# CMP417

# Part 1 – Software Security

Ethan Hastie

1801853

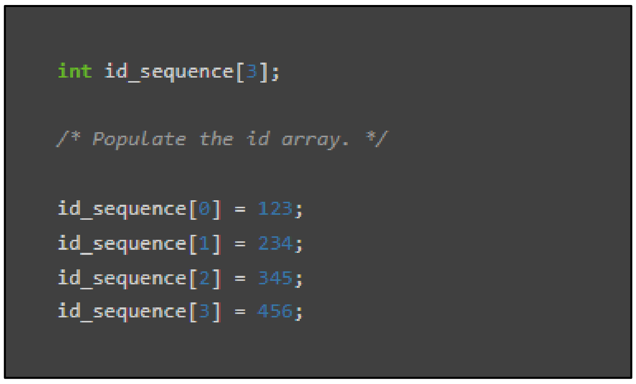
ScottishGlen

## Table of Contents

## Context

This report attempts to show that the PostgreSQL database utilised by ScottishGlen poses a significant risk to the security posture of the company. As a result, its current state would allow an entry into the company's system from the hacktivist group that had previously threatened the CEO of ScottishGlen, as well as other attackers that could circumvent its defences. Since the nature of the attack is unknown to defenders, the current plan should be to minimise the attack surface to prevent a possible attack.

An objective of this report is to demonstrate that memory corruption, otherwise referred to as CWE-787, is one of the most serious class of security faults that affect the database server hosted within ScottishGlen. Memory corruption, specifically an out-of-bounds write, occurs when software writes data past or before the boundaries of the intended buffer. The software that possesses the security fault may modify an index or perform pointer arithmetic that references a memory location outside the boundaries of the buffer. A write operation to this buffer can produce unexpected results, such as corruption of data, crashes, or code execution (CWE, 2021). An example of such a fault is creating an array with three elements and attempting to store a fourth element into the array (Martello Security, 2022).



```
int id_sequence[3];

/* Populate the id array. */

id_sequence[0] = 123;
id_sequence[1] = 234;
id_sequence[2] = 345;
id_sequence[3] = 456;
```

*Figure 1 Example of Out-of-Bounds Write*

In a report from the MITRE Corporation published in 2021, the number one weakness introduced in the creation of software in a list of the top 25 was an out-of-bounds write. This error was present in 3033 entries for NVD with a score of 65.93 (CWE, 2021), which demonstrates how serious this vulnerability is and how much of it is on the rise. Compared to 2021, in 2019 this security fault was ranked much lower with a rank of 12, with recent years showing an increase in this fault. Memory corruption may be introduced in multiple stages of the software development lifecycle. One of these is the requirements stage (CWE, 2021) where choice of language can affect how this weakness is introduced. Most notably, in languages such as C/C++ this fault can be introduced. This can also be introduced in the implementation phase where programmers can use unsafe functions to execute code. This had led to several well-known vulnerabilities, although one vulnerability caused by this fault will be discussed in this report, which is CVE-2019-10164.

This report begins by discussing the secure development practice that would fix the vulnerabilities described. It will then go onto demonstrate that the vulnerability would be fixed using the proposed secure practice. Finally, the report ends with a brief conclusion.

## Recommendation

To fix the vulnerability described by CVE-2019-10164, static code analysis tools are recommended in this report to be used on the source code. To be able to critically demonstrate why this technique can fix the vulnerability, an explanation of static analysis is required.

Static code analysis is demonstrated by carrying out tests on code within a non-runtime environment. This removes the requirement to test code and execute it without a debugger. During the early stages of development, developers can employ the use of static analysis tools before executing programs to ensure it does not violate standard and predefined rules. Coding issues such as standards, errors, and security weaknesses can be highlighted using static analysis before the program is executed. Multiple static analysis methods can be applied to the code, with techniques such as control analysis allowing for a structured control flow of the program, data flow analysis, which concerns that defined data is used properly, fault analysis analyses components to identify faults, and interface analysis involves authenticating simulations to ensure it is suitable for simulation (Codacy, 2020).

The benefits of static analysis are that it helps identify coding issues before running the program. In the case of ScottishGlen, the software developers can benefit from the added knowledge of standard coding practices reported by static tools and decrease their development process times in the future. Static tools can provide a greater insight into analysis of code as issues are hard to detect and are more effective than traditional manual code review and less time consuming, providing early feedback for development. Not only does static analysis detect direct bugs, but it helps find situations in the code that decrease code understanding (Issam, 2017). Despite the benefits of static analysis, a survey performed by Voke showed that only 15% of developers use these tools (Codacy, 2020). Many static analysis tools exist, commercial and open source, so there is nothing to lose, apart from costs if using commercial, for downloading and using these tools against the software. Once the tool is installed, the analysis can take on average 30 minutes to complete which can provide developers opportunities to complete other tasks. One of the main drawbacks of static analysis tools are that it generates many false positives, which can be time consuming to review by software developers and determine if they are faults. Developers will have to review results and fix mistakes starting from critical to menial (Codacy, 2020). It also has difficulty and a longer time taken to report out-of-bounds memory operations, although many modern tools take extra care to minimise the number of false positives it produces by using data flow analysis or constraint-based methods (CWE, 2021). Another technique may also be used to detect buffer overflow errors, which is referred to dynamic analysis.

Dynamic analysis is much more different compared to static analysis in that it runs code in a run-time environment using various inputs. An example of a dynamic analysis tool is a fuzzer, which generates large numbers of inputs and tests the program using this data to reveal defects within the program. An advantage of dynamic analysis is that it can reveal subtle errors that may not be discovered by static analysis, providing security assurance. Although, static analysis can uncover all possible execution paths and variable values, revealing errors that may not be discovered until after deployment (Intel, 2021). That is not to say that one method should be used over the other; the usage of both methods should be considered in development environments. Dynamic analysis can also cause the software to slow down. (CWE, 2021). Compared to static code analysis, dynamic analysis is conducted in the testing phase while static analysis is deployed during development, helping to inform developers of flaws before it gets tested. The inputs generated using dynamic analysis allow long input strings to be fuzzed into the program, providing an ability to detect buffer overflow vulnerabilities in the program's running state. This would provide a better visibility of the program while it is running, compared to static analysis. However, they can report several false positives and false negatives, same as static analysis, which can make exact pinpointing of faults difficult. It also may be difficult to find a trained dynamic analysis professional to test the application and using it on its own can give a false sense of security. In addition, employing the use of dynamic tools can make errors difficult to track in the code, meaning that it is more costly to fix coding issues (testbytes, 2017). Issues present in automated code analysis tools can be combined with manual reviews of software to form a defence in depth strategy.

As part of the software development process, manual code review can be conducted which analyses the code line by line to identify flaws. This can be a time-consuming process compared to using automated review tools, such as static analysis, and requires significant expertise to be performed properly. Although automated tools can identify flaws quicker, they only identify specific flaws and not ones that may be picked up otherwise manually. This can be mitigated using multiple automated tools, as seen later, but still this approach may not pick up every error (MITRE, 2021). In the case of ScottishGlen, their developers are competent with modifying the software and therefore this assessment can be conducted internally. This would save costs compared to relying on a third party for the code review. A form of manual review suitable for this case would be a structured code walkthrough. This is a form of static testing for software.

The developer would lead the review process and demonstrate the code they have created, with other team members present asking questions and spotting errors. This allows code to be peer reviewed identifying flaws in the code. Members should be kept relatively small to avoid non-productive reviews, roughly 2-6 members, with IEEE recommending main roles (IEEE, 2008):

- Author – creator of the product responsible for answering questions and acting as observer within the review.
- Presenter – responsible for the creation of the agenda and presenting the product, encouraging participation of all in the review. Presenter should be a member of the development team.
- Moderator – responsible for ensuring participation in the review and the walkthrough within the agenda.
- Reviewer/s – evaluating the product and its features, according to standards and any errors spotted. To make their task easier, can utilise a checklist built from previous software application weaknesses.
- Scribe – takes notes during the review, noting any issues spotted.

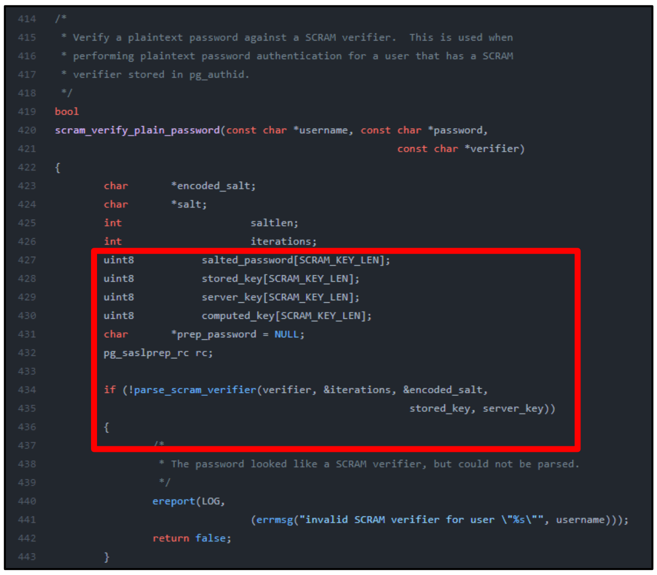
Errors can be found through early feedback and general concerns raised during the review are diverse due to different technical backgrounds from the reviewers. The project team can benefit from regular updates on development progress and provide professional growth to the team (ProfessionalQA, 2018). However, performing a code review does not guarantee security on its own but combined with static analysis can ultimately provide more assurance compared to utilising one technique and as a result form a defence-in-depth approach to securing their software.

# Implementation: Describing CVE-2019-10164

*Table 1 Vulnerability Details for CVE-2019-10164 (CVE Details, 2020)*

| | |
|---|---|
| CVE Number | CVE-2019-10164 |
| CVSS Score | 9.0 |
| Confidentiality | Complete |
| Integrity | Complete |
| Availability | Complete |
| Access | Low |
| NVD Link | NVD - CVE-2019-10164 (nist.gov) |

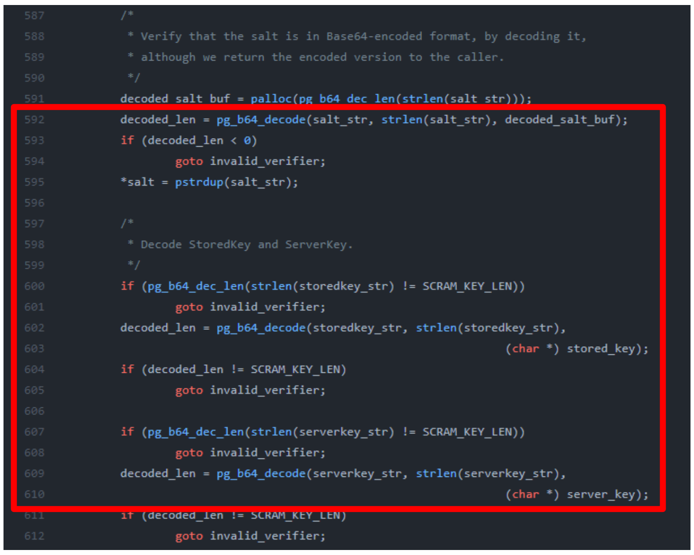
A stack-based buffer overflow exists in PostgreSQL versions 10 before 10.9, and versions 11 before 11.4, which is caused by an authenticated user changing the user's own password to a specially crafted value, allowing for arbitrary code to be executed as the PostgreSQL OS account by overflowing the stack-based buffer (CVE, 2021). This affects total information disclosure, integrity, and availability, as seen in Table 1, meaning it should be taken very seriously. It has previously been observed that this issue concerns SCRAM verifiers in PostgreSQL (MinatoTW, 2021). Moreover, the issue is caused in the 'parse_scram_verifier()' function in 'src/backend/libpq/auth-scram.c' where it copies the server key present in the verified into a buffer of fixed size, with no restrictions present in that write operation. The length of the buffer is set by a constant variable, which is equal to SHA256 digest's length. So, a base64 encoded string containing the specially crafted password can be used to overflow the buffer and cause unexpected results. This is highlighted in Figures 2, 3, and 4.



```
/*
 * Verify a plaintext password against a SCRAM verifier.  This is used when
 * performing plaintext password authentication for a user that has a SCRAM
 * verifier stored in pg_authid.
 */
bool
scram_verify_plain_password(const char *username, const char *password,
                                                        const char *verifier)
{
        char       *encoded_salt;
        char       *salt;
        int                     saltlen;
        int                     iterations;
        uint8           salted_password[SCRAM_KEY_LEN];
        uint8           stored_key[SCRAM_KEY_LEN];
        uint8           server_key[SCRAM_KEY_LEN];
        uint8           computed_key[SCRAM_KEY_LEN];
        char       *prep_password = NULL;
        pg_saslprep_rc rc;

        if (!parse_scram_verifier(verifier, &iterations, &encoded_salt,
                                                          stored_key, server_key))
        {
                /*
                 * The password looked like a SCRAM verifier, but could not be parsed.
                 */
                ereport(LOG,
                                (errmsg("invalid SCRAM verifier for user \"%s\"", username)));
                return false;
        }
```

*Figure 2 'parse_scram_verifier()' Function Flaw*

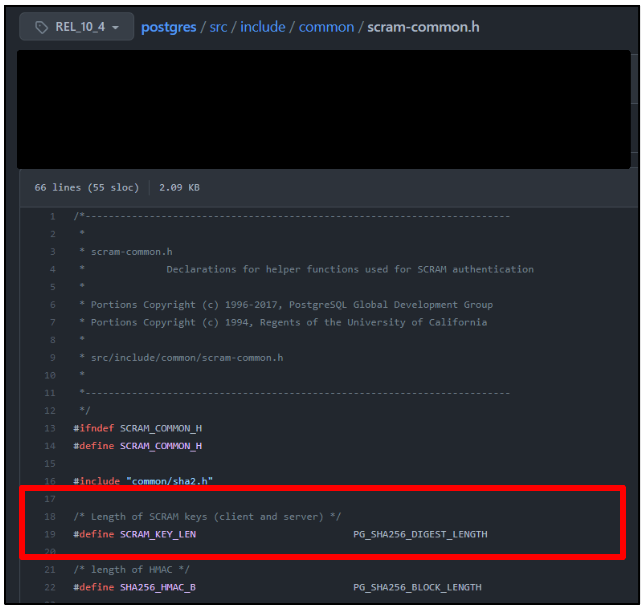


```c
	/*
	 * Verify that the salt is in Base64-encoded format, by decoding it,
	 * although we return the encoded version to the caller.
	 */
	decoded_salt_buf = palloc(pg_b64_dec_len(strlen(salt_str)));
	decoded_len = pg_b64_decode(salt_str, strlen(salt_str), decoded_salt_buf);
	if (decoded_len < 0)
		goto invalid_verifier;
	*salt = pstrdup(salt_str);

	/*
	 * Decode StoredKey and ServerKey.
	 */
	if (pg_b64_dec_len(strlen(storedkey_str) != SCRAM_KEY_LEN))
		goto invalid_verifier;
	decoded_len = pg_b64_decode(storedkey_str, strlen(storedkey_str),
								(char *) stored_key);
	if (decoded_len != SCRAM_KEY_LEN)
		goto invalid_verifier;

	if (pg_b64_dec_len(strlen(serverkey_str) != SCRAM_KEY_LEN))
		goto invalid_verifier;
	decoded_len = pg_b64_decode(serverkey_str, strlen(serverkey_str),
								(char *) server_key);
	if (decoded_len != SCRAM_KEY_LEN)
		goto invalid_verifier;
```

*Figure 3 Code within The Vulnerable Function*



REL_10_4 postgres / src / include / common / scram-common.h

[redacted]

66 lines (55 sloc) | 2.09 KB

```c
/*-------------------------------------------------------------------------
 *
 * scram-common.h
 *		Declarations for helper functions used for SCRAM authentication
 *
 * Portions Copyright (c) 1996-2017, PostgreSQL Global Development Group
 * Portions Copyright (c) 1994, Regents of the University of California
 *
 * src/include/common/scram-common.h
 *
 *-------------------------------------------------------------------------
 */
#ifndef SCRAM_COMMON_H
#define SCRAM_COMMON_H

#include "common/sha2.h"

/* Length of SCRAM keys (client and server) */
#define SCRAM_KEY_LEN				PG_SHA256_DIGEST_LENGTH

/* length of HMAC */
#define SHA256_HMAC_B				PG_SHA256_BLOCK_LENGTH
```

*Figure 4 SCRAM Key Length Leads to Buffer Overflow (32 Bytes)*

More information is highlighted on this vulnerability via the NVD:

[https://nvd.nist.gov/vuln/detail/CVE-2019-10164](https://nvd.nist.gov/vuln/detail/CVE-2019-10164)

## Implementation: Demonstrating Detection

Employing the use of static analysis to analyse the code obtained from the 10.4 release, via *https://github.com/postgres/postgres/tree/REL_10_4* (postgres, 2018), did not return the results required for identification of the vulnerability. The disadvantage of this report is that the investigator was not able to demonstrate use of commercial tools to identify the security faults present, with open-source tools failing to do this. The use of static commercial tools such as SonarCube, Coverity, and premium versions of tools used should be considered in future scenarios to highlight more accurate results. Three open-source static tools were used, namely Flawfinder, Cppcheck, and VisualCodeGrepper, which had varying levels of results. Full options were configured for each scanner, with the scans taking minimal time, although failing to identify the specific fault outlined in this report. The results of these scans are highlighted in the Appendix. Despite these results, running multiple static tools provides more assurance than running just one. In conjunction with code walkthroughs, running static analysis tools earlier can reduce errors detected during this review.

Code walkthroughs could be conducted after the completion of a feature/module and reviewers will utilise a checklist, helping to inform them of issues that could be present. The rise of memory corruption security faults means there is guidance on how to mitigate against these. When investigating the code for possible buffer overflow vulnerabilities, careful attention must be paid to the code when it is dealing with buffer operations. In this case, it may be noticed that the server key is copied into a fixed length buffer, leading to the overflow. This flaw will be pointed out during the review and then immediate action will be taken to fix the affected code.

## Conclusion

The severity of memory corruption faults cannot be understated and is one of the most dangerous vulnerabilities introduced in software. To remediate this, recommendations, such as a defence in depth approach, have been made which help protect against the vulnerability identified, which would have found the issue had it not been utilised, and should improve the effectiveness of ScottishGlen's software development process going forward.

# References

Codacy, 2020. *What are Static Analysis Tools?.* [Online]
Available at: https://blog.codacy.com/what-are-static-analysis-tools/
[Accessed 17 March 2022].

CVE Details, 2020. *Vulnerability Details: CVE-2019-10164.* [Online]
Available at: https://www.cvedetails.com/cve/CVE-2019-10164/?q=CVE-2019-10164
[Accessed 16 March 2022].

CVE, 2021. *CVE-2019-10164.* [Online]
Available at: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2019-10164
[Accessed 14 February 2022].

CWE, 2021. *2021 CWE Top 25 Most Dangerous Software Weaknesses.* [Online]
Available at: https://cwe.mitre.org/top25/archive/2021/2021_cwe_top25.html
[Accessed 17 March 2022].

CWE, 2021. *CWE-787: Out-of-Bounds Write.* [Online]
Available at: https://cwe.mitre.org/data/definitions/787.html
[Accessed 10 March 2022].

IEEE, 2008. IEEE Standard for Software Reviews and Audits. *IEEE Std 1028-2008,* pp. 1-53.

Intel, 2021. *User Guide: Dynamic Analysis vs. Static Analysis.* [Online]
Available at: https://www.intel.com/content/www/us/en/develop/documentation/inspector-user-guide-windows/top/getting-started/dynamic-analysis-vs-static-analysis.html
[Accessed 17 March 2022].

Issam, I., 2017. *Why you should really care about C/C++ static analysis.* [Online]
Available at: https://hackernoon.com/why-you-should-really-care-about-c-c-static-analysis-db13f4463b2d
[Accessed 17 March 2022].

Martello Security, 2022. *Out-of-Bounds Write.* [Online]
Available at: https://www.martellosecurity.com/kb/mitre/cwe/787/
[Accessed 16 March 2022].

MinatoTW, 2021. *CVE-2019-10164: A Case Study.* [Online]
Available at: https://www.hackthebox.com/blog/CVE-2019-10164
[Accessed 16 March 2022].

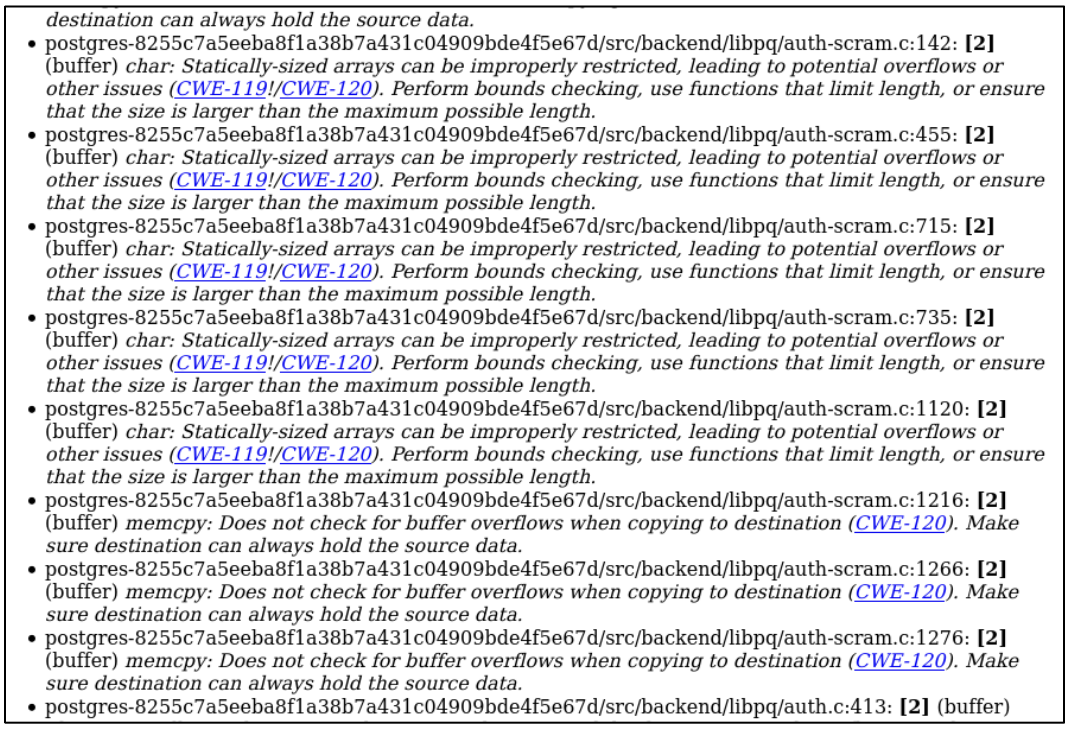
MITRE, 2021. *Secure Code Review.* [Online]
Available at: https://www.mitre.org/publications/systems-engineering-guide/enterprise-engineering/systems-engineering-for-mission-assurance/secure-code-review
[Accessed 17 March 2022].

postgres, 2018. *PostgresSQL 10.4 Github.* [Online]
Available at: https://github.com/postgres/postgres/tree/REL_10_4
[Accessed 17 March 2022].

ProfessionalQA, 2018. *Structured Walkthrough.* [Online]
Available at: https://www.professionalqa.com/structured-walkthrough
[Accessed 18 March 2022].

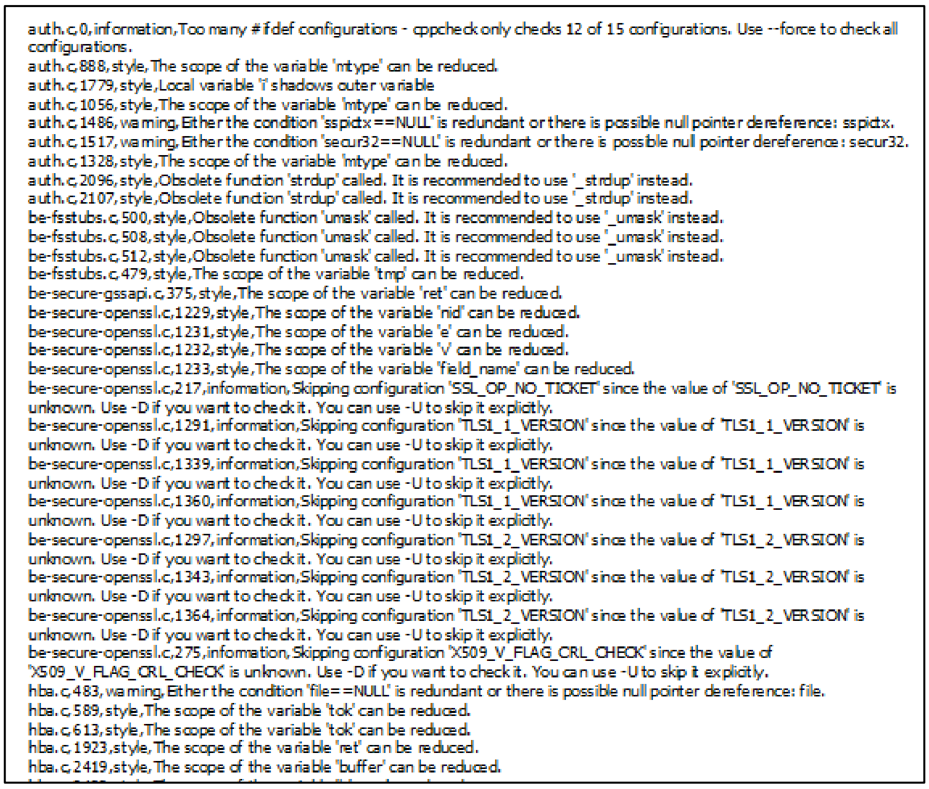
testbytes, 2017. *Pros and Cons of Dynamic Testing and Static Testing.* [Online]
Available at: https://www.testbytes.net/blog/dynamic-testing-and-static-testing/
[Accessed 17 March 2022].

# Appendix: Static Analysis Tool Results



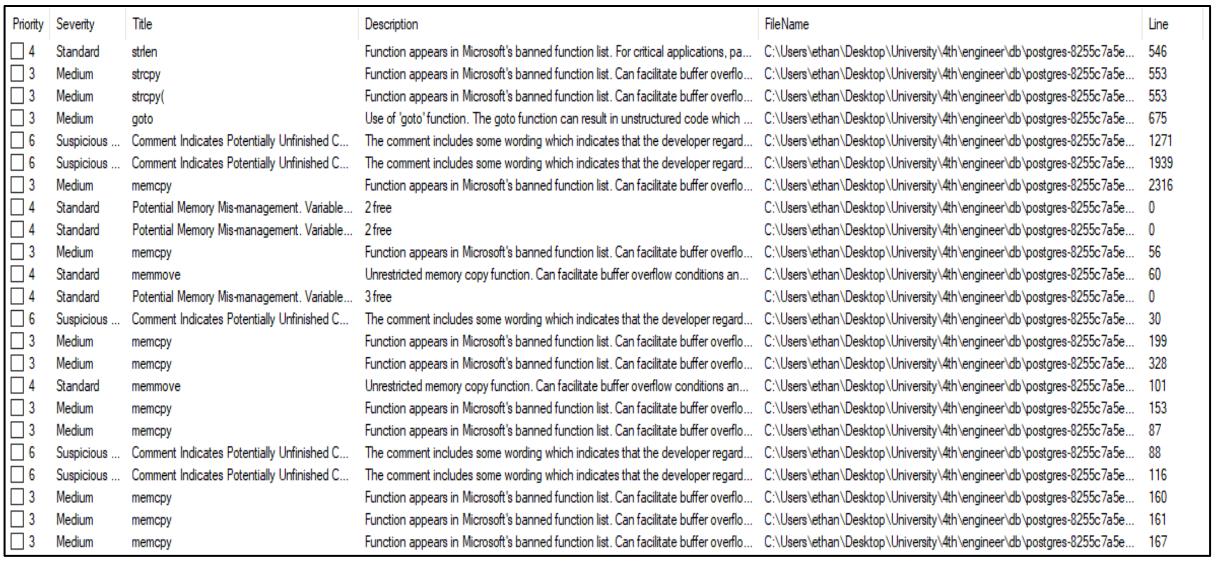
```
destination can always hold the source data.
• postgres-8255c7a5eeba8f1a38b7a431c04909bde4f5e67d/src/backend/libpq/auth-scram.c:142: [2]
  (buffer) char: Statically-sized arrays can be improperly restricted, leading to potential overflows or
  other issues (CWE-119!/CWE-120). Perform bounds checking, use functions that limit length, or ensure
  that the size is larger than the maximum possible length.
• postgres-8255c7a5eeba8f1a38b7a431c04909bde4f5e67d/src/backend/libpq/auth-scram.c:455: [2]
  (buffer) char: Statically-sized arrays can be improperly restricted, leading to potential overflows or
  other issues (CWE-119!/CWE-120). Perform bounds checking, use functions that limit length, or ensure
  that the size is larger than the maximum possible length.
• postgres-8255c7a5eeba8f1a38b7a431c04909bde4f5e67d/src/backend/libpq/auth-scram.c:715: [2]
  (buffer) char: Statically-sized arrays can be improperly restricted, leading to potential overflows or
  other issues (CWE-119!/CWE-120). Perform bounds checking, use functions that limit length, or ensure
  that the size is larger than the maximum possible length.
• postgres-8255c7a5eeba8f1a38b7a431c04909bde4f5e67d/src/backend/libpq/auth-scram.c:735: [2]
  (buffer) char: Statically-sized arrays can be improperly restricted, leading to potential overflows or
  other issues (CWE-119!/CWE-120). Perform bounds checking, use functions that limit length, or ensure
  that the size is larger than the maximum possible length.
• postgres-8255c7a5eeba8f1a38b7a431c04909bde4f5e67d/src/backend/libpq/auth-scram.c:1120: [2]
  (buffer) char: Statically-sized arrays can be improperly restricted, leading to potential overflows or
  other issues (CWE-119!/CWE-120). Perform bounds checking, use functions that limit length, or ensure
  that the size is larger than the maximum possible length.
• postgres-8255c7a5eeba8f1a38b7a431c04909bde4f5e67d/src/backend/libpq/auth-scram.c:1216: [2]
  (buffer) memcpy: Does not check for buffer overflows when copying to destination (CWE-120). Make
  sure destination can always hold the source data.
• postgres-8255c7a5eeba8f1a38b7a431c04909bde4f5e67d/src/backend/libpq/auth-scram.c:1266: [2]
  (buffer) memcpy: Does not check for buffer overflows when copying to destination (CWE-120). Make
  sure destination can always hold the source data.
• postgres-8255c7a5eeba8f1a38b7a431c04909bde4f5e67d/src/backend/libpq/auth-scram.c:1276: [2]
  (buffer) memcpy: Does not check for buffer overflows when copying to destination (CWE-120). Make
  sure destination can always hold the source data.
• postgres-8255c7a5eeba8f1a38b7a431c04909bde4f5e67d/src/backend/libpq/auth.c:413: [2] (buffer)
```

*Figure 5 Example Results of The Vulnerable File Identified by Flawfinder*



```
auth.c,0,information,Too many #ifdef configurations - cppcheck only checks 12 of 15 configurations. Use --force to check all
configurations.
auth.c,888,style,The scope of the variable 'mtype' can be reduced.
auth.c,1779,style,Local variable 'i' shadows outer variable
auth.c,1056,style,The scope of the variable 'mtype' can be reduced.
auth.c,1486,warning,Either the condition 'sspictx==NULL' is redundant or there is possible null pointer dereference: sspictx.
auth.c,1517,warning,Either the condition 'secur32==NULL' is redundant or there is possible null pointer dereference: secur32.
auth.c,1328,style,The scope of the variable 'mtype' can be reduced.
auth.c,2096,style,Obsolete function 'strdup' called. It is recommended to use '_strdup' instead.
auth.c,2107,style,Obsolete function 'strdup' called. It is recommended to use '_strdup' instead.
be-fsstubs.c,500,style,Obsolete function 'umask' called. It is recommended to use '_umask' instead.
be-fsstubs.c,508,style,Obsolete function 'umask' called. It is recommended to use '_umask' instead.
be-fsstubs.c,512,style,Obsolete function 'umask' called. It is recommended to use '_umask' instead.
be-fsstubs.c,479,style,The scope of the variable 'tmp' can be reduced.
be-secure-gssapi.c,375,style,The scope of the variable 'ret' can be reduced.
be-secure-openssl.c,1229,style,The scope of the variable 'nid' can be reduced.
be-secure-openssl.c,1231,style,The scope of the variable 'e' can be reduced.
be-secure-openssl.c,1232,style,The scope of the variable 'v' can be reduced.
be-secure-openssl.c,1233,style,The scope of the variable 'field_name' can be reduced.
be-secure-openssl.c,217,infomation,Skipping configuration 'SSL_OP_NO_TICKET' since the value of 'SSL_OP_NO_TICKET' is
unknown. Use -D if you want to check it. You can use -U to skip it explicitly.
be-secure-openssl.c,1291,information,Skipping configuration 'TLS1_1_VERSION' since the value of 'TLS1_1_VERSION' is
unknown. Use -D if you want to check it. You can use -U to skip it explicitly.
be-secure-openssl.c,1339,information,Skipping configuration 'TLS1_1_VERSION' since the value of 'TLS1_1_VERSION' is
unknown. Use -D if you want to check it. You can use -U to skip it explicitly.
be-secure-openssl.c,1360,information,Skipping configuration 'TLS1_1_VERSION' since the value of 'TLS1_1_VERSION' is
unknown. Use -D if you want to check it. You can use -U to skip it explicitly.
be-secure-openssl.c,1297,information,Skipping configuration 'TLS1_2_VERSION' since the value of 'TLS1_2_VERSION' is
unknown. Use -D if you want to check it. You can use -U to skip it explicitly.
be-secure-openssl.c,1343,information,Skipping configuration 'TLS1_2_VERSION' since the value of 'TLS1_2_VERSION' is
unknown. Use -D if you want to check it. You can use -U to skip it explicitly.
be-secure-openssl.c,1364,information,Skipping configuration 'TLS1_2_VERSION' since the value of 'TLS1_2_VERSION' is
unknown. Use -D if you want to check it. You can use -U to skip it explicitly.
be-secure-openssl.c,275,infomation,Skipping configuration 'X509_V_FLAG_CRL_CHECK' since the value of
'X509_V_FLAG_CRL_CHECK' is unknown. Use -D if you want to check it. You can use -U to skip it explicitly.
hba.c,483,warning,Either the condition 'file==NULL' is redundant or there is possible null pointer dereference: file.
hba.c,589,style,The scope of the variable 'tok' can be reduced.
hba.c,613,style,The scope of the variable 'tok' can be reduced.
hba.c,1923,style,The scope of the variable 'ret' can be reduced.
hba.c,2419,style,The scope of the variable 'buffer' can be reduced.
```

*Figure 6 Cppcheck Analysis*



| Priority | Severity | Title | Description | FileName | Line |
|---|---|---|---|---|---|
| ☐ 4 | Standard | strlen | Function appears in Microsoft's banned function list. For critical applications, pa... | C:\Users\ethan\Desktop\University\4th\engineer\db\postgres-8255c7a5e... | 546 |
| ☐ 3 | Medium | strcpy | Function appears in Microsoft's banned function list. Can facilitate buffer overflo... | C:\Users\ethan\Desktop\University\4th\engineer\db\postgres-8255c7a5e... | 553 |
| ☐ 3 | Medium | strcpy( | Function appears in Microsoft's banned function list. Can facilitate buffer overflo... | C:\Users\ethan\Desktop\University\4th\engineer\db\postgres-8255c7a5e... | 553 |
| ☐ 3 | Medium | goto | Use of 'goto' function. The goto function can result in unstructured code which ... | C:\Users\ethan\Desktop\University\4th\engineer\db\postgres-8255c7a5e... | 675 |
| ☐ 6 | Suspicious ... | Comment Indicates Potentially Unfinished C... | The comment includes some wording which indicates that the developer regard... | C:\Users\ethan\Desktop\University\4th\engineer\db\postgres-8255c7a5e... | 1271 |
| ☐ 6 | Suspicious ... | Comment Indicates Potentially Unfinished C... | The comment includes some wording which indicates that the developer regard... | C:\Users\ethan\Desktop\University\4th\engineer\db\postgres-8255c7a5e... | 1939 |
| ☐ 3 | Medium | memcpy | Function appears in Microsoft's banned function list. Can facilitate buffer overflo... | C:\Users\ethan\Desktop\University\4th\engineer\db\postgres-8255c7a5e... | 2316 |
| ☐ 4 | Standard | Potential Memory Mis-management. Variable... | 2 free | C:\Users\ethan\Desktop\University\4th\engineer\db\postgres-8255c7a5e... | 0 |
| ☐ 4 | Standard | Potential Memory Mis-management. Variable... | 2 free | C:\Users\ethan\Desktop\University\4th\engineer\db\postgres-8255c7a5e... | 0 |
| ☐ 3 | Medium | memcpy | Function appears in Microsoft's banned function list. Can facilitate buffer overflo... | C:\Users\ethan\Desktop\University\4th\engineer\db\postgres-8255c7a5e... | 56 |
| ☐ 4 | Standard | memmove | Unrestricted memory copy function. Can facilitate buffer overflow conditions an... | C:\Users\ethan\Desktop\University\4th\engineer\db\postgres-8255c7a5e... | 60 |
| ☐ 4 | Standard | Potential Memory Mis-management. Variable... | 3 free | C:\Users\ethan\Desktop\University\4th\engineer\db\postgres-8255c7a5e... | 0 |
| ☐ 6 | Suspicious ... | Comment Indicates Potentially Unfinished C... | The comment includes some wording which indicates that the developer regard... | C:\Users\ethan\Desktop\University\4th\engineer\db\postgres-8255c7a5e... | 30 |
| ☐ 3 | Medium | memcpy | Function appears in Microsoft's banned function list. Can facilitate buffer overflo... | C:\Users\ethan\Desktop\University\4th\engineer\db\postgres-8255c7a5e... | 199 |
| ☐ 3 | Medium | memcpy | Function appears in Microsoft's banned function list. Can facilitate buffer overflo... | C:\Users\ethan\Desktop\University\4th\engineer\db\postgres-8255c7a5e... | 328 |
| ☐ 4 | Standard | memmove | Unrestricted memory copy function. Can facilitate buffer overflow conditions an... | C:\Users\ethan\Desktop\University\4th\engineer\db\postgres-8255c7a5e... | 101 |
| ☐ 3 | Medium | memcpy | Function appears in Microsoft's banned function list. Can facilitate buffer overflo... | C:\Users\ethan\Desktop\University\4th\engineer\db\postgres-8255c7a5e... | 153 |
| ☐ 3 | Medium | memcpy | Function appears in Microsoft's banned function list. Can facilitate buffer overflo... | C:\Users\ethan\Desktop\University\4th\engineer\db\postgres-8255c7a5e... | 87 |
| ☐ 6 | Suspicious ... | Comment Indicates Potentially Unfinished C... | The comment includes some wording which indicates that the developer regard... | C:\Users\ethan\Desktop\University\4th\engineer\db\postgres-8255c7a5e... | 88 |
| ☐ 6 | Suspicious ... | Comment Indicates Potentially Unfinished C... | The comment includes some wording which indicates that the developer regard... | C:\Users\ethan\Desktop\University\4th\engineer\db\postgres-8255c7a5e... | 116 |
| ☐ 3 | Medium | memcpy | Function appears in Microsoft's banned function list. Can facilitate buffer overflo... | C:\Users\ethan\Desktop\University\4th\engineer\db\postgres-8255c7a5e... | 160 |
| ☐ 3 | Medium | memcpy | Function appears in Microsoft's banned function list. Can facilitate buffer overflo... | C:\Users\ethan\Desktop\University\4th\engineer\db\postgres-8255c7a5e... | 161 |
| ☐ 3 | Medium | memcpy | Function appears in Microsoft's banned function list. Can facilitate buffer overflo... | C:\Users\ethan\Desktop\University\4th\engineer\db\postgres-8255c7a5e... | 167 |

*Figure 7 Example Output from Visual Code Grepper*