PRESENTS

Fuzzing integration of etcd

In collaboration with the etcd project maintainers and The Linux Foundation

Authors

Adam Korczynski <adam@adalogics.com>
David Korczynski <david@adalogics.com>
Date: 11th March, 2022

Executive summary

This report details the engagement whereby Ada Logics integrated continuous fuzzing into the etcd project. At the beginning of this engagement, etcd was not being fuzzed continuously, but had previously had limited fuzzing done as part of a security audit. The focus of the engagement with Ada Logics was solely on building a solid fuzzing infrastructure that would hit critical parts of the code and run continuously.

Etcd is a core component of Kubernetes, and the benefits of etcd having fuzzers running continuously benefits all Kubernetes users. Maintaining security and stability in the etcd project is a critical task which fuzzing can help with.

Results summarised

18 fuzzers were developed.

OSS-Fuzz integration for continuous fuzzing set up.

8 bugs were found.

- 2 nil-dereferences
- 2 slice/index out of range
- 2 panics from invalid utf-8
- 2 invalid type assertions

All bugs were fixed during the engagement.

All fuzzers are merged into the CNCF-fuzzing repository.

Engagement process and methodology

All work was done against the latest etcd code from Github and the first step was to create an extended set of fuzzers. Once the fuzzers had been written, the next step was to set these up to run continuously by way of OSS-Fuzz. OSS-Fuzz is a free open source service that manages the execution of fuzzers, triaging of bugs, conveniently alerts when issues are found and much more, and the service is available to critical open source projects. The OSS-Fuzz integration is set up such that all fuzzers running continuously will test against the latest etcd main branch. All fuzzers are implemented by way of the go-fuzz engine which was the go engine supported by OSS-Fuzz at the time of the engagement.

During the engagement we set up each fuzzer to run continuously as soon as the given fuzzer was ready, which means the number of fuzzers running on OSS-Fuzz would increase steadily as the engagement progressed. This allowed us to use the feedback provided by OSS-Fuzz to help steer the engagement.

The main focus in this engagement was to test for code errors. The types of errors to find include out of bounds, out of range, nil-pointer dereference, faulty type assertion, out of memory, off-by-one, infinite loop, timeout and divide by zero.

Overview of fuzzers

In this section we will briefly iterate through the fuzzers that were developed and set up to run continuously. All fuzzers are uploaded to the [cncf-fuzzing repository](). Some of them are being built by OSS-Fuzz from the `go.etcd.io/etcd/tests/v3/fuzzing`. This is a directory that is created by OSS-Fuzz. The OSS-Fuzz build file can be found here: https://github.com/cncf/cncf-fuzzing/blob/main/projects/etcd/build.sh.

#	Fuzzer Name	Package	Uploaded to
1	FuzzAPIMarshal	go.etcd.io/etcd/tests/v3/fuzzing	CNCF-fuzzing
2	FuzzWalCreate	go.etcd.io/etcd/server/v3/storage/wal	CNCF-fuzzing
3	FuzzMinimalEtcdVersion	go.etcd.io/etcd/server/v3/storage/wal	CNCF-fuzzing
4	FuzzKVProxy	go.etcd.io/etcd/tests/v3/fuzzing	CNCF-fuzzing
5	FuzzGRPCApis	go.etcd.io/etcd/tests/v3/fuzzing	CNCF-fuzzing
6	FuzzSnapLoad	go.etcd.io/etcd/server/v3/etcdserver/api/snap	CNCF-fuzzing
7	FuzzMvccStorage	go.etcd.io/etcd/server/v3/storage/mvcc	CNCF-fuzzing
8	FuzzMvccIndex	go.etcd.io/etcd/server/v3/storage/mvcc	CNCF-fuzzing
9	FuzzProxyServer	go.etcd.io/etcd/pkg/v3/proxy	CNCF-fuzzing
10	Fuzzapply	go.etcd.io/etcd/server/v3/etcdserver	CNCF-fuzzing
11	FuzzapplierV3backendApply	go.etcd.io/etcd/server/v3/etcdserver	CNCF-fuzzing
12	FuzzV3Server	go.etcd.io/etcd/server/v3/etcdserver	CNCF-fuzzing
13	FuzzAuthStore	go.etcd.io/etcd/server/v3/auth	CNCF-fuzzing
14	FuzzBackend	go.etcd.io/etcd/server/v3/storage/backend/testing	CNCF-fuzzing
15	FuzzRaftHttpRequests	go.etcd.io/etcd/server/v3/etcdserver/api/rafthttp	CNCF-fuzzing
16	FuzzMessageEncodeDecode	go.etcd.io/etcd/server/v3/etcdserver/api/rafthttp	CNCF-fuzzing
17	FuzzStep	go.etcd.io/etcd/raft/v3	CNCF-fuzzing
18	FuzzPurgeFile	go.etcd.io/etcd/client/pkg/v3/fileutil	CNCF-fuzzing

Rundown of fuzzers

FuzzAPIMarshal
Tests all the protobuf marshalling routines. It autogenerates a harness for each protobuf and creates a fuzzer that calls into the harnesses. An example of a harness is:

```
func FuzzetcdserverpbDeleteRangeResponse(data []byte) error {
        f := fuzz.NewConsumer(data)
        s := &etcdserverpb.DeleteRangeResponse{}
        err := f.GenerateStruct(s)
        if err != nil {
                return err
        }
        b, err := s.Marshal()
        if err != nil {
                return err
        }
        s2 := &etcdserverpb.DeleteRangeResponse{}
        err = s2.Unmarshal(b)
        if err != nil {
                return err
        }
        newBytes, err := f.GetBytes()
        if err != nil {
                return err
        }
        s3 := &etcdserverpb.DeleteRangeResponse{}
        err = s3.Unmarshal(newBytes)
        return err
}
```

The code of the fuzzer is auto generated at build time because the resulting fuzzer is enormous and contains a lot of repetitive code.

FuzzWalCreate
Creates a wal and saves it. The wal is then loaded and the metadata is compared between the created and loaded wal to ensure they are equal.

FuzzMinimalEtcdVersion
Creates a slice of go.etcd.io/etcd/raft/v3/raftpb.Entry and passes it to go.etcd.io/etcd/server/v3/storage/wal.MinimalEtcdVersion().

FuzzKVProxy
Sets up a KVProxyServer and makes pseudo-random requests against it.

FuzzGRPCApis

Sets up a GrpcAPI and makes pseudo-random requests against it.

FuzzSnapLoad
Creates a snapshotter from a directory containing pseudo-random files. The fuzzer then loads the snapshotter.

FuzzMvccStorage
Creates an MVCC storage and calls `Put()`, `Range()`, `DeleteRange()` with pseudo-random data against it.

FuzzMvccIndex
Creates a treeindex and calls `Put()`, Get(), Range(), Equal() with pseudo-random data against it.

FuzzProxyServer
Sets up a proxy server and sends pseudo-random data to it.

Fuzzapply
Sets up an EtcdServer and calls apply against it with a slice of pseudo-random `go.etcd.io/etcd/raft/v3/raftpb.raftpb.Entry`.

FuzzapplierV3backendApply
Sets up an `applierV3backend` and makes pseudo-random requests to it.

FuzzV3Server
Sets up an `EtcdServer` and makes pseudo-random requests to it to test the APIs in https://github.com/etcd-io/etcd/blob/main/server/etcdserver/v3_server.go

FuzzAuthStore
Sets up an `authStore` and makes pseudo-random requests to it.

FuzzBackend
Sets up a storage `Backend` and makes pseudo-random `UnsafePut` and `UnsafeRange` requests to it.

FuzzRaftHttpRequests
Sets up a pipelineHandler and makes pseudo-random requests to it.

FuzzMessageEncodeDecode
Encodes and subsequently decodes a `go.etcd.io/etcd/raft/v3/raftpb.Message`.

FuzzStep
Sets up a raft and calls its `Step()` method with a pseudo-random `go.etcd.io/etcd/raft/v3/raftpb.Message`.

FuzzPurgeFile

Tests `go.etcd.io/etcd/client/pkg/v3/fileutil.purgeFile` with a directory containing pseudo-random files.

Findings

In this section we iterate through the bugs found and present root-cause analysis.

#	Type	ID	Fixed
1	Nil-dereference	ADA-etcd-22-01	Yes
2	Slice bounds out of range	ADA-etcd-22-02	Yes
3	Panic from invalid utf-8	ADA-etcd-22-03	Yes
4	Index out of range	ADA-etcd-22-04	Yes
5	Invalid type assertion	ADA-etcd-22-05	Yes
6	Invalid type assertion	ADA-etcd-22-06	Yes
7	Nil-dereference	ADA-etcd-22-07	Yes
8	Panic from invalid utf-8	ADA-etcd-22-08	Yes

Issue 1

Type	Nil-dereference
Source	go.etcd.io/etcd/server/v3/etcdserver.(a *applierV3backend) Range()
Issue link	https://bugs.chromium.org/p/oss-fuzz/issues/detail?id=41579
Fix	https://github.com/etcd-io/etcd/pull/13555
ID	ADA-etcd-22-01

Etcd uses `sort.Sort()` in the `(*applierV3backend).Range()` API. If the parameter to `sort.Sort()` is `nil`, a nil-dereference would be triggered.

```
if sortOrder != pb.RangeRequest_NONE {
	var sorter sort.Interface
	switch {
	case r.SortTarget == pb.RangeRequest_KEY:
		sorter = &kvSortByKey{&kvSort{rr.KVs}}
	case r.SortTarget == pb.RangeRequest_VERSION:
		sorter = &kvSortByVersion{&kvSort{rr.KVs}}
	case r.SortTarget == pb.RangeRequest_CREATE:
		sorter = &kvSortByCreate{&kvSort{rr.KVs}}
	case r.SortTarget == pb.RangeRequest_MOD:
		sorter = &kvSortByMod{&kvSort{rr.KVs}}
	case r.SortTarget == pb.RangeRequest_VALUE:
		sorter = &kvSortByValue{&kvSort{rr.KVs}}
	}
	switch {
	case sortOrder == pb.RangeRequest_ASCEND:
		sort.Sort(sorter)
	case sortOrder == pb.RangeRequest_DESCEND:
		sort.Sort(sort.Reverse(sorter))
	}
}
```

The crash was fixed by defaulting to logging at panic-level in case sorter is invalid:

```
if sortOrder != pb.RangeRequest_NONE {
	var sorter sort.Interface
	switch {
	case r.SortTarget == pb.RangeRequest_KEY:
```

```
            sorter = &kvSortByKey{&kvSort{rr.KVs}}
      case r.SortTarget == pb.RangeRequest_VERSION:
            sorter = &kvSortByVersion{&kvSort{rr.KVs}}
      case r.SortTarget == pb.RangeRequest_CREATE:
            sorter = &kvSortByCreate{&kvSort{rr.KVs}}
      case r.SortTarget == pb.RangeRequest_MOD:
            sorter = &kvSortByMod{&kvSort{rr.KVs}}
      case r.SortTarget == pb.RangeRequest_VALUE:
            sorter = &kvSortByValue{&kvSort{rr.KVs}}
      default:
            lg.Panic("unexpected sort target", zap.Int32("sort-target",
int32(r.SortTarget)))
      }
      switch {
      case sortOrder == pb.RangeRequest_ASCEND:
            sort.Sort(sorter)
      case sortOrder == pb.RangeRequest_DESCEND:
            sort.Sort(sort.Reverse(sorter))
      }
}
```

Issue 2

Type	Slice bounds out of range
Source	go.etcd.io/etcd/raft/v3.(*raftLog).maybeAppend()
Issue link	https://bugs.chromium.org/p/oss-fuzz/issues/detail?id=42570
Fix	https://github.com/etcd-io/etcd/pull/13603
ID	ADA-etcd-22-02

An out of range panic was found in `maybeAppend()`:

```
func (l *raftLog) maybeAppend(index, logTerm, committed uint64, ents
...pb.Entry) (lastnewi uint64, ok bool) {
	if l.matchTerm(index, logTerm) {
		lastnewi = index + uint64(len(ents))
		ci := l.findConflict(ents)
		switch {
		case ci == 0:
		case ci <= l.committed:
			l.logger.Panicf("entry %d conflict with committed
entry [committed(%d)]", ci, l.committed)
		default:
			offset := index + 1
			l.append(ents[ci-offset:]...)
		}
		l.commitTo(min(committed, lastnewi))
		return lastnewi, true
	}
	return 0, false
}
```

The issue was resolved by logging at panic-level, if ci-offset is out of bounds:

```
func (l *raftLog) maybeAppend(index, logTerm, committed uint64, ents
...pb.Entry) (lastnewi uint64, ok bool) {
	if l.matchTerm(index, logTerm) {
		lastnewi = index + uint64(len(ents))
		ci := l.findConflict(ents)
		switch {
		case ci == 0:
		case ci <= l.committed:
			l.logger.Panicf("entry %d conflict with committed
entry [committed(%d)]", ci, l.committed)
```

```
            default:
                  offset := index + 1
                  if ci-offset > uint64(len(ents)) {
                        l.logger.Panicf("index, %d, is out of range
[%d]", ci-offset, len(ents))
                  }
                  l.append(ents[ci-offset:]...)
            }
            l.commitTo(min(committed, lastnewi))
            return lastnewi, true
      }
      return 0, false
}
```

Issue 3

Type	Panic from invalid utf-8
Source	`go.etcd.io/etcd/server/v3/etcdserver.(*EtcdServer).applyV2Request`
Issue link	https://bugs.chromium.org/p/oss-fuzz/issues/detail?id=42947
Fix	https://github.com/etcd-io/etcd/pull/13560
ID	ADA-etcd-22-03

A client-api-version containing invalid utf-8 could be passed to etcdserver which would cause a `panic` from the `github.com/prometheus/client_golang` dependency.

The issue happens in `go.etcd.io/etcd/server/v3/etcdserver/api/v3rpc.newUnaryInterceptor(*etcdserver.EtcdServer)` in the invocation on the marked line:

```
func newUnaryInterceptor(s *etcdserver.EtcdServer)
grpc.UnaryServerInterceptor {
      return func(ctx context.Context, req interface{}, info
*grpc.UnaryServerInfo, handler grpc.UnaryHandler) (interface{}, error) {
            if !api.IsCapabilityEnabled(api.V3rpcCapability) {
                  return nil, rpctypes.ErrGRPCNotCapable
            }

            if s.IsMemberExist(s.ID()) && s.IsLearner() &&
!isRPCSupportedForLearner(req) {
                  return nil, rpctypes.ErrGPRCNotSupportedForLearner
            }

            md, ok := metadata.FromIncomingContext(ctx)
            if ok {
                  ver, vs := "unknown",
md.Get(rpctypes.MetadataClientAPIVersionKey)
                  if len(vs) > 0 {
                        ver = vs[0]
                  }

                  clientRequests.WithLabelValues("unary", ver).Inc()

                  if ks := md[rpctypes.MetadataRequireLeaderKey];
len(ks) > 0 && ks[0] == rpctypes.MetadataHasLeader {
                        if s.Leader() == types.ID(raft.None) {
                              return nil, rpctypes.ErrGRPCNoLeader
```

```
                        }
                }
        }

        return handler(ctx, req)
    }
}
```

`clientRequests` is a `github.com/prometheus/client_golang/prometheus.CounterVec` that is initialized in https://github.com/etcd-io/etcd/blob/main/server/etcdserver/api/v3rpc/metrics.go, and its method `WithLabelValues` will throw a panic will throw a panic in case of an invalid utf-8 string:

```
func (v *CounterVec) WithLabelValues(lvs ...string) Counter {
    c, err := v.GetMetricWithLabelValues(lvs...)
    if err != nil {
        panic(err)
    }
    return c
}
```

The issue was fixed by checking the the string for invalid utf-8 before passing it onto the 3rd party dependency:

```
func newUnaryInterceptor(s *etcdserver.EtcdServer)
grpc.UnaryServerInterceptor {
    return func(ctx context.Context, req interface{}, info
*grpc.UnaryServerInfo, handler grpc.UnaryHandler) (interface{}, error) {
        if !api.IsCapabilityEnabled(api.V3rpcCapability) {
            return nil, rpctypes.ErrGRPCNotCapable
        }

        if s.IsMemberExist(s.ID()) && s.IsLearner() &&
!isRPCSupportedForLearner(req) {
            return nil, rpctypes.ErrGPRCNotSupportedForLearner
        }

        md, ok := metadata.FromIncomingContext(ctx)
        if ok {
            ver, vs := "unknown",
md.Get(rpctypes.MetadataClientAPIVersionKey)
            if len(vs) > 0 {
                ver = vs[0]
```

```
                }
                if !utf8.ValidString(ver) {
                        return nil,
rpctypes.ErrGRPCInvalidClientAPIVersion
                }
                clientRequests.WithLabelValues("unary", ver).Inc()

                if ks := md[rpctypes.MetadataRequireLeaderKey];
len(ks) > 0 && ks[0] == rpctypes.MetadataHasLeader {
                        if s.Leader() == types.ID(raft.None) {
                                return nil, rpctypes.ErrGRPCNoLeader
                        }
                }
        }

        return handler(ctx, req)
    }
}
```

Issue 4

Type	Index out of range
Source	`google.golang.org/protobuf/internal/filedesc.(*EnumValues).Get()`
Issue link	https://bugs.chromium.org/p/oss-fuzz/issues/detail?id=44433
Fix	https://github.com/etcd-io/etcd/pull/13689
ID	ADA-etcd-22-04

Etcd was using `google.golang.org/protobuf/internal/filedesc.(*EnumValues).Get()` without checking whether the passed parameter was in range. Since `google.golang.org/protobuf/internal/filedesc.(*EnumValues).Get()` does not do that check either, an out of range was triggered:

```
func visitEnumNumber(enum protoreflect.EnumDescriptor, number
protoreflect.EnumNumber, visitor Visitor) error {
	err := visitDescriptor(enum, visitor)
	if err != nil {
		return err
	}
	return visitEnumValue(fields.Get(intNumber), visitor)
}
```

The issue is fixed by checking if `number` is out of range and if `number` is negative:

```
func visitEnumNumber(enum protoreflect.EnumDescriptor, number
protoreflect.EnumNumber, visitor Visitor) error {
	err := visitDescriptor(enum, visitor)
	if err != nil {
		return err
	}
	intNumber := int(number)
	fields := enum.Values()
	if intNumber >= fields.Len() || intNumber < 0 {
		return fmt.Errorf("could not visit EnumNumber [%d]",
intNumber)
	}
	return visitEnumValue(fields.Get(intNumber), visitor)
}
```

Issue 5

Type	Invalid type assertion
Source	go.etcd.io/etcd/server/v3/storage/mvcc.(*treeIndex).Equal()
Issue link	https://bugs.chromium.org/p/oss-fuzz/issues/detail?id=44449
Fix	https://github.com/etcd-io/etcd/pull/13681
ID	ADA-etcd-22-05

A panic from an invalid type assertion was found in `go.etcd.io/etcd/server/v3/storage/mvcc.(*treeIndex).Equal()`. In this case, `b.tree.Get(item)` would be `nil` which caused the panic:

```
func (ti *treeIndex) Equal(bi index) bool {
	b := bi.(*treeIndex)

	if ti.tree.Len() != b.tree.Len() {
		return false
	}

	equal := true

	ti.tree.Ascend(func(item btree.Item) bool {
		aki := item.(*keyIndex)
		bki := b.tree.Get(item).(*keyIndex)
		if !aki.equal(bki) {
			equal = false
			return false
		}
		return true
	})

	return equal
}
```

The fix is to check the type assertion of `bki`, and the same is done for `aki` as a preventative measure:

```
func (ti *treeIndex) Equal(bi index) bool {
	b := bi.(*treeIndex)
```

```
	if ti.tree.Len() != b.tree.Len() {
		return false
	}

	equal := true

	ti.tree.Ascend(func(item btree.Item) bool {
		var aki, bki *keyIndex
		var ok bool
		if aki, ok = item.(*keyIndex); !ok {
			return false
		}
		if bki, ok = b.tree.Get(item).(*keyIndex); !ok {
			return false
		}
		if !aki.equal(bki) {
			equal = false
			return false
		}
		return true
	})

	return equal
}
```

Issue 6

Type	Invalid type assertion
Source	go.etcd.io/etcd/server/v3/auth.(*tokenSimple).assign()
Issue link	https://bugs.chromium.org/p/oss-fuzz/issues/detail?id=44478
Fix	https://github.com/etcd-io/etcd/pull/13682/files
ID	ADA-etcd-22-06

An unchecked type assertion would cause a panic in `go.etcd.io/etcd/server/v3/auth.(*tokenSimple).assign()` if the variable was nil:

```
func (t *tokenSimple) assign(ctx context.Context, username string, rev
uint64) (string, error) {
	// rev isn't used in simple token, it is only used in JWT
	index := ctx.Value(AuthenticateParamIndex{}).(uint64)
	simpleTokenPrefix :=
ctx.Value(AuthenticateParamSimpleTokenPrefix{}).(string)
	token := fmt.Sprintf("%s.%d", simpleTokenPrefix, index)
	t.assignSimpleTokenToUser(username, token)

	return token, nil
}
```

The panic is fixed by checking the type assertion:

```
func (t *tokenSimple) assign(ctx context.Context, username string, rev
uint64) (string, error) {
	// rev isn't used in simple token, it is only used in JWT
	var index uint64
	var ok bool
	if index, ok = ctx.Value(AuthenticateParamIndex{}).(uint64); !ok {
		return "", errors.New("failed to assign")
	}
	simpleTokenPrefix :=
ctx.Value(AuthenticateParamSimpleTokenPrefix{}).(string)
	token := fmt.Sprintf("%s.%d", simpleTokenPrefix, index)
	t.assignSimpleTokenToUser(username, token)

	return token, nil
}
```

Issue 7

Type	Nil-dereference
Source	go.etcd.io/etcd/server/v3/etcdserver.(*EtcdServer).applyEntryNormal()
Issue link	https://bugs.chromium.org/p/oss-fuzz/issues/detail?id=42181
Fix	https://github.com/etcd-io/etcd/pull/13695
ID	ADA-etcd-22-07

If a raft request had its Header field set to `nil`, `applyEntryNormal()` would panic with a nil-dereference:

```
func (s *EtcdServer) applyEntryNormal(e *raftpb.Entry) {

	...

	var raftReq pb.InternalRaftRequest
	if !pbutil.MaybeUnmarshal(&raftReq, e.Data) { // backward
compatible
		var r pb.Request
		rp := &r
		pbutil.MustUnmarshal(rp, e.Data)
		s.lg.Debug("applyEntryNormal", zap.Stringer("V2request",
rp))
		s.w.Trigger(r.ID, s.applyV2Request((*RequestV2)(rp),
shouldApplyV3))
		return
	}
	s.lg.Debug("applyEntryNormal", zap.Stringer("raftReq", &raftReq))

	if raftReq.V2 != nil {
		req := (*RequestV2)(raftReq.V2)
		s.w.Trigger(req.ID, s.applyV2Request(req, shouldApplyV3))
		return
	}

	id := raftReq.ID
	if id == 0 {
		id = raftReq.Header.ID
	}

	...
```

The issue is resolved by checking if `raftReq.Header` is `nil` in which case etcd would log at panic level:

```
func (s *EtcdServer) applyEntryNormal(e *raftpb.Entry) {

	...

	var raftReq pb.InternalRaftRequest
	if !pbutil.MaybeUnmarshal(&raftReq, e.Data) { // backward
compatible
		var r pb.Request
		rp := &r
		pbutil.MustUnmarshal(rp, e.Data)
		s.lg.Debug("applyEntryNormal", zap.Stringer("V2request",
rp))
		s.w.Trigger(r.ID, s.applyV2Request((*RequestV2)(rp),
shouldApplyV3))
		return
	}
	s.lg.Debug("applyEntryNormal", zap.Stringer("raftReq", &raftReq))

	if raftReq.V2 != nil {
		req := (*RequestV2)(raftReq.V2)
		s.w.Trigger(req.ID, s.applyV2Request(req, shouldApplyV3))
		return
	}

	id := raftReq.ID
	if id == 0 {
		if raftReq.Header == nil {
			s.lg.Panic("applyEntryNormal, could not find a
header")
		}
		id = raftReq.Header.ID
	}

	...
```

Issue 8

Type	Panic from invalid utf-8
Source	go.etcd.io/etcd/server/v3/etcdserver.(*EtcdServer).applyV2Request()
Issue link	https://oss-fuzz.com/testcase-detail/6750173361995776
Fix	https://github.com/etcd-io/etcd/pull/13700
ID	ADA-etcd-22-08

Similar to issue 3, another usage of `WithLabelValues()` in the `github.com/prometheus/client_golang` could crash etcd with string containing invalid utf8:

```
func (s *EtcdServer) applyV2Request(r *RequestV2, shouldApplyV3
membership.ShouldApplyV3) (resp Response) {
	stringer := panicAlternativeStringer{
		stringer:    r,
		alternative: func() string { return
fmt.Sprintf("id:%d,method:%s,path:%s", r.ID, r.Method, r.Path) },
	}
	defer func(start time.Time) {
		success := resp.Err == nil
		applySec.WithLabelValues(v2Version, r.Method,
strconv.FormatBool(success)).Observe(time.Since(start).Seconds())
		warnOfExpensiveRequest(s.Logger(),
s.Cfg.WarningApplyDuration, start, stringer, nil, nil)
	}(time.Now())
```

The panic is fixed by checking the string for invalid utf8:

```
func (s *EtcdServer) applyV2Request(r *RequestV2, shouldApplyV3
membership.ShouldApplyV3) (resp Response) {
	stringer := panicAlternativeStringer{
		stringer:    r,
		alternative: func() string { return
fmt.Sprintf("id:%d,method:%s,path:%s", r.ID, r.Method, r.Path) },
	}
	defer func(start time.Time) {
		if !utf8.ValidString(r.Method) {
			s.lg.Info("method is not valid utf-8")
			return
		}
		success := resp.Err == nil
		applySec.WithLabelValues(v2Version, r.Method,
```

```
strconv.FormatBool(success)).Observe(time.Since(start).Seconds())
            warnOfExpensiveRequest(s.Logger(),
s.Cfg.WarningApplyDuration, start, stringer, nil, nil)
      }(time.Now())
```

Advice following engagement

Short-term advice

1. Create a strategy for where the fuzzers should be maintained. They are now hosted at the cncf-fuzzing repository, however it is recommended for the etcd maintainers to move the fuzzers upstream.
2. Fuzzing will be natively supported in Go 1.18 and it may be worthwhile to rewrite the fuzzers to native Go fuzzers and place them in their respective directories similar to how unit tests are managed. OSS-Fuzz is able to handle native Go fuzzers as of a recent PR, so continuous fuzzing will remain supported.
3. Run the fuzzers in the CI with either CIFuzz or as native Go fuzzers when Go 1.18 is released.
4. Improve the procedures and expectations among the maintainers to respond to reports by OSS-Fuzz. During the engagement, Ada Logics experienced an impressive response to PRs made by Ada Logics team members to fix the found bugs. PRs were reviewed and subsequently merged in a quick manner, however looking forward, the etcd maintainers should integrate a response procedure.

Long-term advice

1. Assess which parts of the etcd ecosystem are missing coverage and write fuzzers to cover the missing parts. These fuzzers should run continuously on OSS-Fuzz, and if any bugs are found, they should be fixed in line with how the bugs in this engagement were fixed.
2. When new code is submitted to etcd that will not be covered by existing fuzzers, make it a routine to include fuzzers that cover this code.

Conclusions and future work

In this engagement we, Ada Logics, developed an extensive fuzzing suite for the etcd project. We integrated the fuzzing suite into the OSS-Fuzz fuzzing service such that all fuzzers are now running continuously by OSS-Fuzz indefinitely. A total of 18 fuzzers were developed and a total of 8 bugs were found. All of these bugs are now fixed.

We would like to acknowledge the etcd maintainers and reviewers, specially to Sahdev Zala (IBM), Marek Siarkowicz (Google), Piotr Tabor (Google) and Benjamin Wang (VMware) who all supported our work during the engagement.

This work was commissioned by the Cloud Native Computing Foundation (CNCF).