Tracing Vulnerability CVE-2018-1002105 on Kubernetes

By: Zhaoyan Xu, Yanhui Jia, Bo Qu, Xin Ouyang

Date: December 14, 2018

Introduction

On Dec 3, 2018, a new critical vulnerability in Kubernetes was disclosed to the public. The vulnerability itself originates from a programming error resulting in incorrect handling of error responses to proxy upgrade requests. This is a privilege escalation vulnerability which allows crafted requests that are authenticated with the Kubernetes API server's TLS credentials to establish a backend connection.

Different from misconfiguration vulnerabilities of K8s, this vulnerability cast the spotlight on K8s network interaction between user/pod and api-server. It is also a perfect example to demonstrate best practices for securing traffic inside the Kubernetes cluster.

Vulnerability Timeline

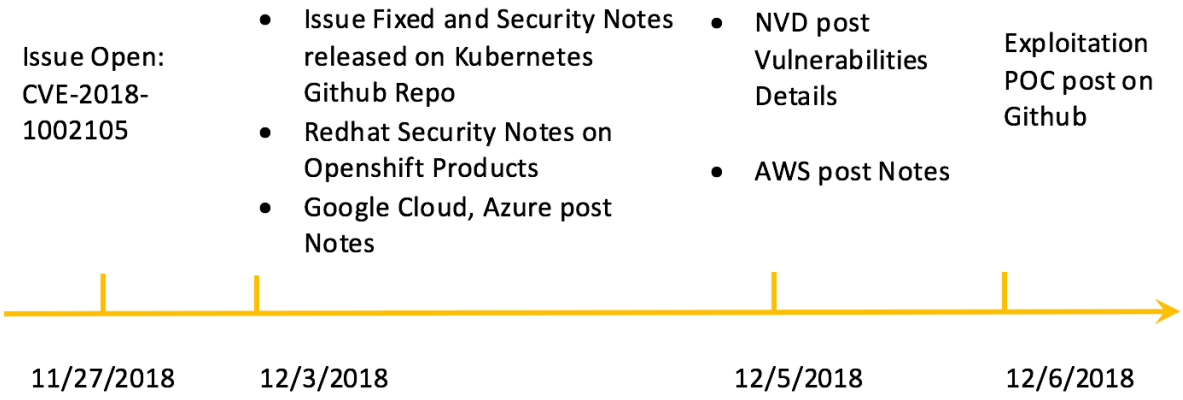

Figure 1: Timeline of Vulnerability CVE-2018-1002105

According to public resources [1], the issue was first discovered on Nov 27th, 2018 as Kubernetes Issue #71411. The initial issue states, *"When proxying to a backend server, if the server returns an error response, the connection gets stuck."* Initially, the issue was labeled as a program bug and on Dec 3rd, it was labelled as a critical and urgent security bug. On the same day, Google released its security notes about the vulnerability.

In the following two days, the main cloud service providers, Google Cloud [2], Azure [3] and AWS [4], posted their security and mitigation notes. They also provided patches for their managed K8s clusters. The mitigation method is to upgrade the K8s to unaffected versions.

The NVD [5] posted full vulnerability details on Dec 5th, 2018 and the security researchers posted the exploitation POC on Dec 6th, 2018. Now, the code patch is being thoroughly studied by the security community [6-7].

Mechanism and Mitigation

The vulnerability exists in the logic of handling proxy requests. The vulnerable code is in *UpgradeAwareHandler* which handles HTTP Upgrade requests.

The HTTP upgrade request is a HTTP request with an "Upgrade" Header field. This request is used when a user begins a normal HTTP request but needs to switch to a different protocol, such as TLS or Websocket. The WebSocket Protocol has two parts: a handshake to establish the upgraded connection, then the actual data transfer. First, a client requests a WebSocket connection by using the *Upgrade: WebSocket* and *Connection: Upgrade* headers, along with a few protocol-specific headers to establish the version being used and set up a handshake. The server, if it supports the protocol, replies with the same *Upgrade: WebSocket* and *Connection: Upgrade* headers and completes the handshake. Once the handshake is completed successfully, data transfer begins.

In K8s context, the *kubelet* on each compute node uses Websocket connection to communicate with the api-server. When the api-server connects to the kubelet, it uses its certificate with '`cluster-admin`' privilege, one of the highest privileges in K8s.

The vulnerability occurs when an authorized HTTP request comes with an Upgrade header. The request goes through the proxy and tries to connect to the kubelet. However, even when the kubelet responds with a non-101 response code, the proxy server successfully builds the connection between the client and the kubelet. Using this '`cluster-admin`' privilege connection, the attacker can subsequently send additional requests to the kubelet even though these requests were never authorized. Similar issue is also seen in an aggregated server. The aggregated server is a set of servers working as extensions to the api-server. The proxy working in the middle is also vulnerable. As long as the received upgrade request is authorized, the proxy builds a connection even when the internal aggregated server issues a non-101 response code.

Another dangerous impact is observed when the cluster allows anonymous access to the api-server. Since anonymous access passes the check on the api-server, the attacker could gain a direct connection to the aggregated server(s), such as *servercatalog* and *metrics-server*. It is even possible for an anonymous user to execute code if the aggregated server provides such functionality.

The fix for the vulnerability [8] closes the network connection if the protocol switch fails at the kubelet or the aggregated server. Apart from a code patch, the mitigation steps include:

(1) Disallowing anonymous access on api-server.
(2) Restrict access to aggregated APIs from users that should not have full access to the aggregated APIs.
(3) Remove pod exec/attach/portforward permissions from users that should not have full access to the kubelet API.

More details can be found in references [1-7].

Exploitation

We also constructed a proof-of-concept lab to exploit this vulnerability. We set up a version 1.12.0 Kubernetes server for this experiment and the api-server also runs one aggregated server (*servercatalog)* as its extension.

The configuration is illustrated in Figure 2.

```
kubernetes@kubernetes:~$ kubelet --version
Kubernetes v1.12.0
kubernetes@kubernetes:~$ kubectl get pod --all-namespaces
NAMESPACE     NAME                                          READY   STATUS              RESTARTS   AGE
catalog       catalog-catalog-apiserver-74ccfdccb5-72fgt    2/2     Running             10         13h
catalog       catalog-catalog-controller-manager-578b5f4c54-fk9lf   1/1     Running             12         13h
default       redis                                         0/1     ContainerCreating   0          20h
kube-system   busybox-test5                                 1/1     Running             1          20h
kube-system   coredns-869f847d58-9nbcg                      1/1     Running             1          5d20h
kube-system   coredns-869f847d58-h9ndd                      1/1     Running             1          5d20h
kube-system   etcd-kubernetes                               1/1     Running             1          15h
kube-system   kube-apiserver-kubernetes                     1/1     Running             1          13h
kube-system   kube-controller-manager-kubernetes            1/1     Running             11         13h
kube-system   kube-proxy-7dsrh                              1/1     Running             2          5d20h
kube-system   kube-scheduler-kubernetes                     1/1     Running             10         13h
kube-system   tiller-deploy-6f4dbc6d67-sd8xg                1/1     Running             1          14h
kube-system   weave-net-xrgx7                               2/2     Running             5          5d20h
```

Figure 2: Vulnerability Reproduction Environment

There are two steps for a successful exploitation. First, we need to build a tunnel using the protocol upgrade request. As shown in Figure 3, the first request has an Upgrade HTTP header for a protocol switch to WebSocket. The second step is the actual exploitation. As discussed, there are two possible exploitation scenarios: *authenticated* and *unauthenticated*. Authenticated case requires that the attacker has permissions for pod exec/attach/portforward and such a condition is hard to

achieve in a real-world attack. In Figure 3, we illustrate the unauthenticated case we were able to reproduce. As shown, we successfully escalate the anonymous user's privilege and obtained information from the aggregated server.

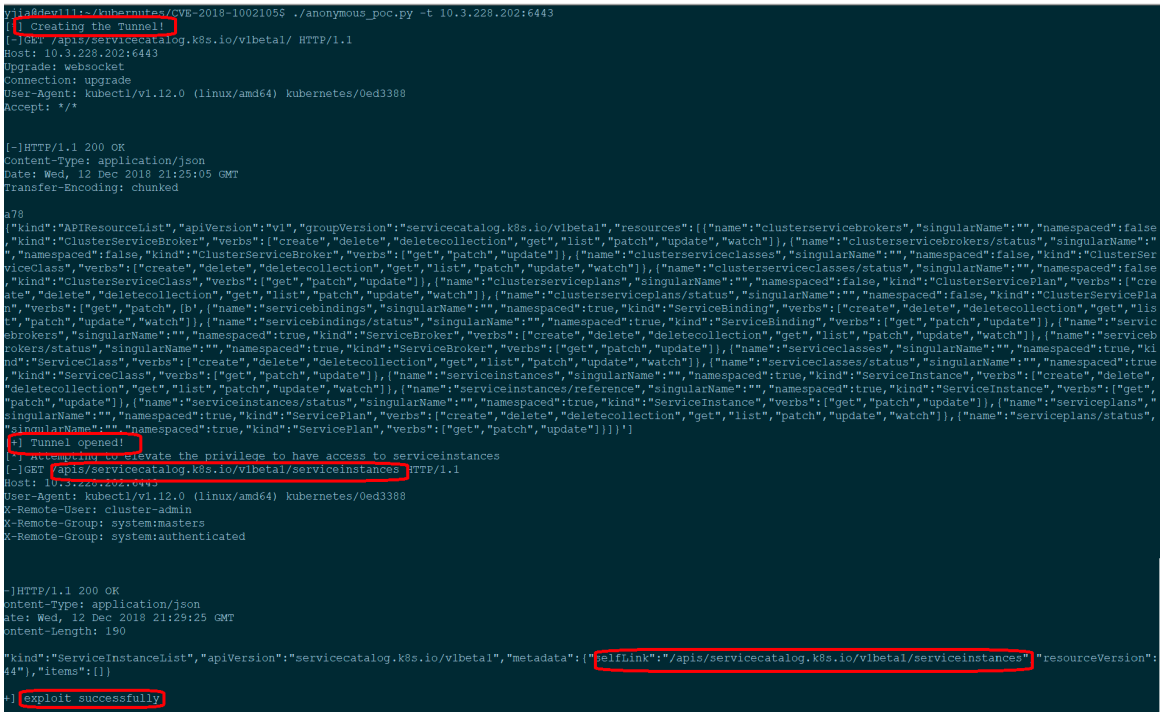

Figure 3: Exploitation Reproduction Illustration

Securing Communication in K8s

Palo Alto networks has been researching Kubernetes related vulnerabilities since mid 2017. In this time, there have been 292 reported security issues in K8s repository. Among them were 15 critical issues that led to 3 critical security vulnerabilities, CVE-2017-1002101, CVE-2017-1002101 and CVE-2018-1002105., CVE-2018-1002105 is different from the previous two vulnerabilities since it directly allows privilege escalation. More importantly, there exists multiple attack vectors for exploitation and it is challenging to provide coverage for all of them.

The most efficient mitigation is to upgrade the cluster or patch the server. Other mitigation methods require discrete compliance management which may not be available to all users.

Based on prior experience with defense against emerging network vulnerabilities, we have found that the first few weeks after vulnerability disclosure is when the exploitation traffic is at its peak. Taking proper mitigation steps in a timely fashion is

critical to securing K8s cluster. Since mitigation might involve engaging the right technical experts and/or scheduling a downtime, we believe network-based intrusion prevention is of utmost importance to reduce the window of exposure.

Palo Alto Networks intrusion prevention system has released multiple signatures (Threat Prevention Signature 40580 and 54815) for the *possible* detection of CVE-2018-1002105. These signatures match against a failed HTTP upgrade request that subsequently sends requests to K8s aggregated server or requests exec/attach/portforward permissions on any pod resource. Taking advantage of these signatures could help detect the possible intrusion before all the required mitigation steps have been adopted.

A highlighted item in the security and mitigation notes released by Google on Dec *5th, 2018,* states, *"The requests do appear in the kubelet or aggregated API server logs but are indistinguishable from correctly authorized and proxied requests via the Kubernetes API server".* This is indeed correct because the protocol switch request is a valid request and is a common occurrence in K8's cluster traffic. This is why Palo Alto Networks categorized the signatures released for CVE-2018-1002105 as *informational*. A block action on these signatures may cause false positives in practice.

Conclusion

In this blog, we discuss a recent critical Kubernetes vulnerability, CVE-2018-1002105 that could potentially cause unwanted privilege escalation and compromise the whole Kubernetes cluster. We explain the mechanism behind this vulnerability and illustrate how we reproduced the exploitation in our Lab environment. Lastly, we summarize and disclose the following notes for the vulnerability:

CVE-2018-1002105 is in the wild and will likely be exploited for years to come. To remediate this issue, administrators should deploy Kubernetes's patch for this vulnerability, available here [1].

Those who cannot deploy the patch should consider disabling anonymous access, suspending use of aggregated API servers and removing pod exec/attach/portforward permissions from users that should not have full access to the kubelet API [1].

Palo Alto Networks customers should use the below steps to detect this vulnerability:

- Download latest content update on the Next Generation Firewall and use Threat Prevention Signatures 40580 and 54815 to identify the traffic containing the malicious behavior.
- Customers should configure their security profiles to alert on these signatures since it has a high False Positive risk.
- Customers should also enable SSL Decryption in their Next Generation Firewall to detect this attack.

As always, Palo Alto Networks recommends that you follow Best Practices for upgrading appliances to the latest version of applications and threats content updates according to your network environment and that you review your policies to ensure you have configured the appropriate actions for all Security policy rules.

Reference

[1] CVE-2018-1002105. https://github.com/kubernetes/kubernetes/issues/71411
[2] Kubernetes Security Bulletins: https://cloud.google.com/kubernetes-engine/docs/security-bulletins
[3] AKS clusters patched for Kubernetes Vulnerability. https://azure.microsoft.com/en-us/updates/aks-clusters-patched-for-kubernetes-vulnerability/
[4] AWS security note. https://aws.amazon.com/security/security-bulletins/AWS-2018-020/
[5] NVD vulnerability. https://nvd.nist.gov/vuln/detail/CVE-2018-1002105
[6] POC for CVE-2018-1002105. https://github.com/evict/poc_CVE-2018-1002105
[7] Gravitational/Blog. https://gravitational.com/blog/kubernetes-websocket-upgrade-security-vulnerability/
[8] Code Patch for CVE-2018-1002105. https://github.com/kubernetes/kubernetes/commit/b84e3dd6f80af4016acfd891ef6cc50ce05d4b5b