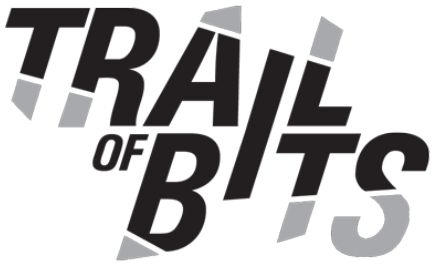

etcd

Security Assessment

February 7, 2020

Prepared For:
Xiang Li | *etcd*
xiangli.cs@gmail.com

Chris Aniszczyk | *Linux Foundation*
caniszczyk@linuxfoundation.org

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

Dominik Czarnota | *Trail of Bits*
dominik.czarnota@trailofbits.com

Sai Vegasena | *Trail of Bits*
sai.vegasena@trailofbits.com

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

Change Log:
April 14, 2020: Corrected typo in finding difficulties.
May 11, 2020: Corrected severity inconsistency.
July 22, 2020: Added Appendix D: Gateway Finding Remediations.
August 04, 2020: Updated executive summary to highlight Appendix D.

Executive Summary

From January 21 through January 31, 2020, the Linux Foundation engaged Trail of Bits to review the security of etcd. Trail of Bits conducted this assessment over the course of four person-weeks with four engineers working from release 3.4.3 of the etcd-io/etcd repository.

In the first week of the assessment, Trail of Bits set up local environments for building and testing the etcd system. During this time, we performed a mixture of manual and automated review. Automated review consisted of running various static analysis tools, such as `errcheck`, `ineffassign`, and `go-sec`. Results were subsequently reviewed and triaged as necessary. Manual review focused on gaining familiarity with the implementation details of etcd, such as configuration options, default settings, service discovery, RAFT consensus, and leader election.

During the second week, we continued our manual review with the same targets, plus coverage of the etcd proxy and gateway. We also began instrumenting custom tooling. On the automated side, `google/gofuzz` and `dvyukov/go-fuzz` testing harnesses were developed to test the WAL file implementation (see Appendix C. Fuzzing results).

Our assessment revealed a total of 17 findings ranging from high- to informational-severity. Overall, the etcd codebase represents a mature and heavily adopted product. However, there are many edge-cases not caught by the current test suite, and there are areas where the expected functionality of etcd does not match its implementation. These gaps can affect the security posture of the system since etcd gateway users may make inaccurate assumptions.

Examples: TOB-ETCD-001: Gateway TLS endpoint validation only confirms TCP reachability, or TOB-ETCD-004: Gateway TLS endpoint validation only applies to endpoints detected in DNS SRV records. Another example can be seen in the WAL implementation, where each WAL entry has semi-trusted metadata, resulting in a potential crash of quorum instances if a malicious leader propagates malicious entries. See further details in TOB-ETCD-006: An entry with large index causes panic in WAL.ReadAll method.

To improve the security posture of etcd, Trail of Bits recommends first addressing the findings in this report, prioritizing short-term recommendations, and integrating long-term recommendations into future releases. Once fixes are applied and recommendations addressed, a future assessment should be performed to ensure the fixes are adequate and do not introduce additional security risks.

After the assessment was completed, the Etcd team followed-up with the assessment team to review findings related to the gateway. Further details about how they were addressed were included in Appendix D.

Project Dashboard

Application Summary

Name	Etcd
Version	Release 3.4.3
Type	Datastore
Platforms	Go

Engagement Summary

Dates	January 11–21, 2020
Method	Whitebox
Consultants Engaged	2
Level of Effort	4 person-weeks

Vulnerability Summary

Total High-Severity Issues	1	■
Total Medium-Severity Issues	6	■■■■■■
Total Low-Severity Issues	6	■■■■■■
Total Informational-Severity Issues	4	■■■■
Total	17	

Category Breakdown

Data Validation	5	■■■■■
Access Controls	2	■■
Cryptography	4	■■■■
Logging	2	■■
Authentication	1	■
Data Exposure	1	■
Denial of Service	1	■
Configuration	1	■
Total	17	

Engagement Goals

The engagement was scoped to provide a security assessment of etcd as a whole.

Specifically, we sought to answer the following questions:

- Is there any way an attacker could impede service discovery?
- Is there any way a malicious etcd leader could impact its peers?
- Are the filesystem permissions used by etcd secure?
- Are there any major problems with the use of cryptography in etcd?
- Are there any correctness issues in error-handling within etcd?
- Does etcd provide the necessary logging for sensitive operations?
- Are there any major concerns with the use of the gateway and proxy functionality of etcd?

Coverage

Authentication and authorization. The methods used to authenticate and authorize clients with etcd were reviewed, including the TLS authentication assumptions of the ancillary proxies and gateways.

Filesystem permissions. We reviewed assumptions around file and directory permissions, focusing on enforcement and validation of permissions, including the impact of package implementation semantics on these validations.

Cryptography. We reviewed TLS authentication methods and assumptions, focusing on client-facing communications.

Default settings. The defaults used by etcd were reviewed for their impact on the system's security posture.

Data validation. We reviewed data validation throughout the system, focusing on areas where data from external sources is retrieved and subsequently parsed, such as service discovery and RAFT operations.

Error-handling. Error-handling was reviewed to identify areas where etcd could fail in an unexpected way, or otherwise operate in an unintended fashion. Additional focus was placed on packages used in critical locations which may be modified later, such as packages assisting in WAL file operations.

Logging. We reviewed logging, focusing on correctness of reporting and ability to log and implement alerting on security-critical events such as authentication.

Service discovery. Our review of service discovery operations focused on problems that could prevent peers from successfully completing discovery, or could otherwise impact the stability of discovered peers.

WAL operations. The package used to interact with WAL files was reviewed for problems that could lead to the destabilization of RAFT operations, or any of the peers attempting to form consensus.

Gateway and proxy. The gateway and proxy functionalities of etcd were reviewed to ensure operations such as TLS connection handling and load balancing were correct. We focused primarily on issues such as resource exhaustion, authentication, and TOCTOU.

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

When the trusted CA flag is provided to the etcd gateway, ensure dialing uses a TLS connection instead of a TCP connection. This will ensure all connections from gateway to endpoint are appropriately authenticated. TOB-ETCD-001

The gateway should filter endpoints to exclude those that resolve to the listening address of the gateway. This will help ensure circular references are more difficult to configure by users. TOB-ETCD-002

When using utilities such as `os.MkdirAll`, check all directories in the path and validate their owner and permissions before performing operations on those that already exist. This will help avoid situations in which sensitive information is written to a pre-existing attacker-controlled path. TOB-ETCD-003

Ensure TLS authentication is applied to endpoints provided in the `--endpoints` flag on the etcd gateway. This will ensure consistent functionality across both endpoint discovery methods. TOB-ETCD-004

Authenticate each endpoint using the provided CA certificate upon each connection established by the etcd gateway. This will help prevent any endpoint that's only been authenticated once from being trusted. TOB-ETCD-005

Ensure proper size checks on the final `e.Index-w.start.Index-1` in each WAL entry, and gracefully error if the size is too large for the given slice. This will avoid a hard crash if an invalid WAL entry is encountered. TOB-ETCD-006

Provide a size check on the `decodeRecord` method and gracefully exit if the frame size is larger than the maximum slice length. This will prevent crashing due to out-of-bounds indexing of the allocated array if the WAL entry has been mutated. TOB-ETCD-007

Consider improving the password requirements to a higher minimum character amount. This will make it harder for an attacker to brute-force authentication of an etcd user. TOB-ETCD-008

Ensure an appropriate error is logged when an authentication attempt has failed due to a client certificate authentication. Ideally, each authentication method should generate unique log entries to allow operators to differentiate interactions. TOB-ETCD-009

Ensure the `dbCompactionKeysCounter` variable properly represents the etcd operations that have occurred. This will help ensure users can properly track and alert on metrics. TOB-ETCD-010

Do not accept a negative value of automatic compaction retention. Data validation should ensure the provided values are within the expected range. TOB-ETCD-011

Do not store credentials in the WAL. Instead, consider having a centralized location for credentials that can be hardened appropriately, limiting the attack surface for sensitive information. TOB-ETCD-012

Initialize a decoder in the `wal.Create function`, so the WAL* may decode information in the `wal.ReadAll` function. This will prevent users from encountering an error due to an uninitialized decoder. TOB-ETCD-013

Ensure proper validation for all values retrieved and parsed from outside sources. This will help reduce the effect of a third-party system compromise on the integrity of etcd. TOB-ETCD-014

Disable weak ciphers and require a special flag to enable them. By default, only enable the modern ciphers as recommended by the Mozilla Security/Server side document. TOB-ETCD-015

Deprecate the insecure defaults and promote more secure usage of etcd. For example, list all insecure options at once during startup, describing the problems and suggesting fixes, and linking to a documentation page that would show how to set up etcd securely. TOB-ETCD-016

Review configurations that use InsecureSkipVerify. Use insecure TLS selectively and with caution. There are few purposes for which unverified certificates should be supported. TOB-ETCD-017

Long Term

Ensure all dialers are using the appropriate features for their expected protocols. Consider indexing the supported protocols—where they are used and how they are enforced—in a centralized fashion. TOB-ETCD-001

Ensure error values are appropriately reported instead of silently failing. Consider implementing a generic graceful exit routine. TOB-ETCD-002

Enumerate files and directories for their expected permissions overall, and build validation to ensure appropriate permissions are applied before creation and upon use. Ideally, this validation should be centrally defined and used throughout the application as a whole. TOB-ETCD-003

Ensure TLS connections are always used for gateway endpoints where the protocol schema is HTTPS. This will force a fast failure for an endpoint that cannot successfully authenticate with the gateway, and ensure the client only connects to an authenticated backend. TOB-ETCD-004

Consider implementing periodic endpoint validation to ensure all specified endpoints are not only reachable, but also authenticated. This will allow proactive detection of unhealthy endpoints, as well as those no longer able to successfully authenticate with the given configuration. TOB-ETCD-005

Restructure WALs to be limited by entry index and size instead of just size. This will help prevent mutated entries from inducing a panic when parsing the WAL. TOB-ETCD-006

Consider building a method to identify expected ranges for valid values into the format for the WAL. This will assist in additional validations of the WAL file, allowing the detection of corruption or manipulation. TOB-ETCD-007

Consider adopting a specific standard for password requirements, such as NIST SP 800-204, and enforce it across the codebase. TOB-ETCD-008

Validate whether all significant program paths are logged distinguishably and documented appropriately. This will help ensure proper logging and detections can be built on each authentication method. TOB-ETCD-009

Expand unit tests to include testing metrics endpoints to ensure changes to the metrics endpoints still produce the expected behavior. This will prevent future introduction of breaking changes and ensure the stability of loggable metrics. TOB-ETCD-010

Account for negative values provided as the auto compaction value. This should also be tested within the corresponding `TestAutoCompactionModeParse` test. TOB-ETCD-011

Check if file permissions used by etcd are within an acceptable range. If the permissions are too broad, etcd should exit with an error for further investigation. TOB-ETCD-012

Add validation before the decoder is accessed in WAL functions. If the decoder is not initialized, error out or initialize appropriately. TOB-ETCD-013

Consider consolidating validation routines into a specific set of helper libraries used across the codebase. Avoid using `strconv.Atoi` without validating the parsed value. TOB-ETCD-014

Determine the most popular ciphers used by etcd clients and consider removing weak ciphers from support. This will help prevent clients from accidentally configuring an insecure cipher when using etcd. TOB-ETCD-015

Make etcd secure by default, by requiring a minimally secure launch configuration. To ease development, testing, and debugging, consider providing a simpler, but less secure configuration under `--insecure`. TOB-ETCD-016

Avoid use of insecure TLS configurations altogether. Verify certificates in all scenarios by default. TOB-ETCD-017

Findings Summary

#	Title	Type	Severity
1	Gateway TLS endpoint validation only confirms TCP reachability	Cryptography	Medium
2	The gateway can include itself as an endpoint, resulting in resource exhaustion	Denial of Service	High
3	Directories created via os.MkdirAll are not checked for permissions	Access Controls	Medium
4	Gateway TLS authentication only applies to endpoints detected in DNS SRV records	Cryptography	Medium
5	TOCTOU of gateway endpoint authentication	Authentication	Low
6	An entry with large index causes panic in WAL.ReadAll method	Data Validation	Medium
7	A large slice causes panic in decodeRecord method	Data Validation	Medium
8	No minimum password length	Access Controls	Medium
9	Inaccurate logging of authentication attempts for users with CN-based auth only	Logging	Low
10	The “Total number of db keys compacted” metric is never changed	Logging	Informational
11	Auto compaction retention can be set to negative value causing a compaction loop or a crash	Data Validation	Low
12	User credentials are stored in WAL logs in plaintext	Data Exposure	Low

13	Null pointer exception when calling wal.Readall after wal.Create	Data Validation	Informational
14	Submitting a -1 for cluster node size results in an index out-of-bound panic during service discovery	Data Validation	Low
15	Insecure ciphers are enabled by default	Cryptography	Low
16	etcd is insecure by default	Configuration	Informational
17	Use of TLS InsecureSkipVerify	Cryptography	Informational

1. Gateway TLS endpoint validation only confirms TCP reachability

Severity: Medium
Type: Cryptography
Target: `pkg/transport/tls.go, etcdmain/util.go`

Difficulty: Low
Finding ID: TOB-ETCD-001

Description

Secure endpoint validation is performed by the `etcd gateway start` command when the `--discovery-srv` flag is enabled. However, as currently implemented, it only validates TCP reachability, effectively allowing connections to an endpoint that doesn't accept TLS connections through the HTTPS URL.

The `transport.ValidateSecureEndpoints` function (Figure TOB-ETCD-001.1) is used to perform validation of formatted HTTPS endpoints within the `etcdmain.discoverEndpoints` function (Figure TOB-ETCD-001.2) used by both `etcdmain.mustNewClient` and `etcdmain.startGateway`.

The validation works in two steps. First, if the endpoint does not start with HTTPS, the endpoint is not considered secure. Second, if the address is unreachable (with HTTPS stripped from the beginning), it is considered insecure. Because of these two steps, if you prepend HTTPS to any address listening for TCP (but not TLS) connections, validation will succeed. The provided certificate file and server name are ignored when performing the `Dial` operation because the `net.Dial` function does not initiate a TLS handshake—it only establishes a TCP connection to the endpoint.

The proof of concept in Figure TOB-ETCD-001.3 is an extracted example of how `transport.ValidateSecureEndpoints` is used within the `etcdmain.discoverEndpoints` function. The certificate authority (CA) certificate used in the proof of concept can also be seen in Figure TOB-ETCD-001.4. Figures TOB-ETCD-001.5–6 show a Python simple HTTP server (not using TLS) and the output of the proof of concept (PoC). As the PoC output shows, it successfully validated the HTTPS-formatted URL to the simple HTTP server, and to Google's HTTP endpoint.

```
// ValidateSecureEndpoints scans the given endpoints against tls info, returning only those
// endpoints that could be validated as secure.
func ValidateSecureEndpoints(tlsInfo TLSInfo, eps []string) ([]string, error) {
	t, err := NewTransport(tlsInfo, 5*time.Second)
	if err != nil {
		return nil, err
	}
	var errs []string
	var endpoints []string
	for _, ep := range eps {
		if !strings.HasPrefix(ep, "https://") {
			errs = append(errs, fmt.Sprintf("%q is insecure", ep))
			continue
```

```
                }
                conn, cerr := t.Dial("tcp", ep[len("https://"):])
                if cerr != nil {
                        errs = append(errs, fmt.Sprintf("%q failed to dial (%v)", ep, cerr))
                        continue
                }
                conn.Close()
                endpoints = append(endpoints, ep)
        }
        if len(errs) != 0 {
                err = fmt.Errorf("%s", strings.Join(errs, ","))
        }
        return endpoints, err
}
```

Figure TOB-ETCD-001.1: The `ValidateSecureEndpoints` *function definition (pkg/transport/tls.go#L23-L49).*

```
...
        // confirm TLS connections are good
        tlsInfo := transport.TLSInfo{
                TrustedCAFile: ca,
                ServerName:    dns,
        }


        if lg != nil {
                lg.Info(
                        "validating discovered SRV endpoints",
                        zap.String("srv-server", dns),
                        zap.Strings("endpoints", endpoints),
                )
        } else {
                plog.Infof("validating discovered endpoints %v", endpoints)
        }


        endpoints, err = transport.ValidateSecureEndpoints(tlsInfo, endpoints)
...
```

Figure TOB-ETCD-001.2: The snippet of `discoverEndpoints` *where* `ValidateSecureEndpoints` *is used (etcdmain/util.go#L51-L67).*

```
package main

import (
        "fmt"
        "go.etcd.io/etcd/pkg/transport"
)

func main() {
        // Amazon CA
        ca := "./ca.crt"
        dns := `amazon.com`

        res, err := transport.ValidateSecureEndpoints(
                transport.TLSInfo{
```

```
                        TrustedCAFile: ca,
                        ServerName:    dns,
                },
                []string{"http://0.0.0.0:8000", "https://0.0.0.0:8000",
                          "http://www.google.com:80", "https://www.google.com:80"})

        // Print the output
        fmt.Println(res, err)
}
```

Figure TOB-ETCD-001.3: A proof of concept showing the failure to validate an endpoint as using HTTPS.

```
-----BEGIN CERTIFICATE-----
MIIDQTCCAimgAwIBAgITBmyfz5m/jAo54vB4ikPmljZbyjANBgkqhkiG9w0BAQsF
ADA5MQswCQYDVQQGEwJVUzEPMA0GA1UEChMGQW1hem9uMRkwFwYDVQQDExBBbWF6
b24gUm9vdCBDQSAxMB4XDTE1MDUyNjAwMDAwMFoXDTM4MDExNzAwMDAwMFowOTEL
MAkGA1UEBhMCVVMxDzANBgNVBAoTBkFtYXpvbjEZMBcGA1UEAxMQQW1hem9uIFJv
b3QgQ0EgMTCCASIwDQYJKoZIhvcNAQEBBQADggEPADCCAQoCggEBALJ4gHHKeNXj
ca9HgFB0fW7Y14h29Jlo91ghYPl0hAEvrAIthtOgQ3pOsqTQNroBvo3bSMgHFzZM
9O6II8c+6zf1tRn4SWiw3te5djgdYZ6k/oI2peVKVuRF4fn9tBb6dNqcmzU5L/qw
IFAGbHrQgLKm+a/sRxmPUDgH3KKHOVj4utWp+UhnMJbulHheb4mjUcAwhmahRWa6
VOujw5H5SNz/0egwLX0tdHA114gk957EWW67c4cX8jJGKLhD+rcdqsq08p8kDi1L
93FcXmn/6pUCyziKrlA4b9v7LWIbxcceVOF34GfID5yHI9Y/QCB/IIDEgEw+OyQm
jgSubJrIqg0CAwEAAaNCMEAwDwYDVR0TAQH/BAUwAwEB/zAOBgNVHQ8BAf8EBAMC
AYYwHQYDVR0OBBYEFIQYzIU07LwMlJQuCFmcx7IQTgoIMA0GCSqGSIb3DQEBCwUA
A4IBAQCY8jdaQZChGsV2USggNiMOruYou6r4lK5IpDB/G/wkjUu0yKGX9rbxenDI
U5PMCCjjmCXPI6T53iHTfIUJrU6adTrCC2qJeHZERxhlbI1Bjjt/msv0tadQ1wUs
N+gDS63pYaACbvXy8MWy7Vu33PqUXHeeE6V/Uq2V8viTO96LXFvKWlJbYK8U90vv
o/ufQJVtMVT8QtPHRh8jrdkPSHCa2XV4cdFyQzR1bldZwgJcJmApzyMZFo6IQ6XU
5MsI+yMRQ+hDKXJioaldXgjUkK642M4UwtBV8ob2xJNDd2ZhwLnoQdeXeGADbkpy
rqXRfboQnoZsG4q5WTP468SQvvG5
-----END CERTIFICATE-----
```

Figure TOB-ETCD-001.4: An arbitrary Amazon CA certificate.

```
$ python3 -m http.server
Serving HTTP on 0.0.0.0 port 8000 (http://0.0.0.0:8000/) ...
```

Figure TOB-ETCD-001.3: A basic HTTP server.

```
$ go run main.go
[https://0.0.0.0:8000 https://www.google.com:80] "http://0.0.0.0:8000" is
insecure,"http://www.google.com:80" is insecure
```

Figure TOB-ETCD-001.5: The proof of concept output from Figure TOB-ETCD-001.1.

Exploit Scenario

An etcd endpoint is misconfigured without client TLS authentication behind a gateway. Because the gateway fails to properly perform TLS authentication, the server is added as a valid endpoint, and a client connects without TLS authentication. The gateway selects the client-TLS-disabled endpoint, and client data is transmitted in the clear.

Recommendation

Short term, when the trusted CA flag is provided to the etcd gateway, ensure dialing uses a TLS connection instead of a TCP connection. This will ensure all connections from gateway to endpoint are appropriately authenticated.

Long term, ensure all dialers are using the appropriate features for their expected protocols. Consider indexing the supported protocols—where they are used and how they are enforced—in a centralized fashion.

2. The gateway can include itself as an endpoint, resulting in resource exhaustion

Severity: High
Type: Denial of Service
Target: `etcdmain/gateway.go`

Difficulty: High
Finding ID: TOB-ETCD-002

Description
The etcd gateway is a simple TCP proxy to allow for basic service discovery and access. However, it is possible to include the gateway address as an endpoint. This results in a denial of service, since the endpoint can become stuck in a loop of requesting itself until there are no more available file descriptors to accept connections on the gateway.

```
./etcd gateway start --endpoints=http://127.0.0.1:23790
```

Figure TOB-ETCD-002.1: Starting the gateway with the endpoints flag including the gateway address and port.

```
{"level":"info","ts":1579890333.565956,"caller":"tcpproxy/userspace.go:90",
"msg":"ready to proxy client requests","endpoints":["127.0.0.1:23790"]}
accept tcp 127.0.0.1:23790: accept: too many open files
```

Figure TOB-ETCD-002.2: The error message when the etcd gateway crashes (recovered through a basic `fmt.Printf` *instruction at the point of error, due to error-logging not capturing the error).*

Exploit Scenario
An attacker compromises the DNS server used to return `SRV` records for an etcd gateway. The attacker configures the DNS server to return the address of each gateway, causing the gateways to connect to themselves and exhaust file descriptor resources.

Recommendation
Short term, the gateway should filter endpoints to exclude those that resolve to the listening address of the gateway. This will help ensure circular references are more difficult for users to configure.

Long term, ensure error values are appropriately reported instead of silently failing. Consider implementing a generic graceful exit routine.

3. Directories created via os.MkdirAll are not checked for permissions

Severity: Medium
Type: Access Controls
Target: Multiple locations

Difficulty: High
Finding ID: TOB-ETCD-003

Description
etcd creates certain directory paths with restricted access permissions (0700) by using the `os.MkdirAll`. This function does not perform any permission checks when a given directory path exists already. Then an attacker can create a directory used by etcd with broad permissions before etcd attempts the same, which allows the attacker to read sensitive data produced by etcd's operations.

This happens in the following places:

- In the `TouchDirAll` utility function, which also checks if the end directory is writable (Figure TOB-ETCD-003.1). However, if a user controls the directory, they might change its permission after `TouchDirAll` checks it. This function is used for actions such as snapshot and WAL directories creation.
- In the `SelfCert` function (Figure TOB-ETCD-003.2) used for `clientCerts` directory creation.
- In the `startProxy` function (Figure TOB-ETCD-003.3) for `proxy` directory creation.

```
func TouchDirAll(dir string) error {
        // If path is already a directory, MkdirAll does nothing
        // and returns nil.
        err := os.MkdirAll(dir, PrivateDirMode)
        if err != nil {
                // if mkdirAll("a/text") and "text" is not
                // a directory, this will return syscall.ENOTDIR
                return err
        }
        return IsDirWriteable(dir)
}
```

Figure TOB-ETCD-003.1: The `TouchDirAll` function definition (pkg/fileutil/fileutil.go#L48-L58).

```
func SelfCert(lg *zap.Logger, dirpath string, hosts []string, additionalUsages
...x509.ExtKeyUsage) (info TLSInfo, err error) {
        if err = os.MkdirAll(dirpath, 0700); err != nil {
                return
        }
        info.Logger = lg

        certPath := filepath.Join(dirpath, "cert.pem")
        keyPath := filepath.Join(dirpath, "key.pem")
         // (...) - rest of the code
```

Figure TOB-ETCD-003.2: The `SelfCert` function definition (pkg/transport/listener.go#L115-L123).

```
        cfg.ec.Dir = filepath.Join(cfg.ec.Dir, "proxy")
        err = os.MkdirAll(cfg.ec.Dir, fileutil.PrivateDirMode)
        if err != nil {
                return err
        }

        var peerURLs []string
        clusterfile := filepath.Join(cfg.ec.Dir, "cluster")

        b, err := ioutil.ReadFile(clusterfile)
```

Figure TOB-ETCD-003.3: The `startProxy` *function definition (etcdmain/etcd.go#L360-L369).*

Exploit Scenario
Eve has unprivileged access to Alice's server, where an etcd server will be deployed or updated to a new version that introduces new directories/paths. Eve knows that etcd will use certain directory paths, so they create them with `0777` permissions. This allows Eve to prevent etcd from running correctly or (in some cases) make etcd leak sensitive information.

Recommendation
Short term, when using utilities such as `os.MkdirAll`, check all directories in the path and validate their owner and permissions before performing operations on them. This will help avoid situations where sensitive information is written to a pre-existing attacker-controlled path.

Long term, enumerate files and directories for their expected permissions overall, and build validation to ensure appropriate permissions are applied before creation and upon use. Ideally, this validation should be centrally defined and used throughout the application as a whole.

4. Gateway TLS authentication only applies to endpoints detected in DNS SRV records

Severity: Medium
Type: Cryptography
Target: `etcdmain/gateway.go`

Difficulty: Low
Finding ID: TOB-ETCD-004

Description
When starting a gateway, TLS authentication will only be attempted on endpoints identified in DNS `SRV` records for a given domain, which occurs in the `discoverEndpoints` function. No authentication is performed against endpoints provided in the `--endpoints` flag.

```
	srvs := discoverEndpoints(lg, gatewayDNSCluster, gatewayCA, gatewayInsecureDiscovery,
gatewayDNSClusterServiceName)
	if len(srvs.Endpoints) == 0 {
		// no endpoints discovered, fall back to provided endpoints
		srvs.Endpoints = gatewayEndpoints
	}
```

Figure TOB-ETCD-004.1: A snippet of the `startGateway` *function definition (etcdmain/gateway.go#L102-L106).*

Exploit Scenario
The etcd gateway is configured with HTTPS-formatted endpoints. However, no authentication of these endpoints is performed using the certificate provided by the certificate authority; thus, TLS is not properly enforced.

Recommendation
Short term, ensure TLS authentication is applied to endpoints provided in the `--endpoints` flag on the etcd gateway. This will ensure consistent functionality across both endpoint discovery methods.

Long term, ensure TLS connections are always used for gateway endpoints where the protocol schema is HTTPS. This will force a fast failure for an endpoint that cannot successfully authenticate with the gateway, and ensure the client only connects to an authenticated backend.

5. TOCTOU of gateway endpoint authentication

Severity: Low
Type: Authentication
Target: `etcdmain/gateway.go`

Difficulty: Low
Finding ID: TOB-ETCD-005

Description
The gateway only authenticates endpoints detected from DNS SRV records as documented in TOB-ETCD-004: Gateway TLS authentication only applies to endpoints detected in DNS SRV records, and it only authenticates the detected endpoints once. Therefore, if an endpoint changes its authentication settings, the gateway will continue to assume the endpoint is still authenticated.

Exploit Scenario
An attacker compromises an etcd gateway endpoint and subsequently modifies the authentication settings. Because the gateway does not enforce authentication upon each attempted connection, a misconfigured client may connect to the unauthenticated endpoint and request attacker-controlled values.

Recommendation
Short term, authenticate each endpoint using the provided CA certificate upon each connection established by the etcd gateway. This will help prevent any endpoint that's only been authenticated once from being trusted.

Long term, consider implementing periodic endpoint validation to ensure all specified endpoints are not only reachable, but also authenticated. This will allow proactive detection of unhealthy endpoints, as well as those no longer able to successfully authenticate with the given configuration.

6. An entry with large index causes panic in WAL.ReadAll method

Severity: Medium
Type: Data Validation
Target: `wal/wal.go`

Difficulty: High
Finding ID: TOB-ETCD-006

Description
After a record's data is successfully unmarshalled, it is possible to have an entry index, `e.index`, greater then the number of entries in `ents`. Furthermore, `e.index` could be `math.MaxInt64` while `w.start.index` could be `0`. This could cause issues when WAL entries are being read during consensus as an arbitrary etcd consensus participant could go down from a runtime panic when reading the entry.

```
func (w *WAL) ReadAll() (metadata []byte, state raftpb.HardState, ents []raftpb.Entry, err
error) {
        w.mu.Lock()
        defer w.mu.Unlock()

        rec := &walpb.Record{}
        decoder := w.decoder

        var match bool
        for err = decoder.decode(rec); err == nil; err = decoder.decode(rec) {
                switch rec.Type {
                case entryType:
                        e := mustUnmarshalEntry(rec.Data)
                        if e.Index > w.start.Index {
                                ents = append(ents[:e.Index-w.start.Index-1], e)
                        }
                        w.enti = e.Index
```

Figure TOB-ETCD-006.1: A snippet of the `ReadAll` *function in the* `entryType` *case, where the* `ents` *slice is indexed with a large integer, leading to a panic (wal/wal.go#L423-L438).*

A WAL entry that caused the following runtime panic was produced. Notice how the `e.index` and `w.start.Index` satisfy the "greater than" check. However, when `13038096702221461993 - 0 - 1` evaluates to an upper bound of `13038096702221461992`, the slice fails on the premise of being too large.

```
2020-01-28 16:55:53.987278 I | wal: test10
2020-01-28 16:55:54.032191 I | wal: e.Index: 13038096702221461993
2020-01-28 16:55:54.032248 I | wal: w.start.Index: 0
panic: runtime error: slice bounds out of range [:13038096702221461992] with capacity 0

goroutine 1 [running]:

go.etcd.io/etcd/wal.(*WAL).ReadAll(0xc0000de2a0, 0xc0000ae688, 0x4, 0x8, 0x0, 0x0, 0x0, 0x0,
0x0, 0x0, ...)

/.../go.etcd.io/etcd/wal/wal.go:439 +0x1121
```

```
go.etcd.io/etcd/wal.FuzzRecover()

/.../go.etcd.i/etcd/wal/wal_fuzzer.go:592 +0x77e

go.etcd.io/etcd/wal.FuzzStep()

/.../go.etcd.io/etcd/wal/wal_fuzzer.go:1067 +0x48d

go.etcd.io/etcd/wal.FuzzWal()

/.../go.etcd.io/etcd/wal/wal_fuzzer.go:1090 +0x8b

main.main()

/.../go.etcd.io/etcd/fuzz.go:10 +0x20
```

Figure TOB-ETCD-006.2: Trace of the triggered panic.

Exploit Scenario
A RAFT participant receives a corrupted WAL from a compromised leader after an election. The participant subsequently `panic`s when the metadata is being parsed, which causes the participant to crash. A participant could also `panic` on reading RAFT state from a WAL with this error.

Recommendation
Short term, ensure proper size checks on the final `e.Index-w.start.Index-1` in each WAL entry, and gracefully error if the size is too large for the given slice. This will prevent a hard crash if an invalid WAL entry is encountered.

Long term, restructure WALs to be limited by entry index *and* size instead of size alone. This will help prevent mutated entries from inducing a panic when parsing the WAL.

7. A large slice causes panic in decodeRecord method

Severity: Medium
Type: Data Validation
Target: `wal/decoder.go`

Difficulty: High
Finding ID: TOB-ETCD-007

Description
The size of a record is stored in the length field of a WAL file, and no additional validation is done on this data. Therefore, it is possible to forge an extremely large frame size—so large, in fact, that the `data := make([]byte, recBytes+padBytes)` can unintentionally panic at the expense of any RAFT participant trying to decode the WAL.

```
func (d *decoder) decodeRecord(rec *walpb.Record) error {
	if len(d.brs) == 0 {
		return io.EOF
	}

	l, err := readInt64(d.brs[0])
	if err == io.EOF || (err == nil && l == 0) {
		// hit end of file or preallocated space
		d.brs = d.brs[1:]
		if len(d.brs) == 0 {
			return io.EOF
		}
		d.lastValidOff = 0
		return d.decodeRecord(rec)
	}
	if err != nil {
		return err
	}

	recBytes, padBytes := decodeFrameSize(l)

	data := make([]byte, recBytes+padBytes)
```

Figure TOB-ETCD-007.1: A snippet of the `decodeRecord` *method with vulnerable* `make` *call (wal/decoder.go#L62-L83).*

A WAL file that was generated during fuzzing produced the following panic. Notice the runtime error that was caused by attempting to create a `[]byte` slice the size of `recBytes + padBytes` after decoding the frame size from the length data.

```
panic: runtime error: makeslice: len out of range

goroutine 1 [running]:

go.etcd.io/etcd/wal.(*decoder).decodeRecord(0xc00006a040, 0xc000103d18,
0xc000103c30, 0x41a2a7)
/.../go.etcd.io/etcd/wal/decoder.go:83 +0x21e

go.etcd.io/etcd/wal.(*decoder).decode(0xc00006a040, 0xc000103d18, 0x0, 0x0)
```

```
/.../go.etcd.io/etcd/wal/decoder.go:59 +0xa7

go.etcd.io/etcd/wal.(*WAL).ReadAll(0xc000154000, 0x0, 0x0, 0x0, 0x0, 0x0,
0x0, 0x0, 0x0, 0x0, ...)
/.../go.etcd.io/etcd/wal/wal.go:431 +0x14b

go.etcd.io/etcd/wal.CovFuzz(0x0, 0x0, 0x0, 0x1)
/.../go.etcd.io/etcd/wal/wal_covfuzzer.go:31 +0x171
main.main()
        /.../go.etcd.io/etcd/fuzz.go:31 +0xd6
Process finished with exit code 2
```

Figure TOB-ETCD-007.2: Trace of the panic being triggered.

Exploit Scenario
The exact same exploit scenario from TOB-ETCD-006 can be applied to this vulnerability. However, in this case, any method trying to decode WAL data is vulnerable to this `panic`. The issue could arise from a malicious leader forging the size in their own WAL record.

Recommendation
Short term, provide a size check on the `decodeRecord` method and gracefully exit if the frame size is larger than the maximum slice length. This will prevent crashing due to out-of-bounds indexing of the allocated array if the WAL entry has been mutated.

Long term, consider building a method to identify expected ranges for valid values into the format for the WAL. This will assist additional validations of the WAL file, allowing the detection of corruption or manipulation.

8. No minimum password length

Severity: Medium
Type: Access Control
Target: `auth/store.go`, `UserAdd` function

Difficulty: Low
Finding ID: TOB-ETCD-008

Description
etcd does not perform any password length validation, which allows for very short passwords, such as those with a length of one. This may allow an attacker to guess or brute-force users' passwords with little computational effort.

Exploit Scenario
Alice sets her etcd user password to four characters. Eve, who can connect to the etcd cluster, brute-forces Alice's password and successfully authenticates.

Recommendation
Short term, consider improving the password requirements to a higher minimum character amount. This will make it harder for an attacker to brute-force authentication of an etcd user.

Long term, consider adopting a specific standard for password requirements, such as NIST SP 800-204, and enforce it across the codebase.

References
- NIST SP 800-204

9. Inaccurate logging of authentication attempts for users with CN-based auth only

Severity: Low
Type: Logging
Target: `etcdserver/v3_server.go`

Difficulty: Undetermined
Finding ID: TOB-ETCD-009

Description
etcd users who have no password can authenticate only through a client certificate. When such users try to authenticate into etcd using the `Authenticate` endpoint (Figure TOB-ETCD-009.2), the errors in Figure TOB-ETCD-009.1 are logged.

```
2020-02-01 17:35:04.368034 E | etcdserver: invalid authentication request to user root was
issued
2020-02-01 17:38:09.967097 N | auth: authentication failed, invalid password for user root
```

Figure TOB-ETCD-009.1: Errors logged when trying to authenticate as a user with no password.

These logs provide insufficient information regarding why the authentication failed, and may be misleading when auditing etcd logs.

```
func (s *EtcdServer) Authenticate(ctx context.Context, r *pb.AuthenticateRequest)
(*pb.AuthenticateResponse, error) {
    // (...)
    checkedRevision, err := s.AuthStore().CheckPassword(r.Name, r.Password)
    if err != nil {
        if err != auth.ErrAuthNotEnabled {
            if lg != nil {
                lg.Warn(
                    "invalid authentication was requested",
                    zap.String("user", r.Name),
                    zap.Error(err),
                )
            } else {
                plog.Errorf("invalid authentication request to user %s was issued", r.Name)
            }
```

Figure TOB-ETCD-009.2: The `Authenticate` endpoint function (etcdserver/v3_server.go#L410-L421).

A user with no password can be created with the `etcdctl user add <user> --no-password` command and the authentication request can be performed with the command in Figure TOB-ETCD-009.3.

```
curl -X POST http://127.0.0.1:2379/v3/auth/authenticate --data '{"name":
"<username>", "password": ""}'
```

Figure TOB-ETCD-009.3: a `curl` command of authentication without a password.

Note that this requires authentication to be enabled, which can be done with the `etcdctl auth enable` command.

Recommendation
Short term, ensure an appropriate error is logged when an authentication attempt has failed due to a client certificate authentication. Ideally, each authentication method should generate unique log entries to allow operators to differentiate interactions.

Long term, validate whether all significant program paths are logged distinguishably and documented appropriately. This will help ensure proper logging, and detections can be built on each authentication method.

10. The “Total number of db keys compacted” metric is never changed

Severity: Informational
Type: Logging
Target: `mvcc/kvstore_compaction.go, mvcc/metrics.go`

Difficulty: Undetermined
Finding ID: TOB-ETCD-010

Description
The `dbCompactionKeysCounter` (Figure TOB-ETCD-010.1) never changes. This counter is supposed to be increased by the `keyCompactions` value in the `scheduleCompaction` function (Figure TOB-ETCD-010.2) but the `keyCompactions` value never changes in that function. This can be misleading when debugging or auditing etcd.

```
dbCompactionKeysCounter = prometheus.NewCounter(
        prometheus.CounterOpts{
                Namespace: "etcd_debugging",
                Subsystem: "mvcc",
                Name:      "db_compaction_keys_total",
                Help:      "Total number of db keys compacted.",
        })
```

Figure TOB-ETCD-010.1: The `dbCompactionKeysCounter` *counter (mvcc/metrics.go#L170-L176).*

```
func (s *store) scheduleCompaction(compactMainRev int64, keep map[revision]struct{}) bool {
    totalStart := time.Now()
    // (...)
    keyCompactions := 0
    defer func() { dbCompactionKeysCounter.Add(float64(keyCompactions)) }()
```

Figure TOB-ETCD-010.2: The `scheduleCompaction` *function (mvcc/kvstore_compaction.go#L27-L28).*

Recommendation
Short term, ensure the `dbCompactionKeysCounter` variable properly represents the etcd operations that have occurred. This will help ensure the users can properly track and alert on metrics.

Long term, expand unit tests to include testing metrics endpoints to ensure changes to the metrics endpoints still produce the expected behavior. This will prevent future introduction of breaking changes and ensure the stability of loggable metrics.

11. Auto compaction retention can be set to negative value causing a compaction loop or a crash

Severity: Low
Type: Data Validation
Target: Automatic history compaction

Difficulty: Undetermined
Finding ID: TOB-ETCD-011

Description
The `parseCompactionRetention` function (Figure TOB-ETCD-011.1) parses a `retention` string with `strconv.Atoi` function and casts the result to `int64` to create a `time.Duration` instance. This scheme allows the value to be negative and causes the node to execute the history compaction in a loop, taking more CPU than usual and spamming logs.

```
func parseCompactionRetention(mode, retention string) (ret time.Duration, err error) {
	h, err := strconv.Atoi(retention)
	if err == nil {
		switch mode {
		case CompactorModeRevision:
			ret = time.Duration(int64(h))
		case CompactorModePeriodic:
			ret = time.Duration(int64(h)) * time.Hour
```

Figure TOB-ETCD-011.1: The `parseCompactionRetention` function (embed/etcd.go#L812-L819).

See the auto-compaction example in Figure TOB-ETCD-011.2.

```
$ ./bin/etcd --auto-compaction-retention=-1

# (...) - etcd initialisation logs

2020-02-02 00:23:57.320788 N | compactor: Starting auto-compaction at revision 1 (retention:
-1h0m0s)
2020-02-02 00:23:57.329580 N | compactor: Finished auto-compaction at revision 1
2020-02-02 00:23:57.329790 N | compactor: Starting auto-compaction at revision 1 (retention:
-1h0m0s)
2020-02-02 00:23:57.338759 N | compactor: Finished auto-compaction at revision 1
```

Figure TOB-ETCD-011.2: Launching etcd with the `--auto-compaction-retention=-1` flag makes its compaction thread execute the auto-compaction all the time.

Additionally, on MacOS/Darwin when a zap logger is set and a negative auto compaction retention value is passed, the etcd crashes as shown in Figure TOB-ETCD-011.3.

```
$ ./bin/etcd --auto-compaction-retention=-1 --logger=zap

# (...) - etcd initialisation logs

{"level":"info","ts":"2020-02-02T00:29:40.797+0100","caller":"v3compactor/periodic.go:135","
msg":"starting auto periodic compaction","revision":1,"compact-period":"-1h0m0s"}
panic: runtime error: invalid memory address or nil pointer dereference
```

```
[signal SIGSEGV: segmentation violation code=0x1 addr=0x20 pc=0x1825115]

goroutine 193 [running]:

go.etcd.io/etcd/etcdserver.(*EtcdServer).processInternalRaftRequestOnce(0xc0002bc580,
0x1e70d00, 0xc0001622c0, 0xc00000c040, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, ...)
/Users/dc/tob/projects/audit-etcd/src/etcd-3.4.3/etcdserver/v3_server.go:646 +0x1c5

go.etcd.io/etcd/etcdserver.(*EtcdServer).Compact(0xc0002bc580, 0x1e70d00, 0xc0001622c0,
0xc0000ee070, 0x0, 0x0, 0x0)
/Users/dc/tob/projects/audit-etcd/src/etcd-3.4.3/etcdserver/v3_server.go:208 +0xf1

go.etcd.io/etcd/etcdserver/api/v3compactor.(*Periodic).Run.func1(0xc00016e3f0, 0xb,
0xffffffac2e53f000, 0xfffffcb9cf476000)
/Users/dc/tob/projects/audit-etcd/src/etcd-3.4.3/etcdserver/api/v3compactor/periodic.go:143
+0x53c

created by go.etcd.io/etcd/etcdserver/api/v3compactor.(*Periodic).Run
/Users/dc/tob/projects/audit-etcd/src/etcd-3.4.3/etcdserver/api/v3compactor/periodic.go:103
+0xb0
```

Figure TOB-ETCD-011.3: Launching etcd with the `--auto-compaction-retention=-1 --logger=zap` *flags on MacOS makes it crash.*

Exploitation Scenario
An attacker who can control the auto compaction retention setting sets it to -1 so it keeps compacting etcd forever and saving it to logs, filling all disk space and rendering etcd unable to process properly.

Recommendation
Short term, do not accept a negative value of automatic compaction retention. Data validation should ensure the provided values are within the expected range.

Long term, account for negative values provided as the auto compaction value. This should also be tested within the corresponding `TestAutoCompactionModeParse` test.

12. User credentials are stored in WAL logs in plaintext

Severity: Low
Type: Data Exposure
Target: WAL

Difficulty: High
Finding ID: TOB-ETCD-012

Description
User credentials (login and password) are stored in WAL entries on each user authentication. By default, the WAL files have correct permissions, and are only read-write for the user who launched etcd, but if the path to the WAL log file has been created by an attacker, they could potentially read files containing the sensitive information stored by etcd. TOB-ETCD-003: Directories created via os.MkdirAll are not checked for permissions further details the semantics behind this problem.

There is also an issue for this bug in etcd-io/etcd#10132.

Exploitation Scenario
An attacker is able to create directories on a server where etcd hasn't been launched yet. They also create directories in the path where etcd will store its data. The attacker can then:

1. Create a fake etcd directory and file structure, and set file permissions so the etcd server will be able to write there:
```
mkdir -p default.etcd/member/wal
touch default.etcd/member/wal/0000000000000000-0000000000000000.wal
chmod -R 777 default.etcd
```
2. Wait until etcd server is launched for the first time, authorization is enabled, and some requests are made.
3. Read the `default.etcd/member/wal/0000000000000000-0000000000000000.wal` file which contains credentials.

This scenario exploits the TOB-ETCD-003 issue.

Recommendation
Short term, do not store credentials in the WAL. Instead, consider having a centralized location for credentials that can be hardened appropriately, limiting the attack surface for sensitive information.

Long term, check if permissions of the files used by etcd are within an acceptable range. If the permissions are too broad, etcd should exit with an error for further investigation.

13. Null pointer exception when calling wal.ReadAll after wal.Create

Severity: Informational
Type: Data Validation
Target: `wal/wal.go`

Difficulty: Undetermined
Finding ID: TOB-ETCD-013

Description
The `wal.Create` function creates a WAL object without setting a `decoder` field. Subsequently, when a `ReadAll` method is called on the resulting WAL object which uses the `decoder` field, a nil pointer dereference is triggered. Since the `wal.Create` function's docstring suggests that a `ReadAll` operation can be used after creating the object, this behavior is counter-intuitive and may introduce unexpected failures if such a flow is used in the future. Both functions appear in Figure TOB-ETCD-013.1.

```
// Create creates a WAL ready for appending records. The given metadata is
// recorded at the head of each WAL file, and can be retrieved with ReadAll.
func Create(lg *zap.Logger, dirpath string, metadata []byte) (*WAL, error) { /* (...) */ }

func (w *WAL) ReadAll() (metadata []byte, state raftpb.HardState, ents []raftpb.Entry, err
error) {
        w.mu.Lock()
        defer w.mu.Unlock()

        rec := &walpb.Record{}
        decoder := w.decoder

        var match bool
        for err = decoder.decode(rec); err == nil; err = decoder.decode(rec) {
```

Figure TOB-ETCD-013.1: The `Create` *(wal/wal.go#L95-L497) and* `ReadAll` *functions (wal/wal.go#L423-L431).*

When the following issue occurs, a nil dereference triggers the panic. This issue was found during the unit test fuzzing referenced in the Appendix C. Fuzzing results.

```
panic: runtime error: invalid memory address or nil pointer dereference
[signal SIGSEGV: segmentation violation code=0x1 addr=0x0 pc=0x83f129]

goroutine 1 [running]:
sync.(*Mutex).Lock(...)
        /usr/local/go/src/sync/mutex.go:74
go.etcd.io/etcd/wal.(*decoder).decode(0x0, 0xc0003b3d50, 0x0, 0x0)

/home/sai/github/ToB/assurance/audits/go.etcd.io/etcd/wal/decoder.go:57
+0x59
go.etcd.io/etcd/wal.(*WAL).ReadAll(0xc0000de2a0, 0x0, 0x0, 0x0, 0x0, 0x0,
0x0, 0x0, 0x0, 0x0, ...)
```

```
/home/sai/github/ToB/assurance/audits/go.etcd.io/etcd/wal/wal.go:431 +0x151
go.etcd.io/etcd/wal.GenCorpus()

/home/sai/github/ToB/assurance/audits/go.etcd.io/etcd/wal/wal_gen.go:62
+0x3fe
main.main()
        /home/sai/github/ToB/assurance/audits/go.etcd.io/etcd/fuzz.go:20
+0x20

Process finished with exit code 2
```

Figure TOB-ETCD-013.2: Nil dereference panic traceback.

Exploitation Scenario
A developer implements a new function or unit-test assuming the API supports the subsequent creation and reading of records. This causes unexpected failures.

Recommendation
Short term, initialize a decoder in the `wal.Create` function, so the `WAL` may decode information in the `wal.ReadAll` function. This will prevent users from encountering an error due to an uninitialized decoder.

Long term, check if the WAL has its decoder field set when it is going to be used, or refactor the API to have a single function that creates a fully initialized WAL object.

14. Submitting a -1 for cluster node size results in an index out-of-bound panic during service discovery

Severity: Low
Type: Data Validation
Target: Service discovery

Difficulty: Undetermined
Finding ID: TOB-ETCD-014

Description
When an etcd instance attempts to perform service discovery, if a cluster size of -1 is provided, the etcd instance will panic without recovery.

This `panic` occurs because the etcd instance retrieves a value from a remote key-value store (e.g. an existing etcd instance or cluster) where the value is stored as a string. To convert the string to an integer, `strconv.Atoi` is used (Figure TOB-ETCD-014.1), which returns an `int`. Because an `int` can be negative, a lack of validation results in this value indexing a slice. This in turn creates an index out-of-bounds panic (Figure TOB-ETCD-014.2) until other etcd instances are discoverable (Figure TOB-ETCD-014.3).

```
	size, err := strconv.Atoi(resp.Node.Value)
	if err != nil {
		return nil, 0, 0, ErrBadSizeKey
	}
```

Figure TOB-ETCD-014.1: The use of `strconv.Atoi` *without validating the parsed integer beyond whether an error occurred during conversion (etcdserver/api/v2discovery/discovery.go#L250-L253).*

```
panic: runtime error: slice bounds out of range [:-1]
goroutine 1 [running]:
go.etcd.io/etcd/etcdserver/api/v2discovery.(*discovery).waitNodes(0xc000294
6e0, 0xc000010210, 0x1, 0x1, 0xffffffffffffffff, 0xb, 0x0, 0x0, 0x0, 0x0,
...)
      /.../etcdserver/api/v2discovery/discovery.go:341 +0x1bf6
go.etcd.io/etcd/etcdserver/api/v2discovery.(*discovery).joinCluster(0xc0002
946e0, 0xc0002cd040, 0x1d, 0x0, 0x0, 0x0, 0x0)
      /.../etcdserver/api/v2discovery/discovery.go:180 +0x2f6
go.etcd.io/etcd/etcdserver/api/v2discovery.JoinCluster(0x0, 0x7ffeefbff5de,
0x54, 0x0, 0x0, 0x561d867318552b53, 0xc0002cd040, 0x1d, 0x0, 0x0, ...)
      /.../etcdserver/api/v2discovery/discovery.go:68 +0x16d
go.etcd.io/etcd/etcdserver.NewServer(0x20a4b58, 0x7, 0x7ffeefbff5de, 0x54,
0x0, 0x0, 0xc0001cad80, 0x1, 0x1, 0xc0001caa80, ...)
      /.../etcdserver/server.go:389 +0x4148
go.etcd.io/etcd/embed.StartEtcd(0xc0001a5080, 0xc0001a5600, 0x0, 0x0)
      /.../embed/etcd.go:211 +0x1175
go.etcd.io/etcd/etcdmain.startEtcd(0xc0001a5080, 0x0, 0x0, 0x0, 0x0)
      /.../etcdmain/etcd.go:302 +0x7a
```

```
go.etcd.io/etcd/etcdmain.startEtcdOrProxyV2()
      /.../etcdmain/etcd.go:160 +0x3411
go.etcd.io/etcd/etcdmain.Main()
      /.../etcdmain/main.go:46 +0x43b
main.main()
      /.../main.go:28 +0x20
```

Figure TOB-ETCD-014.2: The traceback for the out-of-bounds index `panic`.

```
func (d *discovery) waitNodes(nodes []*client.Node, size int, index uint64) ([]*client.Node,
error) {
       if len(nodes) > size {
              nodes = nodes[:size]
       }
```

Figure TOB-ETCD-014.3: The negative `size` *is used to index this array, resulting in an index out-of-bounds* `panic` (etcdserver/api/v2discovery/discovery.go#L339-L342).

Exploitation Scenario
An attacker identifies an etcd cluster and discovery URL used for service discovery, then substitutes the size value used to bootstrap with a `-1`. As a result, all instances attempting to discover other instances with that URL panic.

Recommendation
Short term, ensure proper validation for all values retrieved and parsed from outside sources. This will help reduce the effect of a third-party system compromise on the integrity of etcd.

Long term, consider consolidating validation routines into a specific set of helper libraries used across the codebase. Avoid using `strconv.Atoi` without validating the parsed value.

15. Insecure ciphers are allowed by default

Severity: Low
Type: Cryptography
Target: `pkg/tlsutil/cipher_suites.go`

Difficulty: Undetermined
Finding ID: TOB-ETCD-015

Description
The TLS ciphers list supported by etcd contains weak ciphers. This list, as the comment in Figure TOB-ETCD-015.1 notes, was taken from Go cipher suites which disables some of the ciphers by default (marked with the `suiteDefaultOff` flag) but those ciphers are not disabled by etcd. This allows for insecure connections to etcd by default, exposing its users to risk.

This issue has been reported to etcd as issue 10304.

```
// cipher suites implemented by Go
// https://github.com/golang/go/blob/dev.boringcrypto.go1.10/src/crypto/tls/cipher_suites.go
var cipherSuites = map[string]uint16{
    "TLS_RSA_WITH_RC4_128_SHA":                tls.TLS_RSA_WITH_RC4_128_SHA,
    "TLS_RSA_WITH_3DES_EDE_CBC_SHA":           tls.TLS_RSA_WITH_3DES_EDE_CBC_SHA,
    "TLS_RSA_WITH_AES_128_CBC_SHA":            tls.TLS_RSA_WITH_AES_128_CBC_SHA,
    "TLS_RSA_WITH_AES_256_CBC_SHA":            tls.TLS_RSA_WITH_AES_256_CBC_SHA,
    "TLS_RSA_WITH_AES_128_CBC_SHA256":         tls.TLS_RSA_WITH_AES_128_CBC_SHA256,
    "TLS_RSA_WITH_AES_128_GCM_SHA256":         tls.TLS_RSA_WITH_AES_128_GCM_SHA256,
    "TLS_RSA_WITH_AES_256_GCM_SHA384":         tls.TLS_RSA_WITH_AES_256_GCM_SHA384,
    "TLS_ECDHE_ECDSA_WITH_RC4_128_SHA":        tls.TLS_ECDHE_ECDSA_WITH_RC4_128_SHA,
    "TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA":    tls.TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA,
    "TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA":    tls.TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA,
    "TLS_ECDHE_RSA_WITH_RC4_128_SHA":          tls.TLS_ECDHE_RSA_WITH_RC4_128_SHA,
    "TLS_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA":     tls.TLS_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA,
    "TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA":      tls.TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA,
    "TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA":      tls.TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA,
    "TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256": tls.TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256,
    "TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256":   tls.TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256,
    "TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256":   tls.TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256,
    "TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256": tls.TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256,
    "TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384":   tls.TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384,
    "TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384": tls.TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384,
    "TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305":    tls.TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305,
    "TLS_ECDHE_ECDSA_WITH_CHACHA20_POLY1305":  tls.TLS_ECDHE_ECDSA_WITH_CHACHA20_POLY1305,
 }
```

Figure TOB-ETCD-015.1: Ciphers supported by etcd (pkg/tlsutil/cipher_suites.go#L19-L44).

Recommendation
Short term, disable weak ciphers and require a special flag to enable them. By default, only enable the modern ciphers as recommended by the Mozilla Security/Server side document.

Long term, determine the most popular ciphers used by etcd clients and consider removing weak ciphers from support. This will help prevent clients from accidentally configuring an insecure cipher when using etcd.

16. etcd is insecure by default

Severity: Informational
Type: Cryptography
Target: `etcd configuration`

Difficulty: Undetermined
Finding ID: TOB-ETCD-016

Description
When etcd is started without a specific configuration, it:

- Serves traffic from unencrypted endpoint (http://127.0.0.1:2379/),
- Uses simple tokens instead of cryptographically signed ones,
- Allows for unauthenticated client access, and
- Doesn't use TLS for peer-to-peer connections.

Although logs are produced when starting etcd under the first two conditions (Figure TOB-ETCD-016.1), an inexperienced user might gain a false notion of the security of etcd defaults and become prone to further configuring etcd insecurely.

```
auth: simple token is not cryptographically signed
embed: serving insecure client requests on 127.0.0.1:2379, this is strongly discouraged!
```

Figure TOB-ETCD-016.1: Warnings present when launching etcd with no configuration flags.

This issue has been reported to etcd as issue 9475.

Recommendation
Short term, deprecate the insecure defaults and promote more secure usage of etcd. For example, list all insecure options at once during startup: describe the problems, suggest fixes, and link to a documentation page that would show how to set up etcd securely.

Long term, make etcd secure by default, by requiring a minimally secure launch configuration. To ease development, testing, and debugging, consider providing a simpler but less secure configuration under `--insecure`.

17. Use of TLS InsecureSkipVerify

Severity: Informational
Type: Cryptography
Target: `TLS Configuration`

Difficulty: Undetermined
Finding ID: TOB-ETCD-017

Description
Transport Layer Security (TLS) appears in multiple locations throughout the etcd codebase, sometimes including `InsecureSkipVerify` to disable certificate checks. The lack of authentication in some configurations presents opportunities for Monkey-In-The-Middle interference.

```
// If the user wants to skip TLS verification then we should set
// the InsecureSkipVerify flag in tls configuration.
if scfg.insecureSkipVerify && cfg.TLS != nil {
  cfg.TLS.InsecureSkipVerify = true
}
```

Figure TOB-ETCD-017.1: Use of `InsecureSkipVerify` *(etcdctl/ctlv3/command/global.go#L234-L238).*

Exploitation Scenario
`TLSInsecureSkipVerify` is enabled, allowing an attacker to perform Monkey-In-The-Middle operations without the complications of TLS verification.

Recommendation
Short term, review configurations that use `InsecureSkipVerify`. Use insecure TLS selectively and with caution. There are few purposes for which unverified certificates should be supported.

Long term, avoid use of insecure TLS configurations altogether. Verify certificates in all scenarios by default.

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

This appendix details etcd issues that could be improved, but do not impact the project's security posture. These recommendations are proposed to prevent future errors from occurring, and to improve the quality of future code contributions.

etcdctl/ctlv2/command/exec_watch_command.go#L81-L94
The `after-index` configuration variable is fetched as an integer and is then casted to `uint64`. We recommend fetching it as `Uint64`.

```
index = c.Int("after-index")
// (...)
w := ki.Watcher(key, &client.WatcherOptions{AfterIndex: uint64(index), Recursive:
recursive})
```

clientv3/balancer/connectivity/connectivity.go#L71-L72
The `updateVal` in the loop shown below will have a value of `math.MaxUint64` in its first iteration due to the `2*uint64(0) - 1` calculation (which is `18446744073709551615`). While this is not a bug, we recommend changing the code so it won't overflow, making it less confusing for readers.

```
for idx, state := range []connectivity.State{oldState, newState} {
        updateVal := 2*uint64(idx) - 1 // -1 for oldState and +1 for new.
```

auth/simple_token.go#L223-L229
There are places in the codebase where a result of `strconv.Atoi` integer parsing is casted to `uint64`. One of the examples is `auth/simple_token.go#L218-L235` shown below. Those cases should use `strconv.ParseUint` instead to prevent accepting negative integers as valid inputs.

```
index, err := strconv.Atoi(splitted[1])
if err != nil {
        return false
}

select {
case <-t.indexWaiter(uint64(index)):
```

etcdserver/api/rafthttp/http.go#L221-L222
The RAFT's `snapshotHandler`'s `ServeHTTP` function sets a very large `decodeLimit` by using a `uint64(1 << 63)` expression. We recommend using either 1) a proper limit, or 2) the `math.MaxUint64` constant (if the purpose is to set no limit).

```
// let snapshots be very large since they can exceed 512MB for large installations
 m, err := dec.decodeLimit(uint64(1 << 63))
```

embed/config.go#L136-L140 and **embed/config.go#L594-L599**
The `cfg.TickMs` and `cfg.ElectionMs` are of unsigned integer types, but they are checked for being less than zero. The comparisons below should be changed to check only for their value being "equal to zero."

```
// TickMs is the number of milliseconds between heartbeat ticks.
// TODO: decouple tickMs and heartbeat tick (current heartbeat tick = 1).
// make ticks a cluster wide configuration.
TickMs      uint `json:"heartbeat-interval"`
ElectionMs uint `json:"election-timeout"`


// Below are lines 594-599
if cfg.TickMs <= 0 {
        return fmt.Errorf("--heartbeat-interval must be >0 (set to %dms)", cfg.TickMs)
 }
if cfg.ElectionMs <= 0 {
        return fmt.Errorf("--election-timeout must be >0 (set to %dms)", cfg.ElectionMs)
}
```

C. Fuzzing results

Trail of Bits performed fuzz testing of the etcd WAL package using both google/gofuzz and dvyukov/go-fuzz. Several potential issues were identified in the etcd WAL package, ranging from a decoder panic (TOB-ETCD-007) to a null pointer exception in a WAL struct (TOB-ETCD-013).

etcd wal package

To test the WAL package, a corpus of valid WAL files were randomly generated with a script that uses `google/gofuzz` (see Figure C.3). This corpus was then used with `dvyukov/go-fuzz` to perform instrumented coverage-guided fuzzing. The results show the presence of potential edge-cases when the WAL decoder utilizes the size information it parses (TOB-ETCD-007) and a null pointer exception (TOB-ETCD-013).

```
func CovFuzz(data []byte)  int{
  err := ioutil.WriteFile(testPath, data, 0777)
  defer os.Remove(testPath)
  if err != nil {
     plog.Errorf("could not write test file because: %v", err)
     return 0
  }

  var err error
  var w *WAL
  if w, err = Open(zap.NewExample(), kPathToCrash , walpb.Snapshot{}); err != nil {
     plog.Error(err)
     return 0
  }
  metadata, _, _, err := w.ReadAll()
  if err != nil {
     plog.Fatalf("could not read file meta data and collect entries")
     return 1
  }
  if !bytes.Equal(metadata, []byte(metadataStr)) {
     plog.Fatalf("metadata = %s, want %s", metadata, metadataStr)
     return 1
  }
  return 0
}
```

Figure C.1: Fuzzing test harness used with auto-generated corpus. The fuzzing was performed with a single worker because the harness writes to and deletes a file.

Furthermore, many unit tests were translated to fuzz tests with `google/gofuzz` to provide an adequate amount of API coverage. The result of translating unit-tests produced a runtime panic in the `wal.ReadAll` function, detailed in TOB-ETCD-006. During fuzzing, the complex structure of snapshots, records, hardstates, and entries were taken into account.

```
func FuzzWriteRecord() {
```

```
  b := &walpb.Record{}
  encd := walpb.Record{}
  buf := new(bytes.Buffer)
  e := newEncoder(buf, 0, 0)
  complexfuzzer.Fuzz(&encd)
  e.encode(&encd)
  e.flush()
  decoder := newDecoder(ioutil.NopCloser(buf))
  err := decoder.decode(b)
  if err != nil {
     plog.Errorf("err = %v, want nil", err)
  }
  if b.Type != encd.Type {
     plog.Errorf("type = %d, want %d", b.Type, encd.Type)
  }
  if !reflect.DeepEqual(b.Data, encd.Data) {
     plog.Errorf("data = %v, want %v", b.Data, encd.Data)
  }
}
```

Figure C.2: Example of unit-test converted to fuzz test. `TestWriteRecord` *is converted to* `FuzzWriteRecord`*.*

Fuzzing triggered panics in the `MustUnmarshal` function in both fuzzing approaches, but this was not reported as a finding because it is intended behavior.

```
func GenEntries(w *WAL, size int) {
  state := raftpb.HardState{}
  if size < 0 {
     size *= -1
  }
  plog.Info(size)
  entries := make([]raftpb.Entry, size)
  for i:=0; i<size; i++ {
     complexfuzzer.Fuzz(&entries[i])
     entries[i].Index = uint64(i)
  }
  state.Commit = uint64(size-1)
  if err := w.Save(state, entries); err != nil {
     plog.Fatal(err)
  }

}


func GenCorpus() {
  path, err := ioutil.TempDir(os.TempDir(), "corpus")
  if err != nil {
     plog.Fatal(err)
  }

  w, err := Create(zap.NewExample(), path, []byte(metadataStr))
  if err != nil {
     plog.Fatal(err)

  }
  defer w.Close()
```

```
  var size int
  for i:=0; i<numTests; i++{
     fuzzer.Fuzz(&size)
     size %= numTests
     GenEntries(w, size + 2)
     err = Verify(zap.NewExample(), path, walpb.Snapshot{})
     if err != nil {
        plog.Errorf("expected a nil error, got %v", err)
        continue
     }
     w.cut()
  }
}
```

Figure C.3: Script used to generate WAL corpus.

D. Gateway Finding Remediations

After discussions with the Etcd team, we determined that remediating several gateway findings required either documentation improvements or future deprecation. Here are the specific remediations we discussed with the Etcd team:

- TOB-ETCD-001: Gateway TLS endpoint validation only confirms TCP reachability

  It appears that the function `ValidateSecureEndpoints` is misleading in name and intent. We believe appropriate documentation of this functionality plus deprecation of this misleading functionality is an acceptable path forward. During our discussions, the Etcd team noted that if the community wants endpoint TLS validation functionality, it would not be added to the gateway, but instead to a separate utility to avoid backwards incompatibility and reduce confusion.

- TOB-ETCD-004: Gateway TLS authentication only applies to endpoints detected in DNS SRV records

  This finding is an extension of TOB-ETCD-001: Gateway TLS endpoint validation only confirms TCP reachability, as the validation only applies to endpoints identified in DNS SRV records, not the endpoints provided through command line flags. As a result, there is an inconsistent application of validation.

  We believe that because of the aforementioned misleading name and intent of the `ValidateSecureEndpoints` function, this lack of validation does not present significant concern, outside the lack of a TCP reachability check. Appropriate documentation of the selective application of `ValidateSecureEndpoints` should be sufficient to prevent users from believing the same validation is performed on values provided through command line flags.

- TOB-ETCD-005: TOCTOU of gateway endpoint authentication

  As mentioned in TOB-ETCD-004: Gateway TLS authentication only applies to endpoints detected in DNS SRV records, `ValidateSecureEndpoints` is only applied to endpoints discovered through DNS SRV records. However, this validation only applies on boot, and is not checked again before the gateway routes connections to an endpoint.

  After our discussions with the Etcd team, we believe that appropriate documentation of this functionality will adequately prevent users from being misled as to the endpoint validation semantics.