will have access to at least two sets of credentials from kubelet and kube-proxy. When discussing threats, it is assumed that one of these credential sets may be used. A node level attacker may have access to TLS bootstrap credentials and is discussed as a different trust level.
- Bootstrap level attacker: An attacker who has gained access to TLS bootstrap credentials and has network access to the control plane or data plane.
- Tenant level attacker: An attacker who has access to tenant facing components that communicate with the kube-apiserver, for example, a tenant dashboard where Pods can be created.
- Control plane attacker: An attacker with network level access to one or more control plane components (e.g., kube-apiserver, etcd) and may have access to valid credentials (credential access is irrelevant to threat/impact).
- Unauthenticated control plane attacker: An attacker who has network level access to one or more control plane components but does not have valid credentials.
- Authenticated control plane attacker: An attacker who has network level access to one or more control plane components and valid credentials. An example could be an insider threat, but not necessarily. Atredis did not threat model insider threats.
- Unauthenticated data plane attacker: An attacker who has network level access to one or more data plane components but does not have valid credentials.
- Authenticated data plane attacker: An attacker who has network level access to one or more data plane components and valid credentials. An example could be an insider threat, but not necessarily. Atredis did not threat model insider threats.

The focus on this assessment was identifying threats, impacts, and mitigation against a node level attacker. Other areas have been fairly well tested and documented, and the node level attacker presents the most areas of concern in a multi-tenant environment.

Threat ID 1

Scenario

A node level attacker is able to interact with kubelet's HTTP API on another tenant's node or the control plane node.

Impact

Unauthorized access to the kubelet HTTP API would allow an attacker to execute commands on any Pod scheduled on the node. Executing commands would allow access to another tenant's programs and data.

Mitigation

Anonymous access to kubelet must be disabled and X509 authentication enabled.

The `Node Authorizer` does not allow users with the `system:node:` prefix access to the kubelet API (authorization is delegated using a `SubjectAccessReview` to kube-apiserver).

Network filtering should be used to prevent node-to-node communications across tenants, CNI plugins such as calico may provide this functionality.

Testing and Comments

Extensive testing and code review was performed against the kubelet API and `Node Authorizer`. No potential for bypass of authentication or authorization controls was identified.

Threat ID 2

Scenario

Unauthenticated data plane attacker is able to interact with the kubelet HTTP API.

Impact

Unauthorized access to the kubelet HTTP API would allow an attacker to execute commands on any Pod scheduled on the node. Executing commands would allow access to another tenant's programs and data.

Mitigation

Anonymous access to kubelet must be disabled and X509 authentication enabled. Network filtering should be used to only allow access to kubelet from authorized endpoints (kube-apiserver).

Testing and Comments

Extensive testing and code review was performed against the kubelet API and authentication controls. No potential for authentication bypass was identified and no vulnerabilities were identified.

Threat ID 3

Scenario

A node level attacker is able to capture and replay incoming credential material using the operating system or a patched kubelet.

Impact

An attacker able to capture usable credentials for kubelet would be able to authenticate to kubelet on another tenant's node and would allow them to execute commands on any Pod scheduled on that node. This would allow access to another tenant's programs and data.

Mitigation

Configure kubelet to only use X509 authentication. API bearer tokens should not be used (webhook authentication). Design systems and processes so that service account tokens are never sent to kubelet.

Testing and Comments

X509 certificates prevent capture and relaying from being useful as an attacker does not have access to the associated private key for a certificate. A service account token sent to kubelet would be usable.

Threat ID 4

Scenario

A node level attacker creates a CSR to impersonate another node or create a rogue node.

Impact

Unless approved by a third-party service or administrator, this would have no impact.

Mitigation

The kube-controller-manager `csrapprover` controller auto approves incoming CSRs for the `system:node:` prefix, but only when the requested certificate matches the name of the node that submitted the request. No further mitigation is required.

Testing and Comments

Extensive code review and testing was conducted against the `csrapprover` controller. No issues were identified.

Threat ID 5

Scenario

A bootstrap level attacker creates a CSR to impersonate another node or create a rogue node.

Impact

By impersonating any node in the cluster, an attacker would be able to gain access to any secret that is currently mounted in a running Pod. Creating a rogue node would allow an attacker to create a node with taints of any tenant (or all tenants), steering workloads (Pods) to themselves.

Mitigation

Any trace of a bootstrap token must be removed from all nodes. Bootstrap tokens should always be created with a short expiration. Only privileged administrators should be able to create or use them. Monitoring processes should be put into place to detect and respond to the creation of a CSR using a bootstrap token. This is going to be a typical event in an automated environment, but it should be possible to detect anomalies such as the creation of a CSR for an existing node. Kubernetes may be able to make design changes that prevent a bootstrap token from being used to create a CSR for an existing node.

Testing and Comments

No testing was needed for this scenario as this scenario is inherit in the design. Attacks using TLS bootstrap material are well known and documented.

Threat ID 6

Scenario

A node level attacker is able to make authorized requests to the kube-apiserver.

Impact

The impact of an authorized request is dependent on the authorization plugins in use. The `Node Authorizer` and `NodeRestriction` plugins allow an attacker to potentially view sensitive information about the cluster and other tenants. Allowed endpoints and residual risks are documented in the attack surface mapping documents.

Mitigation

The `Node Authorizer` and `NodeRestriction` plugins should be enabled at a minimum to reduce the impact of authorized requests to the kube-apiserver. There are no mitigations currently for preventing access to potentially sensitive information.

Testing and Comments

Extensive review was conducted against all allowed endpoints in an attempt to subvert controls and gain access to other tenant's information. No significant issues were identified that are not mitigation by the `Node Authorizer` and `NodeRestriction` plugins.

Threat ID 7

Scenario

A node level attacker uses the kube-apiserver to modify the taints of their node.

Impact

Modifying the taints for a node would allow an attacker to steer workloads of another tenant to itself.

Mitigation

The `NodeRestriction` plugin prevents a node from modifying its own taints. A node is also not able to create a node that already exists.

Testing and Comments

Extensive code review was conducted against the `NodeRestriction` plugin. No issues were identified that would allow a node's taints to be modified.

Threat ID 8

Scenario

A node level attacker uses the kube-apiserver to modify the labels of their node.

Impact

Modifying the labels for a node does not have any direct impact on the cluster either in or out of a multi-tenant environment. The impact is dependent on solutions built by the cluster implementor using labels.

Mitigation

Don’t depend on the integrity of node labels when designing systems and processes. Use the `NodeRestriction` plugin to prevent a node from adding labels that may have special meaning in the Kubernetes cluster.

Testing and Comments

No further testing was required. The `NodeRestriction` plugin prevents labels from being added that may have some internal control over Kubernetes components.

Threat ID 9

Scenario

An authenticated control plane attacker creates a secret containing a bootstrap token.

Impact

If the bootstrap token is created in the `kube-system` namespace they would be able to impersonate any node in the cluster or create a new node, the impact here has been previously discussed. Bootstrap tokens created in a different namespace have no impact.

Mitigation

Use RBAC controls to limit what users can create secrets in the `kube-system` namespace. Monitor for creation of secrets in the `kube-system` namespace, particularly bootstrap tokens.

Testing and Comments

Code review and dynamic testing was conducted to ensure that bootstrap tokens created outside of the `kube-system` namespace could not be used for authentication.

Threat ID 10

Scenario

A node level attacker is able to interact with the kube-proxy HTTP API.

Impact

There is no impact.

Mitigation

No mitigation is required.

Testing and Comments

Extensive code review and dynamic testing was conducted against the kube-proxy API. No endpoints exist that disclose sensitive information about the cluster or tenant and endpoints exist that can be used to change state. Further, no issues were identified that would allow an attacker to subvert the underlying node.

Threat ID 11

Scenario

A control plane attacker is able leverage a flaw to exhaust resources or crash the kube-apiserver.

Impact

A denial-of-service (DoS) attack against the kube-apiserver would affect all tenants in the cluster and may cause Pods to not execute or services to work.

Mitigation

Ensure the kube-apiserver and associated components are kept up to date. Monitor for availability and resource use and build detection for abnormal HTTP requests to the server.

Testing and Comments

DoS issues have been identified in the kube-apiserver in the past[2]. No new issues were identified during testing. Testing was conducted using various scenarios such as submitting many CSRs.

[2] https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2019-1002100

Threat ID 12

Scenario

A data plane attacker is able to leverage a flaw to exhaust resources or crash kubelet on a node.

Impact

A DoS attack against kubelet would prevent new Pods from running on the node and could affect existing workloads.

Mitigation

Ensure Kubernetes components on nodes, including kubelet, are kept up to date. Network filtering should be used to only allow access to kubelet from authorized endpoints such as kube-apiserver.

Testing and Comments

DoS issues have been identified in kubelet in the past. No new issues were identified during testing. Extensive code review of the kubelet server was conducted.

Threat ID 13

Scenario

A control plane attacker is able to compromise the kube-apiserver or the underlying Operating System using an exploit.

Impact

Impact is dependent on the nature of the exploit, but a compromise of the kube-apiserver could result in a compromise of tenant data such as secrets, as well as access to all nodes and Pods in the cluster.

Mitigation

Ensure Kubernetes components, systems, and services are kept up to date and appropriate security patches are applied regularly. Apply strict network filtering to the kube-apiserver and services on the hosting system. Make use of monitoring and detection capabilities to identify unauthorized access.

Testing and Comments

The kube-apiserver is the most reviewed and tested component in the Kubernetes ecosystem. No critical vulnerabilities have been identified in the past that could allow an attacker to execute code.

Threat ID 14

Scenario

An unauthenticated control plane attacker is able to access etcd.

Impact

Access to etcd would result in a compromise of secrets within the cluster, including bootstrap and service account tokens. An attacker could also insert their own service account and associated RBAC rules to completely compromise the entire cluster.

Mitigation

Use a combination of network filtering and mutual TLS authentication to prevent unauthorized access to etcd. Ensure etcd and the underlying operating system are kept up to date.

Testing and Comments

No testing was conducted against etcd as it was outside the scope of this assessment.

Threat ID 15

A node level attacker is able to relay, or otherwise direct authenticated requests destined for the kubelet HTTP API to the kubelet HTTP API on another node.

Impact

An attacker could cause cluster administrators or services to read log files and Pod information of another node. The impact of this is dependent on how this data is used. For example, if a service was designed to pull log files from a tenant's node (using kubelet) and present them in some way, an attacker could pull logs for an arbitrary node.

Mitigation

Design services and processes in a way that does not depend on the integrity of kubelet connections. SSH tunneling could prevent this issue as SSH connections to the wrong node should result in key errors.

Testing and Comments

Extensive testing was conducted using multiple scenarios. In no scenario was it possible to coerce a connection to URLs containing a different namespace path, including HTTP redirect attacks.

Threat ID 16

Scenario

A tenant level attacker is able to schedule a Pod on an arbitrary node.

Impact

Impact is dependent if an attacker can escape Pod restrictions and gain access to the underlying node. If access to the node was obtained, the attacker would be able to access additional Pods running on the node.

Mitigation

Uses taints to ensure Pods are only scheduled on nodes owned by a tenant. Apply tight inspections around Pod specifications to ensure that taints are explicit. Prevent a tenant from providing their own taints.

Restrict the permissions of Pods and volumes to prevent users from escaping a Pod.

Application kernels such as gVisor could be used to provide additional protections.

Testing and Comments

Dynamic testing and code review was conducted in an attempt to subvert taints within the ecosystem. Further, code review and dynamic testing of the kube-scheduler should be noted as an area for future work. The taint and tolerations plugin for the kube-scheduler was assessed thoroughly.

Threat ID 17

Scenario

A tenant user is able to mount secrets of another tenant in a Pod they control.

Impact

An attacker would be able to view secrets of another tenant.

Mitigation

Use namespaces to separate clients. Ensure that a user cannot control the namespace that resources are created in.

Testing and Comments

Namespaces are an effective control, however, given the number of scenarios where secrets are a target it would be best to research additional controls and possible alternative storage systems. No issues were identified with namespace controls during testing.

Threat ID 18

Scenario

A tenant user is able to mount a privileged service account in a Pod they control.

Impact

The impact is dependent on the permissions of the service account. An attacker would be able to access the kube-apiserver using the service account credentials.

Mitigation

Do not allow tenants to provide service account in formation in their Pod specification files. Additionally, disable automounting the default service account token by explicitly setting `automountServiceAccountToken: false` in the Pod specification.

Testing and Comments

Great care should be taken surrounding the creation and monitoring of service account tokens. This mitigations for this scenario did not require any specific testing. No issues were identified that would allow an attacker to mount service accounts of a different namespace.

Threat ID 19

Scenario

A node level attacker places a backdoor on peripherals with non-volatile memory (e.g., BMC, GPU) exposed to the host via firmware update or management interface.

Impact

Impact is dependent on the nature of the peripheral and the backdoor and could allow an attacker to maintain root level access on the node across tenant switch.

Mitigation

Ensure that firmware signatures are validated and measure non-volatile memory out-of-band between tenant switch.

Testing and Comments

This is within the threat model of cloud providers and could be considered out-of-scope based on their existing mitigations.

Findings and Recommendations

The following section outlines findings identified via manual and automated testing over the course of this engagement. Where necessary, specific artifacts to validate or replicate issues are included, as well as Atredis Partners' views on finding severity and recommended remediation.

Findings Summary

The below tables summarize the number and severity of the unique issues identified throughout the engagement.

CRITICAL	HIGH	MEDIUM	LOW	INFO
0	0	0	4	0

Findings Detail

FINDING NAME	SEVERITY
Kubeadm: Kubelet Configured with Webhook Authentication	Low
Kube-apiserver: Insecure TLS Configuration	Low
Kube-apiserver: Bootstrap Tokens Allowed without Expiration	Low
Kube-apiserver: NodeRestriction Does Not Limit Events API	Low

Kubeadm: Kubelet Configured with Webhook Authentication

Severity: Low

Finding Overview

The kubeadm deployment tool configures kubelet to use webhook authentication mode. This mode allows bearer tokens to be used for authentication. An attacker with root level access to a node would be able to capture incoming bearer tokens and use them to authenticate to kube-apiserver or a sibling node.

There is no reason to authenticate directly to kubelet using a service account. Further, service accounts in a multi-tenant environment should be used sparingly and permissions significantly limited. If this method of authentication is not enabled, there is nothing stopping a service or user from accidently sending a request containing a service account token but disallowing it at the kubelet level would prevent lateral movement.

Finding Detail

When kubeadm is used to deploy a node, kubelet contains the following authentication modes in its configuration file.

```
$ cat /var/lib/kubelet/config.yaml
apiVersion: kubelet.config.k8s.io/v1beta1
authentication:
  anonymous:
    enabled: false
  webhook:
    cacheTTL: 0s
    enabled: true
  x509:
    clientCAFile: /etc/kubernetes/pki/ca.crt
```

kubelet Authentication Config

The webhook authentication mode allows bearer tokens used in conjunction with a third-party service and service account tokens to authenticate to kubelet. If request containing a bearer token is used to authenticate to kubelet, an attacker could use a modified kubelet or system components such as a kernel module to intercept these credentials.

Since a bearer token does not have the same inherit security traits of a X509 certificate, an attacker would be able to use this token to authenticate to another kubelet instance or kube-apiserver. The impact would be dependent on the authorization controls around the account used.

Consider a service account created with the following permissions.

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: cluster-debug
  namespace: default
```

Hypothetical Service Account Definition

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: debug-admin
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: cluster-admin
subjects:
- kind: ServiceAccount
  name: cluster-debug
  namespace: default
```

Hypothetical ClusterRoleBinding

Note, a service account with the `cluster-admin` `ClusterRole` should never be created but is simple for demonstrations purposes.

The service account token is then used by either a service or a user to make an authenticated request to kubelet.

```
$ curl -ki https://192.168.7.213:10250/pods -H "Authorization: bearer $(kubectl get secret
cluster-debug-token-fvn8d -o json | jq -r '.data.token' | base64 -d)"
HTTP/2 200
content-type: application/json
date: Fri, 21 May 2021 07:12:18 GMT

{"kind":"PodList","apiVersion":"v1","metad
```

kubelet Service Account Authentication

There are various ways to intercept or otherwise access the authorization header from a compromised node. The simplest would be to patch the HTTP handler for the kubelet server with the following code.

```
diff --git a/pkg/kubelet/server/server.go b/pkg/kubelet/server/server.go
index 9c2ab3c0a3a..504f24e3f35 100644
--- a/pkg/kubelet/server/server.go
+++ b/pkg/kubelet/server/server.go
 // ServeHTTP responds to HTTP requests on the Kubelet.
 func (s *Server) ServeHTTP(w http.ResponseWriter, req *http.Request) {
+
+       fmt.Printf("Authorization header %s\n", req.Header.Get("Authorization"))
+
        handler := httplog.WithLogging(s.restfulCont, statusesNoTracePred)
```

kubelet Server Patch

When running the patched version of kubelet, the authorization header is recoverable.

```
$ sudo ./kubelet --bootstrap-kubeconfig=/etc/kubernetes/bootstrap-kubelet.conf --
kubeconfig=/etc/kubernetes/kubelet.conf --config=/var/lib/kubelet/config.yaml --container-
runtime=remote --container-runtime-endpoint=/run/containerd/containerd.sock --pod-infra-
container-image=k8s.gcr.io/pause:3.4.1 --node-ip=192.168.7.13

--snip--
I0521 07:07:55.710591   63177 reconciler.go:224]
"operationExecutor.VerifyControllerAttachedVolume started for volume \"kube-api-access-
267v4\" (UniqueName: \"kubernetes.io/projected/958bb27a-d122-4c27-be02-e4e0280a3831-kube-
api-access-267v4\") pod \"echo\" (UID: \"958bb27a-d122-4c27-be02-e4e0280a3831\") "
I0521 07:07:55.710734   63177 reconciler.go:157] "Reconciler: start to sync state"

Authorization header bearer
eyJhbGciOiJSUzI1NiIsImtpZCI6IjFUaE9Ra2thZ1ppS3Z1aWI1blVyWW1RWVBJVzJBMEpSUkFkbmNOMzlsNmcifQ.
eyJpc3MiOiJrdWJlcm5ldGVzL3NlcnZpY2VhY2NvdW50Iiwia3ViZXJuZXRlcy5pby9zZXJ2aWNlYWNjb3VudC9uYW1
lc3BhY2UiOiJkZWZhdWx0Iiwia3ViZXJuZXRlcy5pby9zZXJ2aWNlYWNjb3VudC9zZWNyZXQubmFtZSI6ImNsdXN0ZX
ItZGVidWctdG9rZW4tZnZuOGQiLCJrdWJlcm5ldGVzLmlvL3NlcnZpY2VhY2NvdW50L3NlcnZpY2UtYWNjb3VudC5uY
W1lIjoiY2x1c3Rlci1kZWJ1ZyIsImt1YmVybmV0ZXMuaW8vc2VydmljZWFjY291bnQvc2VydmljZS1hY2NvdW50LnVp
ZCI6ImE3YmQ4ZTZjLTYwMWUtNDk3OC04MjljLTZjMDAwY2Q4ZmNkZSIsInN1YiI6InN5c3RlbTpzZXJ2aWNlYWNjb3V
udDpkZWZhdWx0OmNsdXN0ZXItZGVidWcifQ.XG5LJLTxWXSrZa-
1936lAKBCvSbL9h7W_THYrlDEyHxh7Z910yo7TDqkT70dvVJO_kkeQtViM0TonJSV1-
uUBNfQeqwhf6_XqWYVtVhs75uyLR7y4nLRDhE9MA-
D9wSAe7x8RJB08bk15bL2gqNQVG7DGwb60wPO5yql3Iy9ok3X6nUerp8dE8e04thLcD6PP8kOj_ExrX1SW_mfIO2bPF
UIPFzwxUWKwXR6nFJgY69b0uuYshslh5a7WOyYOhUzpsF_ZtqKOMvmX0sVlWdv3HndixA_bwexgfaV6hwnVAGhYNHW8
gbAAomusxjQO1wA1F8ckzuKATSyLdliiTVFUg
```

Kubelet Authorization Header

As expected, this can now be used to authenticate to kube-apiserver and another kubelet instance.

Recommendation(s)

If possible, kubeadm should not enabled this authentication method by default. If this is not resolved at the kubeadm, the kubelet config of a deployed node should have the webhook authentication mode removed.

References

Kubernetes: kubelet Authentication:
https://kubernetes.io/docs/reference/command-line-tools-reference/kubelet-authentication-authorization/

Kube-apiserver: Insecure TLS Configuration

Severity: Low

Finding Overview

TLS connections originating from kube-apiserver to kubelet are not validated. This is a known issue and has been documented in previous security assessments of Kubernetes performed in 2019.

Requests are made by IP address which reduces the potential for attacks. Layer 2 attacks such as ARP spoofing is possible.

This issue does allow a malicious kubelet to redirect traffic to an arbitrary address. The impact of this attack is dependent on how use case.

Finding Detail

The use of `InsecureSkipVerify: true` is used extensively throughout the Kubernetes code base.

```
277   // Proxying to pods and services is IP-based... don't expect to be able to verify the
hostname
278   proxyTLSClientConfig := &tls.Config{InsecureSkipVerify: true}
279   proxyTransport := utilnet.SetTransportDefaults(&http.Transport{
```

InsecureSkipVerify Go TLS Configuration

One attack scenario involves a node patching itself with the address of another node.

```
root@nodea# curl --key /var/lib/kubelet/pki/kubelet-client-current.pem --cert
/var/lib/kubelet/pki/kubelet-client-current.pem https://kube-
apiserver:6443/api/v1/nodes/nodea/status -X PATCH -H 'Content-Type: application/json-
patch+json' -d '[{"op": "replace", "path": "/status/addresses/0/address", "value":
"nodeb"}]' -ki
```

Node PATCH Address

Now, any traffic destined for `nodea` in the example above would be directed to `nodeb`. If some services or user attempted to pull Pod information or log files from a node directly, they would not be able to rely on the integrity of this data.

Depending on how systems are designed this could allow a tenant to pull data from a node of another tenant, although this is very unlikely.

```
$ kubectl proxy --port 8080 &
$ curl http://127.0.0.1:8080/api/v1/nodes/nodea/proxy/pods
{"kind":"PodList","apiVersion":"v1","metadata":{},"items":[{"me --snip--
```

Example Node Proxy Request

Recommendation(s)

Design systems and processes in a way that do not depend on the integrity of these connections. SSH tunneling may resolve this issue as well.

References

CWE-295: Improper Certificate Validation:
https://cwe.mitre.org/data/definitions/295.html

Kube-apiserver: Bootstrap Tokens Allowed without Expiration

Severity: Low

Finding Overview

Bootstrap tokens can be created without an expiration. Additionally, this is the default when an expiration is not specified. This increases the likelihood of a successful attack given one is recovered by an attacker.

Finding Detail

The following file can be used to create a token without an expiration.

```
apiVersion: v1
kind: Secret
metadata:
  name: bootstrap-token-5emizz
  namespace: kube-system
type: bootstrap.kubernetes.io/token
stringData:
  auth-extra-groups: "system:bootstrappers:kubeadm:default-node-token"
  token-id: "5emizz"
  token-secret: "kq4gihvszzgn1p0r"
  usage-bootstrap-authentication: "true"
  usage-bootstrap-signing: "true"
```

Bootstrap Token YAML

After creation, there is no expiration provided.

```
$ kubectl get secret bootstrap-token-5emizz -n kube-system -o yaml
apiVersion: v1
data:
  auth-extra-groups: c3lzdGVtOmJvb3RzdHJhcHBlcnM6a3ViZWFkbTpkZWZhdWx0LW5vZGUtdG9rZW4=
  token-id: NWVtaXp6
  token-secret: a3E0Z2lodnN6emduMXAwcg==
  usage-bootstrap-authentication: dHJ1ZQ==
  usage-bootstrap-signing: dHJ1ZQ==
kind: Secret
metadata:
  creationTimestamp: "2021-05-23T21:24:02Z"
  name: bootstrap-token-5emizz
  namespace: kube-system
  resourceVersion: "344685"
  uid: ed122d0d-e4e9-4f48-8e3b-9d26ed02c663
type: bootstrap.kubernetes.io/token
```

Bootstrap Token Without Expiration

Recommendation(s)

Apply tight controls over the `kube-system` namespace and monitor for the creation of bootstrap tokens, particularly without an expiration. An admission plugin could be written to prevent bootstrap tokens without an expiration.

References

Kubernetes: Bootstrap Tokens:
https://kubernetes.io/docs/reference/access-authn-authz/bootstrap-tokens/

Kube-apiserver: NodeRestriction Does Not Limit Events API

Severity: Low

Finding Overview

The `NodeRestriction` plugin does not limit access to the events API. It is possible for an attacker to use kubelet credentials to create arbitrary events. The impact of this issue is dependent on use.

Finding Detail

In the example below, `nodea's` credentials are used to create an event for `nodeb`.

```
apiVersion: v1
count: 1
eventTime: null
firstTimestamp: "2021-05-23T21:24:55Z"
involvedObject:
  apiVersion: v1
  kind: Node
  name: nodeb
  uid: 894hhf7c-f243-409d-a88c-1bacdbe28dcf
kind: Event
lastTimestamp: "2021-05-23T21:24:55Z"
message: 'Bogus node message'
metadata:
  creationTimestamp: "2021-05-23T21:24:55Z"
  name: nodeb.1681cf2443c46a5c
  namespace: default
  resourceVersion: "344827"
  uid: 9af8dd2c-c177-47cb-8dfb-c6bd96e69928
reason: NodeNotReady
reportingComponent: ""
reportingInstance: ""
source:
  component: node-controller
type: Normal

kubeadmin@nodea:~$ sudo kubectl --kubeconfig /etc/kubernetes/kubelet.conf create -f
event.yaml
event/nodeb.1681cf2443c46a5c created
```

Event Creation

Events can be tied to arbitrary objects including Pods.

Recommendation(s)

Design systems and processes in a way that does not rely on the integrity of events. The `NodeRestriction` plugin could be modified to restrict events, based on comments in the code base this is a known issue and could be resolved in the future.

```
// TODO: restrict to the bound node as creator in the NodeRestrictions admission plugin
rbacv1helpers.NewRule("create", "update",
"patch").Groups(legacyGroup).Resources("events").RuleOrDie(),
```

bootstrap policy Code Comment

References

Kubernetes: Event API documentation:
https://v1-20.docs.kubernetes.io/docs/reference/generated/kubernetes-api/v1.20/#event-v1-core

Appendix I: Assessment Methodology

Atredis Partners draws on our extensive experience in penetration testing, reverse engineering, hardware/software exploitation, and embedded systems design to tailor each assessment to the specific targets, attacker profile, and threat scenarios relevant to our client's business drivers and agreed upon rules of engagement.

Where applicable, we also draw on and reference specific industry best practices, regulations, and principles of sound systems and software design to help our clients improve their products while simultaneously making them more stable and secure.

Our team takes guidance from industry-wide standards and practices such as the National Institute of Standards and Technology's (NIST) Special Publications, the Open Web Application Security Project (OWASP), and the Center for Internet Security (CIS).

Throughout the engagement, we communicate findings as they are identified and validated, and schedule ongoing engagement meetings and touchpoints, keeping our process open and transparent and working closely with our clients to focus testing efforts where they provide the most value.

In most engagements, our primary focus is on creating purpose-built test suites and toolchains to evaluate the target, but we do utilize off-the-shelf tools where applicable as well, both for general patch audit and best practice validation as well as to ensure a comprehensive and consistent baseline is obtained.

Research and Profiling Phase

Our research-driven approach to testing begins with a detailed examination of the target, where we model the behavior of the application, network, and software components in their default state. We map out hosts and network services, patch levels, and application versions. We frequently use a number of private and public data sources to collect Open Source Intelligence about the target, and collaborate with client personnel to further inform our testing objectives.

For network and web application assessments, we perform network and host discovery as well as map out all available application interfaces and inputs. For hardware assessments, we study the design and implementation, down to a circuit-debugging level. In reviewing source code or compiled application code, we map out application flow and call trees and develop a solid working understand of how the application behaves, thus helping focus our validation and testing efforts on areas where vulnerabilities might have the highest impact to the application's security or integrity.

Analysis and Instrumentation Phase

Once we have developed a thorough understanding of the target, we use a number of specialized and custom-developed tools to perform vulnerability discovery as well as binary, protocol, and runtime analysis, frequently creating engagement-specific software tools which we share with our clients at the close of any engagement.

We identify and implement means to monitor and instrument the behavior of the target, utilizing debugging, decompilation and runtime analysis, as well as making use of memory and filesystem

forensics analysis to create a comprehensive attack modeling testbed. Where they exist, we also use common off-the-shelf, open-source and any extant vendor-proprietary tools to aid in testing and evaluation.

Validation and Attack Phase

Using our understanding of the target, our team creates a series of highly-specific attack and fault injection test cases and scenarios. Our selection of test cases and testing viewpoints are based on our understanding of which approaches are most relevant to the target and will gain results in the most efficient manner, and built in collaboration with our client during the engagement.

Once our test cases are validated and specific attacks are confirmed, we create proof-of-concept artifacts and pursue confirmed attacks to identify extent of potential damage, risk to the environment, and reliability of each attack scenario. We also gather all the necessary data to confirm vulnerabilities identified and work to identify and document specific root causes and all relevant instances in software, hardware, or firmware where a given issue exists.

Education and Evidentiary Phase

At the conclusion of active testing, our team gathers all raw data, relevant custom toolchains, and applicable testing artifacts, parses and normalizes these results, and presents an initial findings brief to our clients, so that remediation can begin while a more formal document is created. Additionally, our team shares confirmed high-risk findings throughout the engagement so that our clients may begin to address any critical issues as soon as they are identified.

After the outbrief and initial findings review, we develop a detailed research deliverable report that provides not only our findings and recommendations but also an open and transparent narrative about our testing process, observations and specific challenges in developing attacks against our targets, from the real world perspective of a skilled, motivated attacker.

Automation and Off-The-Shelf Tools

Where applicable or useful, our team does utilize licensed and open-source software to aid us throughout the evaluation process. These tools and their output are considered secondary to manual human analysis, but nonetheless provide a valuable secondary source of data, after careful validation and reduction of false positives.

For runtime analysis and debugging, we rely extensively on Hopper, IDA Pro and Hex-Rays, as well as platform-specific runtime debuggers, and develop fuzzing, memory analysis, and other testing tools primarily in Ruby and Python.

In source auditing, we typically work in Visual Studio, Xcode and Eclipse IDE, as well as other markup tools. For automated source code analysis we will typically use the most appropriate toolchain for the target, unless client preference dictates another tool.

Network discovery and exploitation make use of Nessus, Metasploit, and other open-source scanning tools, again deferring to client preference where applicable. Web application runtime analysis relies extensively on the Burp Suite, Fuzzer and Scanner, as well as purpose-built automation tools built in Go, Ruby and Python.

Engagement Deliverables

Atredis Partners deliverables include a detailed overview of testing steps and testing dates, as well as our understanding of the specific risk profile developed from performing the objectives of the given engagement.

In the engagement summary we focus on "big picture" recommendations and a high-level overview of shared attributes of vulnerabilities identified and organizational-level recommendations that might address these findings.

In the findings section of the document, we provide detailed information about vulnerabilities identified, provide relevant steps and proof-of-concept code to replicate these findings, and our recommended approach to remediate the issues, developing these recommendations collaboratively with our clients before finalization of the document.

Our team typically makes use of both DREAD and NIST CVE for risk scoring and naming, but as part of our charter as a client-driven and collaborative consultancy, we can vary our scoring model to a given client's preferred risk model, and in many cases will create our findings using the client's internal findings templates, if requested.

Sample deliverables can be provided upon request, but due to the highly specific and confidential nature of Atredis Partners' work, these deliverables will be heavily sanitized, and give only a very general sense of the document structure.

Appendix II: Engagement Team Biographies

Shawn Moyer, Founding Partner and CEO

Shawn Moyer scopes, plans, and coordinates security research and consulting projects for the Atredis Partners team, including reverse engineering, binary analysis, advanced penetration testing, and private vulnerability research. As CEO, Shawn works with the Atredis leadership team to build and grow the Atredis culture, making Atredis Partners a home for some of the best minds in information security, and ensuring Atredis continues to deliver research and consulting services that exceed our client's expectations.

Experience

Shawn brings over 25 years of experience in information security, with an extensive background in penetration testing, advanced security research including extensive work in mobile and Smart Grid security, as well as advanced threat modeling and embedded reverse engineering.

Shawn has served as a team lead and consultant in enterprise security for numerous large initiatives in the financial sector and the federal government, including IBM Internet Security Systems' X-Force, MasterCard, a large Federal agency, and Wells Fargo Securities, all focusing on emerging network and application attacks and defenses.

In 2010, Shawn created Accuvant Labs' Applied Research practice, delivering advanced research-driven consulting to numerous clients on mobile platforms, critical infrastructure, medical devices and countless other targets, growing the practice 1800% in its first year.

Prior to Accuvant, Shawn helped develop FishNet Security's penetration testing team as a principal security consultant, growing red team offerings and advanced penetration testing services, while being twice selected as a consulting MVP.

Key Accomplishments

Shawn has written on emerging threats and other topics for Information Security Magazine and ZDNet, and his research has been featured in the Washington Post, BusinessWeek, NPR and the New York Times. Shawn is a twelve-time speaker at the Black Hat Briefings and has been an invited speaker at other notable security conferences around the world.

Shawn is likely best known for delivering the first public research on social network security, pointing out much of the threat landscape still exists on social network platforms today. Shawn also co-authored an analysis of the state of the art in web browser exploit mitigation, creating the first in-depth comparison of browser security models along with Dr. Charlie Miller, Chris Valasek, Ryan Smith, Joshua Drake, and Paul Mehta.

Shawn studied Computer and Network Information Systems at Missouri University and the University of Louisiana at Lafayette, holds numerous information security certifications, and has been a frequent presenter at national and international security industry conferences.

Tom Steele, Research Consulting Director

Tom Steele leads and executes application security assessments and adversarial engagements, ranging from source code review to advanced red team assessments.

Experience

Tom has over eight years of professional experience in information security. During that time, his focus has been on executing and innovating both network and application-level assessments; with a focus on developing new techniques, tools, and processes that improve collaborative testing, coverage, deterrent bypass, and data exfiltration.

In addition to performing assessments, Tom is also a seasoned software developer, and has an expert knowledge of multiple languages and platforms including Go and Node.js. Tom understands how applications fit together and has used his development experience to develop and maintain many widely used open-source and proprietary tools including Lair, a real-time testing collaboration application, and BurpBuddy, an API for BurpSuite Pro.

Prior to joining Atredis, Tom was a practice manager on Optiv's Attack and Penetration team, where he led a team of consultants, developed and enhanced methodologies, toolsets, and processes, and conducted hundreds of security assessments.

Key Accomplishments

Tom is a contributor to the Node Security Project, where he has assisted with the identification and remediation of many vulnerabilities; both in Node core and in widely deployed libraries. He has consulted leaders working at Fortune 500 companies on how to increase the security of their application frameworks. He has presented and lead training at several conferences including Black Hat, DEF CON, BSides, and DerbyCon and is the Co-Author of No Starch Press' "Black Hat Go".

Joshua Dow, Senior Research Consultant

Joshua executes highly technical network, web application, and containerization security assessments, as well as red team and attack simulation engagements.

Experience

Joshua has over 7 years of experience in information security, as both a consultant, in-house security engineer, and software developer. His experience includes application development, application security, container security, network penetration testing, and red team assessments.

Prior to joining Atredis Partners, Joshua performed red team, network, web application, and containerization security assessments as a Senior Security Consultant at NCC Group on both the Full-Spectrum Attack Simulation (FSAS), and Container and Orchestration Security Services (COSS) teams.

Key Accomplishments

Joshua has created open source tooling, presented at a variety of industry conferences, and authored blog posts to give back to the information security community.

Joshua studied computer science at the University of Massachusetts Boston.

Nathan Keltner, Founding Partner and CTO

Nathan Keltner leads, executes and coordinates advanced, custom-scoped projects for Atredis Partners. Nathan's primary focus includes hardware reverse engineering and penetration testing, red teaming, protocol analysis and private vulnerability research.

Experience

Nathan began his security career performing penetration tests and various security assessments for a large retail corporation, later expanding his career in consulting and specialization within red team penetration testing, exploit development, and software and hardware reverse engineering. Prior to starting Atredis Partners, Nathan most recently was a Senior Research Consultant on Accuvant's Applied Research team.

Nathan has also worked extensively as a penetration tester, helping design penetration testing methodologies and workflows as well as leading complex red team, social engineering, and attack simulation engagements, as well as numerous reverse engineering and binary analysis projects.

Nathan's research and exploitation assessments have recently focused on server hardware and embedded appliances, such as identification of vulnerabilities in BMC, UEFI, or OS firmware in related components. Previous expertise includes study of custom RF and ZigBee smart grid infrastructures, 802.15.4 and serial retail networks, multi-function ATM hardware and software, PIN entry devices, IPTV, VoIP hardware and software stacks, and modern networking access controls and identity management systems.

Key Accomplishments

Nathan has spoken at Black Hat USA, REcon, DEF CON, and other similar conferences on topics such as researching and exploiting smart grid radio frequency systems, exploitation in ARM TrustZone, advanced analysis of purpose-built system-on-chip architectures, and exploitation under limited-access user security models on the Windows platform.

Nathan holds a Bachelor of Business Administration degree in Management Information Systems from the University of Oklahoma, has held many information security and audit certifications over the years, and has been a frequent presenter at national and international security industry conferences.

Lacey Kasten, Client Operations, Technical Writer and Editor

Lacey Kasten helps facilitate client operations and deliverable creation/development at Atredis Partners. From supporting pre-sales project scoping and back-end operations efforts to shepherding the technical writing style and voice at Atredis, to the final quality assurance review of penetration test deliverables prior to engagement completion, Lacey seeks to provide readable, understandable communication to Atredis Partners' clientele. Lacey stays embedded in the Information Security community and is passionate about accessible and equitable knowledge transfer in all mediums across a wide span of Cyber Security and Information Technology topics.

Experience

Lacey has worked in communications roles from within the Fine Art and Design industry, Museum and Nonprofit Philanthropy space, Biomedical Computer Science, Higher Education Public Relations, and Event and Tradeshow industry throughout her career. Her work spans writing (technical, copy editing), editing and mentorship of writers in the Information Security space, content creation (web development, event planning, graphic design, and photography), and film and movie production.

Key Accomplishments

Lacey achieved a bachelor's degree in Communication Design from the Pacific Northwest College of Art in Portland, Oregon. She is a member on the Board of Directors for the largest Information Security conference in the United States Pacific Northwest, Security BSides PDX, and serves the charitable 501(c)(3) as coordinator of Sponsorship and Endowment.

Appendix III: About Atredis Partners

Atredis Partners was created in 2013 by a team of security industry veterans who wanted to prioritize offering quality and client needs over the pressure to grow rapidly at the expense of delivery and execution. We wanted to build something better, for the long haul.

In six years, Atredis Partners has doubled in size annually, and has been named three times to the Saint Louis Business Journal's "Fifty Fastest Growing Companies" and "Ten Fastest Growing Tech Companies". Consecutively for the past three years, Atredis Partners has been listed on the Inc. 5,000 list of fastest growing private companies in the United States.

The Atredis team is made up of some of the greatest minds in Information Security research and penetration testing, and we've built our business on a reputation for delivering deeper, more advanced assessments than any other firm in our industry.

Atredis Partners team members have presented research over forty times at the BlackHat Briefings conference in Europe, Japan, and the United States, as well as many other notable security conferences, including RSA, ShmooCon, DerbyCon, BSides, and PacSec/CanSec. Most of our team hold one or more advanced degrees in Computer Science or engineering, as well as many other industry certifications and designations. Atredis team members have authored several books, including *The Android Hacker's Handbook*, *The iOS Hacker's Handbook*, *Wicked Cool Shell Scripts*, *Gray Hat C#*, and *Black Hat Go*.

While our client base is by definition confidential and we often operate under strict nondisclosure agreements, Atredis Partners has delivered notable public security research on improving the security at Google, Microsoft, The Linux Foundation, Motorola, Samsung and HTC products, and were the first security research firm to be named in Qualcomm's Product Security Hall of Fame. We've received four research grants from the Defense Advanced Research Project Agency (DARPA), participated in research for the CNCF (Cloud Native Computing Foundation) to advance the security of Kubernetes, worked with OSTIF (The Open Source Technology Improvement Fund) and The Linux Foundation on the Core Infrastructure Initiative to improve the security and safety of the Linux Kernel, and have identified entirely new classes of vulnerabilities in hardware, software, and the infrastructure of the World Wide Web.

In 2015, we expanded our services portfolio to include a wide range of advanced risk and security program management consulting, expanding our services reach to extend from the technical trenches into the boardroom. The Atredis Risk and Advisory team has extensive experience building mature security programs, performing risk and readiness assessments, and serving as trusted partners to our clients to ensure the right people are making informed decisions about risk and risk management.