Rook

Security Assessment

December 19, 2019

Prepared For:
Jared Watts | *Rook*
jbw976@gmail.com

Prepared By:
Robert Tonic | *Trail of Bits*
robert.tonic@trailofbits.com

Stefan Edwards | *Trail of Bits*
stefan.edwards@trailofbits.com

Kristin Mayo | *Trail of Bits*
kristin.mayo@trailofbits.com

Executive Summary

From December 2 through December 19, 2019, Rook engaged with Trail of Bits to review the security of the storage orchestration system for Kubernetes, also named Rook. Trail of Bits conducted this assessment over the course of two person-weeks with two engineers working from the release-1.1 branch of the rook/rook repository.

The week-long assessment consisted of manual review, static analysis, and operational analysis with a focus on common Go mistakes, security-critical configuration, and protocol use. This resulted in 13 findings ranging from High to Low in severity. Notably, finding TOB-ROOK-005: Logging of sensitive information captures issues in overly-verbose logs disclosing credentials and keypair data.

Throughout the codebase there are common mistakes that were identified by several static analysis tools. Ineffectual assignments detected by `ineffasign` resulted in several findings, such as TOB-ROOK-002: Ineffectual assignments result in unused variable values, which generally capture situations in which values are ignored after assignment.

Improper error handling was identified by `errcheck`, `staticcheck`, and `gosec`, resulting in findings TOB-ROOK-004: Improperly handled type assertion error could result in invalid execution state and TOB-ROOK-003: Incorrect error handling could result in finalizer not being removed, which generally capture situations in which errors are not caught or checked, or do not result in modifications to execution flow.

Similarly, searching through the codebase for type assertions resulted in many instances of type assertions that could result in runtime panics if an `interface{}` is unable to be cast into the asserted type. These problems are captured within findings TOB-ROOK-003: Potential runtime panics from unhandled type assertions. Furthermore, we found large swatches of the code accept `interface{}` as a type; if possible, this type should be narrowed to avoid requiring type analysis and dispatch wherever possible.

Ultimately, Rook should focus on remediating the common problems detected by available open-source static analysis tools. These tools should be integrated into the project's CI/CD environment to help mitigate introduction of these problems into the codebase. Appendix C: Static Analysis Recommendations details the recommended approach for this integration. Beyond static analysis, Rook should review the default logging and permissions applied by each component of the project, including the orchestrated subsystems. Lastly, Rook should also review all "edges" of the system by which user input may traverse the system, including command line arguments, configuration files, and environment variables. Dynamic analysis should also be considered for applicable components. Several options have been detailed in Appendix D: Golang fuzzing and property testing best practices. Once completed, subsequent assessments should be performed to ensure appropriate remediations have been made and that no new issues have been introduced by the changes.

Project Dashboard

Application Summary

Name	Rook
Version	992adff55d0a9961358deb268c35c8a5641de759
Type	Go
Platforms	Go, Kubernetes, Ceph, EdgeFS, various others

Engagement Summary

Dates	December 2–6
Method	Whitebox
Consultants Engaged	2
Level of Effort	2 person-weeks

Vulnerability Summary

Total High-Severity Issues	1	■
Total Medium-Severity Issues	7	■■■■■■■
Total Low-Severity Issues	5	■■■■■
Total	13	

Category Breakdown

Error Handling	4	■■■■
Data Validation	2	■■
Logging	1	■
Undefined Behavior	1	■
Access Controls	1	■
Authentication	1	■
Cryptography	2	■■
Configuration	1	■
Total	13	

Engagement Goals

The engagement was scoped to provide a security assessment of Rook, focusing on the Ceph and EdgeFS file system deployment orchestrators.

Specifically, we sought to answer the following questions:

- Is there any way to manipulate the command line arguments built from untrusted user input?
- Are hashing functions and encryption systems used by Rook and the file systems it orchestrates secure for their intended use?
- Are there any error handling issues that could result in an invalid operational state of the Rook orchestrator?
- Are there any situations in which logging discloses sensitive information?
- Are the files and directories being created with the appropriate permissions?

Coverage

Logging. Sections of the codebase producing log entries were reviewed for sensitive material disclosure such as credentials and keypair data.

Cryptography. The packages used for cryptographic operations, as well as their usage, were reviewed. Specifically, random number generator use, hashing functions, and credential generation were reviewed.

Error handling. Using a mixture of manual review and static analysis, error handling was reviewed to identify situations in which errors were not returned, handled, or properly propagated.

Data validation. Data validation was reviewed, focusing on CRD and operator validation routines. Areas where validations on one layer could be used to cause unintended behavior on another layer were prioritized.

File and directory permissions. The codebase was reviewed for file and directory operations performed with loose permissions or potentially unintended behavior.

Recommendations Summary

This section aggregates all the recommendations made during the engagement. Short-term recommendations address the immediate causes of issues. Long-term recommendations pertain to the development process and long-term design goals.

Short Term

- [ ] **Fix the variable shadowing for the `serverIfName` variable.** Improve testing to cover all branches, detecting misconfiguration in the final result. TOB-ROOK-001

- [ ] **Audit the locations noted above, and determine if the ineffectual assignment is problematic.** Areas where default values aren't used may be replaced with a `var` declaration, and areas where errors or return values are ignored may be replaced with `_`. TOB-ROOK-002

- [ ] **Audit all locations noted above, and switch to the slightly longer form and include a check of the `ok` variable.** This will ensure that conversions are at least minimally checked prior to continuing execution flow, and prevent panics due to incorrect type conversions. TOB-ROOK-003

- [ ] **Change the execution flow to exit early or handle the situation where `ok` is false, rather than simply continuing onwards with an incorrect value.** TOB-ROOK-004

- [ ] **Audit all log locations, and note which information should and should not be included within the logs.** TOB-ROOK-005

- [ ] **Audit all locations within the code base to ensure that the correct file system permissions are used.** These should be the most narrow permissions possible; for example, wherever possible, change `0644` (group and world readable, user writable and readable) into `0600` (only the owner may read). This will ensure that only the user executing Rook will be able to access sensitive information. TOB-ROOK-006

- [ ] **Use a cryptographically secure random number generator such as crypto/rand.** TOB-ROOK-007

- [ ] **Use newer hash methods such as SHA2 that are less prone to collision.** This will help prevent attackers from abusing hash-based areas of the application. TOB-ROOK-008

- [ ] **Improve error handling within the operator.** Ensure validation of the CRD entry leads to the appropriate execution flow change to prevent failure loops. TOB-ROOK-009

❑ **Apply validation on the CRD layer and the operator level for the `CephObjectStore` CRD entries.** TOB-ROOK-010

❑ **Use proper encoders to create structured content.** Ensure values are properly escaped and validated.

❑ **Increase the default length of the passwords and allow customizable lengths.** TOB-ROOK-012

❑ **Fix the logic to correctly report type assertion errors and refrain from reporting the finalizer as removed.** TOB-ROOK-013

Long Term

❑ **Integrate static analysis into the CI/CD pipeline to detect variable shadowing in submitted code.** The shadow tool for `go vet` is recommended. TOB-ROOK-001

❑ **Use the `ineffassign` tool as part of the larger set of CI/CD static analysis tools, and audit all locations noted during builds.** Furthermore, there may be a case to be made that developers use such tools as part of a pre-commit hook in git, so these issues avoid commitment. TOB-ROOK-002

❑ **Review all type conversions within the system, and ensure that they are as robust as possible.** This should include helper functions to validate types after conversion, and ensure that some minimal level of sanity checking can be performed in a centralized and simple way, regardless of where it is needed throughout the code base. TOB-ROOK-003

❑ **Audit all locations where secondary checks are not returned, such as `ok` for conversions, or `err` for normal error handling, and ensure that these secondary values are validated.** Ensure execution flow changes correctly in response to errors. TOB-ROOK-004

❑ **Determine a data classification system that may be applied to all data handled by Rook itself.** This may mirror the classification of the larger Kubernetes system. This system will also help ensure that developers know which data may and may not be displayed within ancillary sources, such as logs, and allow for the development of logging filters to remove such information should it be accidentally included within logs. TOB-ROOK-005

❑ **Centralize file handling into specific locations that may be easily audited and updated.** This will ensure that if updates need to be made, they can be made to a single location and are easily audited. File creation outside of approved areas should be authorized on a case-by-case basis. TOB-ROOK-006

❑ **Ensure all usage of random number generators is appropriate for the sensitivity of their operations.** Cryptographically secure random number generators should be applied whenever the numbers are used in sensitive operations such as password generation and session handling. TOB-ROOK-007

❑ **Consider defaulting to newer hash methods, such as SHA2.** TOB-ROOK-008

❑ **Consider using failure backoffs to prevent constant failure cycles.** Ensure validation propagates through all appropriate processes to prevent validation-induced failure cycles. TOB-ROOK-009

❑ **Enumerate the validations on the CRD and operator level to ensure appropriate validations are being performed in a centralized location where changes cascade across all validation routines.** TOB-ROOK-010

❑ **Centralize input and output encoding and decoding.** Avoid construction and parsing outside of these central locations to ensure validations propagate throughout the project.

❑ **Consider deprecating password generation and offloading password specification to the users of Rook.** Performing CRD-based length validation of the user-provided password could augment this change. Consider aligning with a standard or guideline on handling passwords, such as those standardized by NIST or ISO. TOB-ROOK-012

❑ **Improve validation related to the removal of finalizers.** Check to ensure all finalizers have been removed after the removal process has finished. TOB-ROOK-013

Findings Summary

#	Title	Type	Severity
1	Variable shadowing results in potential misconfiguration	Configuration	Medium
2	Ineffectual assignments result in unused variable values	Undefined Behavior	Medium
3	Potential runtime panics from unhandled type assertions	Error Handling	Medium
4	Improperly handled type assertion error could result in invalid execution state	Error Handling	Medium
5	Logging of sensitive information	Logging	High
6	Insecure file and directory permissions	Access Controls	Medium
7	Insecure PRNG used to generate Ceph dashboard password	Cryptography	Low
8	Use of insecure cryptographic hashing function	Cryptography	Low
9	Failure in edge-fs operator resulting in cluster creation failure loop	Error Handling	Low
10	Lack of validation leads to failure to deploy Ceph object	Data Validation	Low
11	Missing input and output encodings	Data Validation	Medium
12	Default Ceph dashboard credentials are easily brute-force-able	Authentication	Medium
13	Incorrect error handling could result in finalizer not being removed	Error Handling	Low

1. Variable shadowing results in potential misconfiguration

Severity: Medium
Difficulty: Low
Type: Configuration
Finding ID: TOB-ROOK-001
Target: `rook/pkg/operator/edgefs/cluster/configmap.go`

Description

Due to variable shadowing, a misconfiguration of a `ConfigMap`'s values could occur. The `serverIfName` variable is defined in the top-level scope of the `createClusterConfigMap` function, but within a nested `if` statement, the variable is re-defined using the shorthand assignment operator. This results in a shadowed declaration, which does not propagate values to the top-level declaration of `serverIfName`.

```
func (c *cluster) createClusterConfigMap(deploymentConfig edgefsv1.ClusterDeploymentConfig,
resurrect bool) error {
...
	serverIfName := defaultServerIfName
	brokerIfName := defaultBrokerIfName

	serverSelector, serverDefined := c.Spec.Network.Selectors["server"]
	brokerSelector, brokerDefined := c.Spec.Network.Selectors["broker"]
...
		if serverDefined && brokerDefined {
			var err error
			serverIfName, err = k8sutil.GetMultusIfName(serverSelector)
			if err != nil {
				return err
			}
...
		} else if serverDefined {
			serverIfName, err := k8sutil.GetMultusIfName(serverSelector)
			if err != nil {
				return err
			}

			brokerIfName = serverIfName
		} else if brokerDefined {
			serverIfName, err := k8sutil.GetMultusIfName(brokerSelector)
			if err != nil {
				return err
			}
...
```

Figure TOB-ROOK-001.1: The snippet of the `createClusterConfigMap` *definition, withpertinent lines highlighted in red.*

Exploit Scenario

A cluster specification declares the network as Multus. Subsequently, the network “server” or “broker” selector is defined in a mutually exclusive way. The `serverIfName` now will not be propagated into the final configuration due to its shadowed declaration, resulting in a misconfiguration.

Recommendation
Short term, fix the variable shadowing for the `serverIfName` variable. Improve testing to cover all branches, detecting misconfiguration in the final result.

Long term, integrating static analysis into the CI/CD pipeline to detect variable shadowing in submitted code is highly recommended. The [shadow](#) tool for [`go vet`](#) is recommended.

2. Ineffectual assignments result in unused variable values

Severity: Medium
Type: Undefined Behavior
Target: `Multiple locations`

Difficulty: Low
Finding ID: TOB-ROOK-002

Description

Across the codebase, there are several instances in which ineffectual assignment could result in unintentional side effects due to unused variable values. While not all of these instances are directly problematic, future changes could result in undesirable effects.

Ineffectual assignment occurs when a variable is assigned a value that is never referenced before reassignment. This is not caught by the compiler's unused variable check, since the unused variable check evaluates whether the variable is ever used within its declaration scope.

An example can be seen in Figure TOB-ROOK-002.1, where an error from the `jobsClient.Watch` function is never checked before it is reassigned to the error result of the `jobsClient.Get` call. While not exhaustive, further examples are detailed in Figures TOB-ROOK-002.2-6.

```
func (c *Cluster) waitJob(job *batch.Job) error {
	batchClient := c.context.Clientset.BatchV1()
	jobsClient := batchClient.Jobs(job.ObjectMeta.Namespace)
	watch, err := jobsClient.Watch(metav1.ListOptions{LabelSelector: "job-name=" +
job.ObjectMeta.Name})


	k8sjob, err := jobsClient.Get(job.ObjectMeta.Name, metav1.GetOptions{})
	if err != nil {
		return fmt.Errorf("Failed to get job %s", job.ObjectMeta.Name)
	}
...
}
```

Figure TOB-ROOK-002.1: An example of ineffectual assignment resulting in an unchecked error in the `rook/pkg/operator/edgefs/cluster/prepare.waitJob` *function. Pertinent lines are highlighted in red.*

```
func getMonitoringClient() (*monitoringclient.Clientset, error) {
	cfg, err := clientcmd.BuildConfigFromFlags("", "")
	client, err := monitoringclient.NewForConfig(cfg)
	if err != nil {
		return nil, fmt.Errorf("failed to get monitoring client. %+v", err)
	}
	return client, nil
}
```

Figure TOB-ROOK-002.2: The `rook/pkg/operator/k8sutil/prometheus.getMonitoringClient` *function, where the error value of* `clientcmd.BuildConfigFromFlags` *is ignored. Pertinent lines are highlighted in red.*

```
func (c *ISGWController) isgwContainer(svcname, name, containerImage string, isgwSpec
edgefsv1.ISGWSpec) v1.Container {
...
	replication := 3
	if isgwSpec.ReplicationType == "initial" {
		replication = 1
	} else if isgwSpec.ReplicationType == "continuous" {
		replication = 2
	} else {
		replication = 3
	}
...
}
```

Figure TOB-ROOK-002.3: A snippet of the `rook/pkg/operator/edgefs/isgw.isgwContainer` *function, where the* `replication` *variable's initial declaration is ineffectual because all branches of the subsequent* `if` *statement reassign its value. Pertinent lines are highlighted in red.*

```
func Add(mgr manager.Manager, context *controllerconfig.Context) error {
...
	// create a new controller
	c, err := controller.New(controllerName, mgr, controller.Options{Reconciler:
reconciler})
	if err != nil {
		return err
	}


	// Watch for the machines and enqueue the machineRequests if the machine is occupied
by the osd pods
	err = c.Watch(&source.Kind{Type: &mapiv1.Machine{}},
&handler.EnqueueRequestsFromMapFunc{
		ToRequests: handler.ToRequestsFunc(func(obj handler.MapObject)
[]reconcile.Request {
			clusterNamespace, isNamespacePresent :=
obj.Meta.GetLabels()[MachineFencingNamespaceLabelKey]
			if !isNamespacePresent || len(clusterNamespace) == 0 {
				return []reconcile.Request{}
			}
			clusterName, isClusterNamePresent :=
obj.Meta.GetLabels()[MachineFencingLabelKey]
			if !isClusterNamePresent || len(clusterName) == 0 {
				return []reconcile.Request{}
			}
			req := reconcile.Request{NamespacedName: types.NamespacedName{Name:
clusterName, Namespace: clusterNamespace}}
			return []reconcile.Request{req}
		}),
	})
```

```
        // Watch for the osd pods and enqueue the CephCluster in the namespace from the pods
        return ...

}
```

Figure TOB-ROOK-002.4: In the `rook/pkg/operator/ceph/disruption/machinelabel.Add` *function, the* `err` *variable is reassigned to a new value, and is not checked before the function returns. This results in an unchecked error value. Pertinent lines are highlighted in red.*

```
func (c *Cluster) completeOSDsForAllNodes(config *provisionConfig, configOSDs bool,
timeoutMinutes int) bool {
...
        for {
...
                for {
                        select {
                        case e, ok := <-w.ResultChan():
                                if !ok {
...
                                        leftNodes := 0
                                        leftRemainingNodes := util.NewSet()
                                        leftNodes, leftRemainingNodes, completed, statuses, err
= c.checkNodesCompleted(selector, config, configOSDs)
...
                                }
...

                        }
                }
        }
}
```

Figure TOB-ROOK-002.5: In the `rook/pkg/operator/ceph/cluster/osd.completeOSDsForAllNodes` *function there are two sequential ineffectual assignments. The* `leftNodes` *and* `leftRemainingNodes` *variables are declared and assigned values that are immediately replaced by the* `c.checkNodesCompleted` *function's return values. Pertinent lines are highlighted in red.*

```
func GetModifiedRookImagePath(originRookImage, addon string) string {
...
        modifiedImageName := "edgefs"
...
        if len(addon) > 0 {
                modifiedImageName = fmt.Sprintf("%s-%s", imageVersionParts[0], addon)
        } else {
                modifiedImageName = fmt.Sprintf("%s", imageVersionParts[0])
        }
...
}
```

Figure TOB-ROOK-002.6: In the `rook/pkg/apis/edgefs.rook.io/v1.GetModifiedRookImagePath` *function, there is an* `if` *statement that results in the reassignment of the* `modifiedImageName` *variable in all branches, rendering the initial declaration ineffectual. Pertinent lines are highlighted in red.*

Exploit Scenario
In the context of Figure TOB-ROOK-002.1, an error occurs in a call to the `jobsClient.Watch` function when invoking the `waitJob` function. The error is subsequently ignored, and replaced by the error value of `jobsClient.Get`. Therefore, the function continues execution in a potentially erroneous state.

Recommendation
Short term, audit the locations noted above, and determine if the ineffectual assignment is problematic. Areas where default values aren't used may be replaced with a `var` declaration, and areas where errors or return values are ignored may be replaced with `_`.

Long term, use the [`ineffassign`]() tool as part of the larger set of CI/CD static analysis tools, and audit all locations noted during builds. Furthermore, there may be a case to be made that developers use such tools as part of a pre-commit hook in git so these issues avoid commitment.

References
- Ineffassign: Detect ineffectual assignments in Go

3. Potential runtime panics from unhandled type assertions

Severity: Medium
Difficulty: Medium
Type: Error Handling
Finding ID: TOB-ROOK-003
Target: `Multiple locations`

Description
When the syntax `castedValue, ok := someValue.(SomeType)` is used, ok captures errors that may occur when `someValue` is casted. However, throughout the codebase the alternative syntax `castedValue := someValue.(SomeType)` is used, where errors are not captured from the assertion. If this type of assertion fails, a runtime panic will occur.

```
newCluster := obj.(*cassandrav1alpha1.Cluster)
```

Figure TOB-ROOK-003.1: An example of a type assertion that could result in a runtime panic in the `rook/pkg/operator/cassandra/controller.New` *function.*

- rook/pkg/operator/cassandra/controller/controller.go:L124, L128 - 129, L148 - 149, L168, L175 - 176
- rook/pkg/operator/cassandra/sidecar/sidecar.go:L126,L134 - 135, L146
- rook/pkg/operator/ceph/disruption/nodedrain/add.go:L65 - 66
- rook/pkg/operator/ceph/nfs/controller.go: L113 - 114, L149
- rook/pkg/operator/cockroachdb/controller.go: L131, L185
- rook/pkg/operator/edgefs/cluster/controller.go: L120
- rook/pkg/operator/edgefs/cluster/controller.go: L256 - 257

Exploit Scenario
A new version of Rook is deployed to a cluster while an existing operator is running. As a part of that deployment, attributes of the Custom Resource Definition (CRD) are modified. When the existing operator is notified of an event and receives a modified CRD, and then attempts to cast the value into its structure, a runtime panic occurs due to incompatibility.

Recommendation
Short term, audit all locations noted above, switch to the slightly longer form, and include a check of the ok variable. This will ensure that conversions are at least minimally checked prior to continuing execution flow, and prevent panics due to incorrect type conversions.

Long term, review all type conversions within the system, and ensure that they are as robust as possible. This should include helper functions to validate types after conversion, and ensure that some minimal level of sanity checking can be performed in a centralized and simple way, regardless of where it is needed throughout the code base.

4. Improperly handled type assertion error could result in invalid execution state

Severity: Medium
Difficulty: Medium
Type: Error Handling
Finding ID: TOB-ROOK-004
Target: `rook/pkg/operator/ceph/cluster/controller.go`

Description

In the `onK8sNodeAdd` function, there is an attempt to assert the `obj` variable into a `v1.Node` object. If this assertion fails, execution flow is not altered, and the error is only logged before continuing execution. This could result in an invalid execution state, since `newNode`'s value may be incorrect.

```
func (c *ClusterController) onK8sNodeAdd(obj interface{}) {
	newNode, ok := obj.(*v1.Node)
	if !ok {
		logger.Warningf("Expected NodeList but handler received %#v", obj)
	}
...
}
```

Figure TOB-ROOK-004.1: A snippet of the `onK8sNodeAdd` function, highlighting a type assertion in which if an assertion fails, execution flow is not altered (no return), resulting in a potentially erroneous continuation of execution. Pertinent lines are highlighted in red.

Exploit Scenario

The `onK8sNodeAdd` function receives an `obj` value unable to be cast as a `v1.Node`. Because the error is only logged and execution continues, the operator proceeds in an erroneous state.

Recommendation

Short term, change the execution flow to exit early or handle the situation where `ok` is false, rather than simply continuing onwards with an incorrect value.

Long term, audit all locations where secondary checks are not returned, such as `ok` for conversions, or `err` for normal error handling, and ensure that these secondary values are validated. Ensure execution flow changes correctly in response to errors.

5. Logging of sensitive information

Severity: High
Difficulty: Low
Type: Logging
Finding ID: TOB-ROOK-005
Target: Container logs, `rook/pkg/operator/ceph/cluster/mon.extractKey`, `rook/pkg/util/sys.Grep`

Description

When inspecting the logs of the `rook-ceph-mon-*` container, we observed logs including certificate data, keypair data, and administrator username and passwords.

Figure TOB-ROOK-005.1: A screenshot of the `rook-ceph-mon-*` container logs, which include certificate and keypair data in plain-text form.

```
debug 2019-12-05 21:51:06.458 7fc5f15ef700  0 log_channel(audit) log [DBG] :
from='client.4339 -' entity='client.admin' cmd=[{"username": "admin", "prefix": "dashboard
set-login-credentials", "password": "i5Swjd1Uon", "target": ["mgr", ""], "format": "json"}]:
dispatch
```

Figure TOB-ROOK-005.2: A snippet of the `rook-ceph-mon-*` container logs, which include username and password information for the Ceph dashboard. Pertinent lines are highlighted in red.

```go
func extractKey(contents string) (string, error) {
	secret := ""
	slice := strings.Fields(sys.Grep(string(contents), "key"))
	if len(slice) >= 3 {
		secret = slice[2]
	}
	if secret == "" {
		return "", fmt.Errorf("failed to parse secret")
	}
	return secret, nil
```

```
}
```

Figure TOB-ROOK-005.3: The `rook/pkg/operator/ceph/cluster/mon.extractKey` *function attempts to parse secrets from arbitrary string inputs by using the* `rook/pkg/util/sys.Grep` *function, which will log the input provided to it when debug logging is enabled. Pertinent lines are highlighted in red.*

```
func Grep(input, searchFor string) string {
	logger.Debugf("grep. search=%s, input=%s", searchFor, input)
	if input == "" || searchFor == "" {
		return ""
	}
	for _, line := range strings.Split(input, "\n") {
		if matched, _ := regexp.MatchString(searchFor, line); matched {
			logger.Debugf("grep found line: %s", line)
			return line
		}
	}
	return ""
}
```

Figure TOB-ROOK-005.4: The `rook/pkg/util/sys.Grep` *function definition, which contains debug logging of inputs provided to it. Pertinent lines are highlighted in red.*

Additionally, there are a number of locations where items such as secrets names and usernames are stored within the log above the `Debugf` level.

```
pkg/operator/ceph/csi/secrets.go:160:     logger.Infof("created kubernetes csi secrets for cluster %q", namespace)
pkg/operator/ceph/config/keyring/store.go:109:
logger.Warningf("failed to delete keyring secret for %s. user may need to delete the resource manually. %+v", secretName, err)
pkg/operator/ceph/object/user/controller.go:274:
logger.Warningf("failed to delete user %s secret. %+v",
fmt.Sprintf("rook-ceph-object-user-%s-%s", u.Spec.Store, u.Name), err)
pkg/operator/ceph/object/bucket/api-handlers.go:52:   logger.Infof("getting secret %q", namespace+"/"+name)
pkg/operator/minio/controller.go:241:          logger.Errorf("Unable to get secret with name=%s in namespace=%s: %v", secretName, namespace, err)
pkg/operator/ceph/object/user/controller.go:274:
logger.Warningf("failed to delete user %s secret. %+v",
fmt.Sprintf("rook-ceph-object-user-%s-%s", u.Spec.Store, u.Name), err)
pkg/operator/ceph/object/user/controller.go:277:      logger.Infof("user %s deleted successfully", u.Name)
pkg/operator/ceph/object/bucket/rgw-handlers.go:72:   logger.Infof("creating Ceph user %q", username)
```

```
pkg/operator/ceph/object/bucket/rgw-handlers.go:83:
logger.Infof("successfully created Ceph user %q with access keys",
username)
pkg/operator/ceph/object/bucket/rgw-handlers.go:90:   logger.Infof("deleting
Ceph user %s for bucket %q", username, p.bucketName)
pkg/operator/ceph/object/bucket/rgw-handlers.go:94:
logger.Infof("User %s successfully deleted", username)
pkg/operator/ceph/object/bucket/rgw-handlers.go:100:
logger.Infof("User %s does not exist", username)
pkg/operator/ceph/object/bucket/provisioner.go:97:    logger.Infof("set user
%q bucket max to %d", p.cephUserName, maxBuckets)
pkg/operator/ceph/object/bucket/provisioner.go:255:
logger.Infof("principal %q ejected from bucket %q policy. Output: %v",
p.cephUserName, p.bucketName, output)
pkg/operator/k8sutil/cmdreporter/cmdreporter.go:175:
logger.Errorf("continuing after failing delete job %s; user may need to
delete it manually. %+v", jobName, err)
pkg/operator/k8sutil/cmdreporter/cmdreporter.go:181:
logger.Errorf("continuing after failing to delete ConfigMap %s for job %s;
user may need to delete it manually. %+v",
```

Figure TOB-ROOK-005.5: Locations where secret names and related information are logged at Info level.

Exploit Scenario

An attacker is able to observe the logs of a Rook component that contain sensitive information the attacker can use for further activities such as authentication as an administrator, or performing monkey-in-the-middle attacks. Note that logs are often exposed to users of different sensitivity levels, such as lower-privileged administrators. Access to logs should not provide sufficient information to access consoles or other cluster components within the larger system.

Recommendation

Short term, audit all log locations, and note which information should and should not be included within the logs.

Long term, determine a data classification system that may be applied to all data handled by Rook itself. This may mirror the classification of the larger Kubernetes system. This will help ensure that developers know which data may and may not be displayed within ancillary sources, such as logs, and allow for the development of logging filters to remove such information should it be accidentally included within logs.

6. Insecure file and directory permissions

Severity: Medium
Type: Access Controls
Target: `Multiple locations`

Difficulty: Low
Finding ID: TOB-ROOK-006

Description

Throughout the repository there are areas in which files and directories are written and created with statically defined permissions. Many of these permissions are rather open, potentially allowing other system tenants to view and interact with their contents, which may be sensitive.

For example, Figure TOB-ROOK-006.1 shows how missing directories will be created with with `0755` permissions if they do not already exist.

```
func MountDeviceWithOptions(devicePath, mountPath, fstype, options string, executor
exec.Executor) error {
...
	os.MkdirAll(mountPath, 0755)
	cmd := fmt.Sprintf("mount %s", devicePath)
	if err := executor.ExecuteCommand(false, cmd, mountCmd, args...); err != nil {
		return fmt.Errorf("command %s failed: %+v", cmd, err)
	}
...
}
```

Figure TOB-ROOK-006.1: A snippet of the `rook/pkg/util/sys.MountDeviceWithOptions` *function definition. Pertinent lines are highlighted in red.*

Other locations that write files or create directories with loose permissions have been listed below.

- rook/cmd/rookflex/cmd.mountDevice
- rook/cmd/rookflex/cmd.mountCephFS
- rook/pkg/daemon/ceph/agent/flexvolume/manager/ceph.getClusterInfo
- rook/pkg/daemon/ceph/agent/flexvolume.configureFlexVolume
- rook/pkg/daemon/ceph/config.GenerateConfigFile
- rook/pkg/daemon/ceph/config.writeKeyring
- rook/pkg/daemon/ceph/osd.prepareOSDRoot
- rook/pkg/daemon/ceph/osd.repairOSDFileSystem
- rook/pkg/daemon/ceph/osd.createOSDFileSystem
- rook/pkg/operator/ceph/cluster/mon.createNamedClusterInfo
- rook/pkg/operator/cassandra/sidecar.generateCassandraConfigFiles
- rook/pkg/util.WriteFile
- rook/pkg/util/sys.MountDeviceWithOptions

Additionally, the application makes use of `MkdirAll`, which has a subtle semantic issue: Errors are not returned when a directory exists but is not of the requested permission. This may lead to a situation wherein an attacker has created directories prior to Rook, and retains control of these directories. Examples of such locations are noted within the list above.

Exploit Scenario

An attacker gains access to a host running a Rook component. Because the file and directory permissions are loose, the attacker is able to view potentially sensitive configuration values that could be used for accessing other privileged portions of the system.

Recommendation

Short term, audit all locations within the codebase, to ensure the correct file system permissions are used. These should be the most narrow permissions possible; for example, wherever possible, change `0644` (group and world readable, user writable and readable) into `0600` (only the owner may read). This will ensure that only the user executing Rook will be able to access sensitive information.

Long term, centralize file handling into specific locations that may be easily audited and updated. This will ensure that if updates need to be made, they can be made to a single location and are easily audited. File creation outside of approved areas should be authorized on a case-by-case basis.

7. Insecure PRNG used to generate Ceph dashboard password

Severity: Low
Difficulty: Low
Type: Cryptography
Finding ID: TOB-ROOK-007
Target: `rook/pkg/operator/ceph/cluster/mgr/dashboard.go`

Description
When generating the Ceph dashboard username and password, the `math/rand` package is used to create the random values composing the password. However, the `math/rand` package is cryptographically insecure, which could allow an attacker to abuse the values it generates.

```
func generatePassword(length int) string {
	const passwordChars =
"abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789"
	passwd := make([]byte, length)
	for i := range passwd {
		passwd[i] = passwordChars[rand.Intn(len(passwordChars))]
	}
	return string(passwd)
}
```

Figure TOB-ROOK-007: The `rook/pkg/operator/ceph/cluster/mgr.generatePassword` *function definition.*

Exploit Scenario
An attacker is able to predict the values produced by the random number generator, allowing them to identify the generated password.

Recommendation
Short term, use a cryptographically secure random number generator such as `crypto/rand`.

Long term, ensure all usage of random number generators is appropriate for the sensitivity of their operations. Cryptographically secure random number generators should be applied whenever the numbers are used in sensitive operations such as password generation and session handling.

8. Use of weak cryptographic hash algorithm

Severity: Low
Type: Cryptography
Target: `rook/pkg/operator/k8sutil/volume.go`, `rook/pkg/operator/k8sutil/k8sutil.go`

Difficulty: High
Finding ID: TOB-ROOK-008

Description

MD5 is used to generate part of a volume name. Due to the ease of creating an MD5 collision, an attacker could generate an input that collides with an existing volume.

```
func PathToVolumeName(path string) string {
...
	if len(volumeName) > validation.DNS1123LabelMaxLength {
		// keep an equal sample of the original name from both the beginning and from
the end,
		// and add some characters from a hash of the full name to help prevent name
collisions.
		// Make room for 3 hyphens in the middle (~ellipsis) and 1 hyphen to separate
the hash.
		hashLength := 8
		sampleLength := int((validation.DNS1123LabelMaxLength - hashLength - 3 - 1) /
2)
		first := volumeName[0:sampleLength]
		last := volumeName[len(volumeName)-sampleLength:]
		hash := Hash(volumeName)
		volumeName = fmt.Sprintf("%s---%s-%s", first, last, hash[0:hashLength])
	}


	return volumeName
}
```

Figure TOB-ROOK-008.1: A snippet of the `src/rook/pkg/operator/k8sutil.PathToVolumeName` *function definition, highlighting the use of the* `Hash` *function, which is a wrapper around an MD5 hash function. Pertinent lines are highlighted in red.*

```
func Hash(s string) string {
	return fmt.Sprintf("%x", md5.Sum([]byte(s)))
}
```

Figure TOB-ROOK-008.2: The `rook/pkg/operator/k8sutil.Hash` *function definition. Pertinent lines are highlighted in red.*

Exploit Scenario

An attacker modifies a known good application binary, padding it to create an MD5 hash matching another volume hash, which leads to data validation errors.

Recommendation

Short term, use newer hash methods such as SHA2 that are less prone to collision. This will help prevent attackers from abusing hash-based areas of the application.

Long term, consider defaulting to newer hash methods, such as SHA2.

9. Failure in edge-fs operator resulting in cluster creation failure loop

Severity: Low
Type: Error Handling
Target: `rook-edgefs-operator`

Difficulty: Medium
Finding ID: TOB-ROOK-009

Description

When an `edgefs-cluster` is quickly created and deleted, there appears to be a race condition that leads to a cluster creation failure loop when encountering a data validation error.

The problem occurs when the CRD object for the cluster is destroyed while the `rook-edgefs-operator` is still in the process of creating the queued cluster. Then the CRD is hidden, a validation error occurs based on the previously present cluster config, and the loop repeats. The cluster operator must then be restarted to break the loop and restore the operator to a normal state.

```
...
spec:
  edgefsImageName: edgefs/edgefs:1.2.64  # specify version here, i.e. edgefs/edgefs:1.1.0
etc
  serviceAccount: rook-edgefs-cluster
  dataDirHostPath: /data/rook
  dataVolumeSize: 10Gi
  #devicesResurrectMode: "restoreZapWait"
  dashboard:
   localAddr: 0.0.0.0:9999999
...
```

Figure TOB-ROOK-009.1: A snippet of the cluster specification, displaying the presence of both the `dataDirHostPath` *and* `dataVolumeSize` *definitions.*

```
2019-12-09 15:46:26.945206 E | edgefs-op-cluster: failed to get cluster from namespace
rook-edgefs prior to updating its status: clusters.edgefs.rook.io "rook-edgefs" not found
2019-12-09 15:46:26.945237 E | edgefs-op-cluster: Invalid cluster [rook-edgefs] spec. Error:
Both deployment options DataDirHostPath and DataVolumeSize are specified. Should be only one
deployment option in cluster specification
2019-12-09 15:46:26.945241 E | edgefs-op-cluster: failed to create cluster in namespace

rook-edgefs. Both deployment options DataDirHostPath and DataVolumeSize are specified.

Should be only one deployment option in cluster specification
```

Figure TOB-ROOK-009.2: The logs generated by the `rook-edgefs-operator` *during the failure loop.*

```
$ kubectl delete -f cluster.yaml
Error from server (NotFound): error when deleting "cluster.yaml": namespaces "rook-edgefs"
not found
```

```
Error from server (NotFound): error when deleting "cluster.yaml": serviceaccounts
"rook-edgefs-cluster" not found
Error from server (NotFound): error when deleting "cluster.yaml":
roles.rbac.authorization.k8s.io "rook-edgefs-cluster" not found
Error from server (NotFound): error when deleting "cluster.yaml":
rolebindings.rbac.authorization.k8s.io "rook-edgefs-cluster-mgmt" not found
Error from server (NotFound): error when deleting "cluster.yaml":
rolebindings.rbac.authorization.k8s.io "rook-edgefs-cluster" not found
Error from server (NotFound): error when deleting "cluster.yaml": podsecuritypolicies.policy
"privileged" not found
Error from server (NotFound): error when deleting "cluster.yaml":
clusterroles.rbac.authorization.k8s.io "privileged-psp-user" not found
Error from server (NotFound): error when deleting "cluster.yaml":
clusterrolebindings.rbac.authorization.k8s.io "rook-edgefs-system-psp" not found
Error from server (NotFound): error when deleting "cluster.yaml":
clusterrolebindings.rbac.authorization.k8s.io "rook-edgefs-cluster-psp" not found
Error from server (NotFound): error when deleting "cluster.yaml": clusters.edgefs.rook.io

"rook-edgefs" not found
```

Figure TOB-ROOK-009.3: When one attempts to delete the CRD entries used by the `rook-edgefs-operator` *during the failure loop,* `kubectl` *reports they have already been removed.*

Exploit Scenario
An attacker with the ability to create a cluster could use this bug to force the `rook-edgefs-operator` to fall into a failure loop, requiring restarting of the operator and potentially interrupting its service.

Recommendation
Short term, improve error handling within the operator. Ensure validation of the CRD entry leads to the appropriate execution flow change to prevent failure loops.

Long term, consider using failure backoffs to prevent constant failure cycles. Ensure validation propagates through all appropriate processes to prevent validation-induced failure cycles.

10. Lack of validation leads to failure to deploy Ceph object

Severity: Low
Type: Data Validation
Target: `rook-ceph-operator`

Difficulty: Low
Finding ID: TOB-ROOK-010

Description

When creating a `CephObjectStore` entry, there is no validation for the CRD object's gateway `port` attribute (Figure TOB-ROOK-010.1). Therefore, when one attempts to create a CRD entry with an invalid `port` value (Figure TOB-ROOK-010.2), it is accepted by the CRD API (Figure TOB-ROOK-010.3). The `rook-ceph-operator` subsequently attempts to create a Kubernetes `Service` pointing to the un-validated gateway address, but the `Service` CRD validation of the `port` fails and the `Service` is not created. Attempting to create the object again will fail because there is already an instance of the `CephObjectStore` (10.3).

```
...
spec:
  group: ceph.rook.io
  names:
    kind: CephObjectStore
    listKind: CephObjectStoreList
    plural: cephobjectstores
    singular: cephobjectstore
  scope: Namespaced
  version: v1
  validation:
    openAPIV3Schema:
      properties:
        spec:
          properties:
            gateway:
              properties:
                type:
                  type: string
                sslCertificateRef: {}
                port:
                  type: integer
...
```

Figure TOB-ROOK-010.1: A snippet of the `cluster/examples/kubernetes/ceph/common.yaml`, where the `CephObjectStore` CRD is defined.

```
apiVersion: ceph.rook.io/v1
kind: CephObjectStore
metadata:
  name: my-store
  namespace: rook-ceph
spec:
  metadataPool:
    replicated:
      size: 1
  dataPool:
```

```
    replicated:
      size: 1
  gateway:
    type: s3
    port: 999999999
    securePort:
    instances: 1
```

Figure TOB-ROOK-010.2: The `CephObjectStore` specification used to cause the error in the `rook-ceph-operator`.

```
$ kubectl create -f object-test.yaml
cephobjectstore.ceph.rook.io/my-store created
$ kubectl create -f object-test.yaml
Error from server (AlreadyExists): error when creating "object-test.yaml":
cephobjectstores.ceph.rook.io "my-store" already exists
```

Figure TOB-ROOK-010.3: The creation of the `CephObjectStore` entry and subsequent attempt to create it again after the error is encountered.

```
2019-12-09 14:20:19.160330 E | op-object: failed to create or update object store my-store.
failed to start rgw service. failed to create rgw service. Service "rook-ceph-rgw-my-store"
is invalid: [spec.ports[0].port: Invalid value: 999999999: must be between 1 and 65535,
inclusive, spec.ports[0].targetPort: Invalid value: 999999999: must be between 1 and 65535,
inclusive]
```

Figure TOB-ROOK-010.4: The log on the `rook-ceph-operator` detailing the failure to create a Kubernetes `Service` CRD entry for the object store gateway.

Exploit Scenario

An attacker creates a significant number of invalid `CephObjectStore` CRD entries, resulting in the `rook-ceph-operator` generating a large amount of errors until the entries are removed.

Recommendation

Short term, apply validation on the CRD layer as well as on the operator level for the `CephObjectStore` CRD entries.

Long term, enumerate the validations on the CRD and operator level to ensure appropriate validations are being performed in a centralized location where changes cascade across all validation routines.

11. Missing input and output encodings

Severity: Low
Type: Data Validation
Target: `Multiple locations`

Difficulty: Low
Finding ID: TOB-ROOK-011

Description
Across the Rook codebase there are components that will create structured content such as shell commands or JSON without a proper encoder, opting instead for `sprintf` or other similar construction methods. This could lead to an attacker-controlled input to influence the final result of the structured content in a malicious or unintended way.

```
% ack --go 'Sprintf\(.\{'
#...

pkg/daemon/ceph/config/info.go
66:             mons = append(mons, fmt.Sprintf("{Name: %s, Endpoint: %s}", m.Name,
m.Endpoint))

#...

pkg/operator/edgefs/cluster/utils.go
277:    patch := fmt.Sprintf(`{"metadata":{"labels":%v}}`, labelString)
```

Figure TOB-ROOK-011: A few example locations using `Sprintf` *to construct a JSON-like structure, identified through an att*`ack` *of the codebase.*

Exploit Scenario
An attacker is able to provide malicious input through Rook's configuration interface, allowing injection of arbitrary commands or JSON values to the final constructed value.

Recommendation
Short term, use proper encoders to create structured content. Ensure values are properly escaped and validated.

Long term, centralize input and output encoding and decoding. Avoid construction and parsing outside of these central locations to ensure validations propagate throughout the project.

12. Default Ceph dashboard credentials are easily brute–force-able

Severity: Medium
Difficulty: Low
Type: Authentication
Finding ID: TOB-ROOK-012
Target: `Ceph dashboard configured by Rook`

Description
The default passwords generated by Rook for use as the dashboard's admin user are too short, leading to extremely simple brute-force attacks. By default, the generated password is a length of 10, composed of random values from Figure TOB-ROOK-011.1.

```
abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789
```

Figure TOB-ROOK-011.1: The values a Ceph dashboard password can be composed of.

Exploit Scenario
An attacker identifies an exposed Ceph dashboard initialized by Rook. They subsequently brute-force the "admin" user's default password quickly due to the short length and limited valid characters.

Recommendation
Short term, increase the default length of the passwords and allow customizable lengths.

Long term, consider deprecating password generation and offloading password specification to the users of Rook. Performing CRD-based length validation of the user-provided password could augment this change. Consider aligning with a standard or guideline on handling passwords, such as those standardized by NIST or ISO.

13. Incorrect error handling could result in finalizer not being removed

Severity: Low
Difficulty: Low
Type: Error Handling
Finding ID: TOB-ROOK-013
Target: `rook/pkg/operator/edgefs/cluster/controller.go`

Description

This is faulty logic surrounding the removal of a finalizer. If the cast of `obj` fails, the finalizer will be reported as removed.

If `obj` is not successfully cast into an `edgefsv1.Cluster`, ok will be `false`, resulting in the `.Update(cluster)` operation not running, and the declared `err` variable retaining its default `nil` value. Subsequently, a check against `err` is performed, which will evaluate to `false`, resulting in logging the finalizer as removed when it in fact was not.

```
func (c *ClusterController) removeFinalizer(obj interface{}) {
...
	// update the crd to remove the finalizer for good. retry several times in case of
intermittent failures.
	maxRetries := 5
	retrySeconds := 5 * time.Second
	for i := 0; i < maxRetries; i++ {
		var err error
		if cluster, ok := obj.(*edgefsv1.Cluster); ok {
			_, err =
c.context.RookClientset.EdgefsV1().Clusters(cluster.Namespace).Update(cluster)
		}


		if err != nil {
			logger.Errorf("failed to remove finalizer %s from cluster %s. %+v",
fname, objectMeta.Name, err)
			time.Sleep(retrySeconds)
			continue
		}
		logger.Infof("removed finalizer %s from cluster %s", fname, objectMeta.Name)
		return
	}


	logger.Warningf("giving up from removing the %s cluster finalizer", fname)
}
```

Figure TOB-ROOK-013.1: A snippet of the `removeFinalizer` *function, highlighting the incorrect error handling logic which could lead to a finalizer not being correctly removed.*

Exploit Scenario

The `removeFinalizer` function receives an object that fails to be cast to an `edgefsv1.Cluster` object. This results in the finalizer being reported as removed despite the fact that it still remains in the cluster.

Recommendation
Short term, fix the logic to correctly report type assertion errors and refrain from reporting the finalizer as removed.

Long term, improve validation related to the removal of finalizers. Check to ensure all finalizers have been removed after the removal process has finished.

A. Vulnerability Classifications

Vulnerability Classes	
Class	**Description**
Access Controls	Related to authorization of users and assessment of rights
Auditing and Logging	Related to auditing of actions or logging of problems
Authentication	Related to the identification of users
Configuration	Related to security configurations of servers, devices or software
Cryptography	Related to protecting the privacy or integrity of data
Data Exposure	Related to unintended exposure of sensitive information
Data Validation	Related to improper reliance on the structure or values of data
Denial of Service	Related to causing system failure
Error Reporting	Related to the reporting of error conditions in a secure fashion
Patching	Related to keeping software up to date
Session Management	Related to the identification of authenticated users
Timing	Related to race conditions, locking or order of operations
Undefined Behavior	Related to undefined behavior triggered by the program

Severity Categories	
Severity	**Description**
Informational	The issue does not pose an immediate risk, but is relevant to security best practices or Defense in Depth
Undetermined	The extent of the risk was not determined during this engagement
Low	The risk is relatively small or is not a risk the customer has indicated is important
Medium	Individual user's information is at risk, exploitation would be bad for client's reputation, moderate financial impact, possible legal

	implications for client
High	Large numbers of users, very bad for client's reputation, or serious legal or financial implications

Difficulty Levels	
Difficulty	**Description**
Undetermined	The difficulty of exploit was not determined during this engagement
Low	Commonly exploited, public tools exist or can be scripted that exploit this flaw
Medium	Attackers must write an exploit, or need an in-depth knowledge of a complex system
High	The attacker must have privileged insider access to the system, may need to know extremely complex technical details or must discover other weaknesses in order to exploit this issue

B. Code Quality Recommendations

The Code Quality Recommendations appendix details areas of Rook that could be improved, but do not equate to problems impacting the project's security posture. These recommendations are proposed in an effort to prevent future errors from occurring, and to improve the quality of future code contributions.

- **Be consistent when declaring variables to capture errors.** Avoid reassignment of these variables within different scopes. This will reduce the number of false-positives when running static analysis tools such as `ineffassign`.
- **Ensure errors are propagated to the caller.** When error values are ignored and not raised to the caller, this could lead to execution with an invalid program state.
- **Do not ignore error values returned by functions.** In the event an error occurs when executing a function, the error should be almost always be logged at a minimum.
- **Avoid deferring any function that can return an error.** When a deferred function is executed, the error values are not propagated to the caller, resulting in an unchecked error value.
- **Standardize the use of panic and returned errors.** Because the semantics of returning an error value and raising a panic are very different and require different handling techniques, this could lead to unintended runtime consequences if not carefully considered.
- **Ensure in-line documentation is up-to-date.** Certain areas of the project may have conflicting documentation which could lead to incorrect assumptions by the developers when introducing new features.
- **Centralize validations into appropriate packages.** Enforce the use of these packages throughout the project. This will allow changes to validations to be enforced across the package, and prevent disparate components from having different validation parameters.
- **Consider standardizing the verbiage of logs.** In some areas the logging could be confusing if the code path executing it is not traversed. This could make it difficult for operators without in-depth knowledge of the implementation to derive value from the produced logs.

C. Static Analysis Recommendations

To help improve the quality of code within the Rook codebase, there are several static analysis tools available for integration in both Git pre-commit hooks as well as CI/CD pipelines.

- [`Go-sec`] is a static analysis utility that looks for a variety of problems in Go codebases. Notably, `go-sec` will identify potential stored credentials, unhandled errors, cryptographically concerning packages, and similar types of problems.
- [`Go-vet`] is a very popular static analysis utility that searches for more go-specific problems within a codebase such as mistakes pertaining to closures, marshaling, and unsafe pointers. `Go-vet` is integrated within the `go` command itself, with support for other tools through the `vettool` command line flag.
- [`Staticcheck`] is a static analysis utility that identifies both stylistic problems and implementation problems within a Go codebase. Note: Many of the stylistic problems `staticcheck` identifies are also indicative of potential "problem areas" within a project.
- [`Ineffassign`] is a static analysis utility that identifies ineffectual assignments. These ineffectual assignments often identify situations in which errors go unchecked, which could lead to undefined behavior of the program due to execution in an invalid program state.
- [`Errcheck`] is a static analysis utility that identifies situations in which errors are not handled appropriately.

By executing these tools within the Git pre-commit hooks, code can be analyzed for potential problems prior to producing a commit that will be sent to a remote. This will help developers fix problems before a CI/CD pipeline detects them and requires remediation. Additionally, integrating these tools into the CI/CD pipeline will allow double-checking to ensure these problems are not introduced into the remote.

D. Golang fuzzing and property testing best practices

Golang has three notable fuzzing and property testing frameworks developers can use to test their codebases. Both `dvyukov/go-fuzz` and `google/gofuzz` allow developers to fuzz their codebases using different methods of test generation. For property testing, `leanovate/gopter` can be used as a framework to add property testing to your existing testing suite.

dvyukov/go-fuzz

The `dvyukov/go-fuzz` package provides an AFL-like mutational fuzzing interface where testing harnesses can be built entirely in Go. This framework is typically used when a library implemented in Go parses, interprets or otherwise interacts with blobs of data. An example of such a use-case can be seen in Figure D.1, where a harness for the Go standard library's image processing library is defined.

```
package png

import (
	"bytes"
	"image/png"
)

func Fuzz(data []byte) int {
	png.Decode(bytes.NewReader(data))
	return 0
}
```

Figure D.1: An example test harness for the `png.Decode` *function, as seen in the official readme.*

In this example, the function `Fuzz` accepts an array of bytes as data. `data` is then converted into a `Reader` for the `png.Decode` function to read from. When this is compiled and invoked, the function is executed repeatedly, where `data` is the input generated for each test case execution.

To help `go-fuzz` optimize the generation of test case inputs, the use of return values is important to understand. Typically, a panic indicates a crash with a given test case input. However, when there is no crash, but instead errors are raised gracefully, or no errors are raised, return values can be used to help guide `go-fuzz` to mutating inputs appropriately.

- Returning a value of `1` indicates the input generator should increase the priority of a given input during subsequent fuzzing.
- Returning `-1` indicates the input generator should never be added to the corpus, despite added coverage.

- In all other cases, the function should return `0`.

To build and run this example, you must have Go installed, with the `go-fuzz` package downloaded and installed. You can then traverse into the directory where Figure D.1 is stored, and execute `go-fuzz-build` (Figure D.2). Assuming the harness builds correctly, it will produce a Zip file for use with the `go-fuzz` executor. To start the fuzzing harness, you can execute `go-fuzz` in the same directory as the Zip file produced by `go-fuzz-build` (Figure D.3). This will create three directories if they do not already exist.

```
user@host:~/Desktop/png_fuzz$ ls
png_harness.go
user@host:~/Desktop/png_fuzz$ go-fuzz-build
user@host:~/Desktop/png_fuzz$ ls
png-fuzz.zip  png_harness.go
```

Figure D.2: The generated `png-fuzz.zip` *package, used by* `go-fuzz`.

```
user@host:~/Desktop/png_fuzz$ go-fuzz
2019/09/14 16:00:37 workers: 2, corpus: 30 (0s ago), crashers: 0, restarts: 1/0, execs: 0
(0/sec), cover: 0, uptime: 3s
2019/09/14 16:00:40 workers: 2, corpus: 31 (2s ago), crashers: 0, restarts: 1/0, execs: 0
(0/sec), cover: 205, uptime: 6s
2019/09/14 16:00:43 workers: 2, corpus: 31 (5s ago), crashers: 0, restarts: 1/6092, execs:
48742 (5415/sec), cover: 205, uptime: 9s
2019/09/14 16:00:46 workers: 2, corpus: 31 (8s ago), crashers: 0, restarts: 1/7829, execs:
101779 (8481/sec), cover: 205, uptime: 12s
2019/09/14 16:00:49 workers: 2, corpus: 31 (11s ago), crashers: 0, restarts: 1/8147, execs:
146656 (9777/sec), cover: 205, uptime: 15s
2019/09/14 16:00:52 workers: 2, corpus: 31 (14s ago), crashers: 0, restarts: 1/8851, execs:
203582 (11310/sec), cover: 205, uptime: 18s
2019/09/14 16:00:55 workers: 2, corpus: 31 (17s ago), crashers: 0, restarts: 1/8950, execs:
259563 (12360/sec), cover: 205, uptime: 21s
^C2019/09/14 16:00:56 shutting down...
```

Figure D.3: The CLI output of running `go-fuzz` *with the* `png-fuzz.zip` *package.*

The created directories contain suppressions, crashers, and corpus respectively (Figure D.4). The suppressions are used to prevent collecting the same message values every time, polluting your crasher samples. The crashers are crashdumps -- `STDOUT` and `STDERR` of the program when the test case input causes an error. Finally, the corpus directory stores the test case inputs used throughout the test harness's execution. This directory will collect mutated versions of each input as necessary.

```
user@host:~/Desktop/png_fuzz$ ls -R
.:
```

```
corpus  crashers  png-fuzz.zip  png_harness.go  suppressions

./corpus:
21339f0e4b8b5a8e0cb5471f1f91907d1917be50-6
215d99d0c7acdec5ad4c5aa8bec96a171b9ffae0-8
22f545ac6b50163ce39bac49094c3f64e0858403-11
401dfa141de03ca247cec609b6a4dc49994c2402-6
49a4f52fc4d746f3f4dec38bd938bb2c2b0c708f-6
4caece539b039b16e16206ea2478f8c5ffb2ca05-3
51d8c3ea9d7b4057e7949793f75458182336a1bb-4
530cd11f12f83150867fdbedf31615d3b0135d44-10
5f31ee6271c5da037d8e46728ae052b53d738617-6
6360600b950c224b0d0dcbd452e4c2f90a2924b7-1
63f1ad1e8d8f91c460b1c35b5979a7a37f227b8e-7
6dd3bf016eb315f97742289b2891af90b0df1a24-7
7026481ed4bb73bba3effe416c275d42051965be-10
7aa378903ac0080b1e269cabcacf5c1b0911109c-10
7ee4a2407681d5ded0964b66eea10801ea9b0407-4
8db669322f78937de3339bdf2b89fb5d3ee5960f-7
92ce0d9d3d34ccc29ffe91b142174481fdbe656c-5
996ac2854d4a198f638810b343ac0f3fafcaec72-8
a102542a2e88d5b2680eca9a18eec1fc0ba75ac0-5
a29c300c65044ee972b01136c21ecd8a4f091086-7
ab77dc8751b59c12fdb504740105f1a36cf964ef-5
ba32cf320b6ad86b2cb5bab7402aa3f2dd5a6939-6
bc1e7140b449aa59a5b62d2c039571588c6fddae-10
c8cbabd4b5dce54df043a71a71e4f4771ad185a3-10
da39a3ee5e6b4b0d3255bfef95601890afd80709
e003e82cdb7540a8e945834049e14eb3d577813a-2
e0d989e9f2fe64153068730eb34fda0337e67af8-1
e6bb28032a66740ae9bcea57f294fdbd1201054d-8
fc3ee3d4b138abe46a5ace30e1af9c46606b97b2-6
fd804ad601856c0f1be2db5705c4fd587472f72f-7
fe5dbbcea5ce7e2988b8c69bcfdfde8904aabc1f-1

./crashers:

./suppressions:
```

Figure D.4: The directory and file output produced by `go-fuzz`*.*

While running the harness on a single machine can typically produce good results, `go-fuzz` also supports a clustered mode, allowing test harness execution to scale horizontally across an arbitrary number of worker nodes. More information on this functionality can be found within the repository's readme.

google/gofuzz

With the google/gofuzz package, you can easily populate Go structures with randomized values. The package does not provide any simple execution harness, and instead leaves it up to the developer to construct the execution harness. For many projects, the execution harness can be the existing unit and integration testing system.

As an example, Figure D.2 displays a harness for a structure `Compute`, allowing us to `add`, `subtract`, `multiply`, and `divide` two numbers `A` and `B` that are coerced to be between `0` and `9`. The harness will continuously loop, using `gofuzz` to repopulate the `Compute` structure with random values. The random values are then coerced to be a number between `0` and `9`, and all operations are executed against it to see if it will fail.

```
package main

import (
  "fmt"
  "github.com/google/gofuzz"
)

type Compute struct {
  A uint32
  B uint32
}

func (c *Compute) CoerceInt () { c.A = c.A % 10; c.B = c.B % 10; }
func (c Compute) Add () uint32 { return c.A + c.B }
func (c Compute) Subtract () uint32 { return c.A - c.B }
func (c Compute) Divide () uint32 { return c.A / c.B }
func (c Compute) Multiply () uint32 { return c.A * c.B }

func main(){
  // Our instance of Compute to use during fuzzing runs
  var inpCompute Compute

  // Create a new fuzzer instance with the default settings
  f := fuzz.New()

  // Loop forever until we crash
  for {
    // Apply the fuzzer inputs to the inpCompute instance
    f.Fuzz(&inpCompute)

    // Restrict ints to 0 - 9
    inpCompute.CoerceInt()

    // Provide pre-crash feedback
    fmt.Println("Attempting operations with A:", inpCompute.A, "B:", inpCompute.B)

    // Figure out which operation could fail
    inpCompute.Add()
    inpCompute.Subtract()
    inpCompute.Divide()
```

```
    inpCompute.Multiply()
  }
}
```

Figure D.5: The fuzzing execution harness for the `Compute` structure.

To run the execution harness, the standard `go build` and `go run` commands can be used, such as in Figure D.6. The `google/gofuzz` package must be installed for these commands to execute successfully.

```
user@host:~/Desktop/math_fuzz$ go run .
Attempting operations with A: 4 B: 3
Attempting operations with A: 5 B: 2
Attempting operations with A: 8 B: 7
Attempting operations with A: 3 B: 8
Attempting operations with A: 9 B: 4
Attempting operations with A: 2 B: 8
Attempting operations with A: 1 B: 1
Attempting operations with A: 1 B: 4
Attempting operations with A: 1 B: 6
Attempting operations with A: 3 B: 3
Attempting operations with A: 5 B: 4
Attempting operations with A: 2 B: 7
Attempting operations with A: 4 B: 5
Attempting operations with A: 0 B: 5
Attempting operations with A: 6 B: 8
Attempting operations with A: 6 B: 0
panic: runtime error: integer divide by zero

goroutine 1 [running]:
main.Compute.Divide(...)
        /home/user/Desktop/math_fuzz/main.go:16
main.main()
        /home/user/Desktop/math_fuzz/main.go:39 +0x199
exit status 2
```

Figure D.6: Running the execution harness, and observing a runtime error in the `Divide` function, where a division by 0 has been observed.

leanovate/gopter

The [`leanovate/gopter`]() package provides constructs developers can use to implement property testing. Like Google's `gofuzz` package, an execution harness is not provided, and the developer must construct one as necessary. For many projects, the existing project testing framework can be used.

As an example, the same `Compute` structure used within the `google/gofuzz` example will be repurposed to demonstrate a generic set of property tests, as seen in Figure D.7. In these tests, we are ensuring that the sequence of `CoerceInt` and an operation successfully execute with the provided inputs.

```
package main_test
import (
  "github.com/leanovate/gopter"
  "github.com/leanovate/gopter/gen"
  "github.com/leanovate/gopter/prop"
  "math"
  "testing"
)

type Compute struct {
  A uint32
  B uint32
}

func (c *Compute) CoerceInt () { c.A = c.A % 10; c.B = c.B % 10; }
func (c Compute) Add () uint32 { return c.A + c.B }
func (c Compute) Subtract () uint32 { return c.A - c.B }
func (c Compute) Divide () uint32 { return c.A / c.B }
func (c Compute) Multiply () uint32 { return c.A * c.B }

func TestCompute(t *testing.T) {
  parameters := gopter.DefaultTestParameters()
  parameters.Rng.Seed(1234) // Just for this example to generate reproducible results


  properties := gopter.NewProperties(parameters)

  properties.Property("Add should never fail.", prop.ForAll(
    func(a uint32, b uint32) bool {
      inpCompute := Compute{A: a, B: b}
      inpCompute.CoerceInt()
      inpCompute.Add()
      return true
    },
    gen.UInt32Range(0, math.MaxUint32),
    gen.UInt32Range(0, math.MaxUint32),
  ))

  properties.Property("Subtract should never fail.", prop.ForAll(
    func(a uint32, b uint32) bool {
      inpCompute := Compute{A: a, B: b}
      inpCompute.CoerceInt()
      inpCompute.Subtract()
      return true
    },
    gen.UInt32Range(0, math.MaxUint32),
    gen.UInt32Range(0, math.MaxUint32),
  ))

  properties.Property("Multiply should never fail.", prop.ForAll(
    func(a uint32, b uint32) bool {
      inpCompute := Compute{A: a, B: b}
      inpCompute.CoerceInt()
```

```
      inpCompute.Multiply()
      return true
    },
    gen.UInt32Range(0, math.MaxUint32),
    gen.UInt32Range(0, math.MaxUint32),
  ))

  properties.Property("Divide should never fail.", prop.ForAll(
    func(a uint32, b uint32) bool {
      inpCompute := Compute{A: a, B: b}
      inpCompute.CoerceInt()
      inpCompute.Divide()
      return true
    },
    gen.UInt32Range(0, math.MaxUint32),
    gen.UInt32Range(0, math.MaxUint32),
  ))

  properties.TestingRun(t)
}
```

Figure D.7: The `gopter` *property testing harness, taking advantage of the standard library's* `testing` *package.*

Examining the `Divide` property test (Figure D.8), we can see how an individual property is defined with `gopter`. To start, we observe that the Property function accepts a string parameter first, which is used to describe the property in spoken terms. The second argument is a wrapper for a function definition and respective input generators, allowing us to specify the input constraints for our test. Within the function definition, the logic for the property's test can be defined. The return value of this function is a boolean where `true` represents that the property held with the given inputs, and `false` represents a property violation.

In the properties we have defined, we simply want to ensure that with any given input, the appropriate operation on `A` and `B` will succeed. Therefore, we simply always return `true` since if it fails, a runtime exception will be raised and appropriately handled.

```
  properties.Property("Divide should never fail.", prop.ForAll(
    func(a uint32, b uint32) bool {
      inpCompute := Compute{A: a, B: b}
      inpCompute.CoerceInt()
      inpCompute.Divide()
      return true
    },
    gen.UInt32Range(0, math.MaxUint32),
    gen.UInt32Range(0, math.MaxUint32),
  ))
```

Figure D.8: The `Divide` *property test definition.*

To run the property tests, the standard `go test` command can be used, as shown in Figure D.9. The `leanovate/gopter` package must be installed for these commands to execute successfully.

```
user@host:~/Desktop/gopter_math$ go test
+ Add should never fail.: OK, passed 100 tests.
Elapsed time: 253.291µs
+ Subtract should never fail.: OK, passed 100 tests.
Elapsed time: 203.55µs
+ Multiply should never fail.: OK, passed 100 tests.
Elapsed time: 203.464µs
! Divide should never fail.: Error on property evaluation after 1 passed
  tests: Check paniced: runtime error: integer divide by zero
goroutine 5 [running]:
runtime/debug.Stack(0x5583a0, 0xc0000ccd80, 0xc00009d580)
        /usr/lib/go-1.12/src/runtime/debug/stack.go:24 +0x9d
github.com/leanovate/gopter/prop.checkConditionFunc.func2.1(0xc00009d9c0)
        /home/user/go/src/github.com/leanovate/gopter/prop/check_condition_func.g
 o:43 +0xeb
panic(0x554480, 0x6aa440)
        /usr/lib/go-1.12/src/runtime/panic.go:522 +0x1b5
_/home/user/Desktop/gopter_math_test.Compute.Divide(...)
        /home/user/Desktop/gopter_math/main_test.go:18
_/home/user/Desktop/gopter_math_test.TestCompute.func4(0x0, 0x0)
        /home/user/Desktop/gopter_math/main_test.go:63 +0x3d
# <snip for brevity>

ARG_0: 0
ARG_0_ORIGINAL (1 shrinks): 117380812
ARG_1: 0
ARG_1_ORIGINAL (1 shrinks): 3287875120
Elapsed time: 183.113µs
--- FAIL: TestCompute (0.00s)
   properties.go:57: failed with initial seed: 1568637945819043624
FAIL
exit status 1
FAIL    _/home/user/Desktop/gopter_math    0.004s
```

Figure D.9: Executing the test harness and observing the output of the property tests, where `Divide` *fails.*

In Figure D.9 we can observe that the defined `Divide` property was violated. A division by 0 was attempted, resulting in a runtime exception. The inputs leading to this crash are automatically recovered and shrunk to their minimal viable values, as seen in `ARG_0` and `ARG_1`. The original values causing the exception to be raised can be seen in `ARG_0_ORIGINAL` and `ARG_1_ORIGINAL` respectively.

This example displays only a small amount of `gopter`'s functionality. For testing more complex state machine transitions, the [`gopter/commands`](gopter/commands) package should be reviewed.