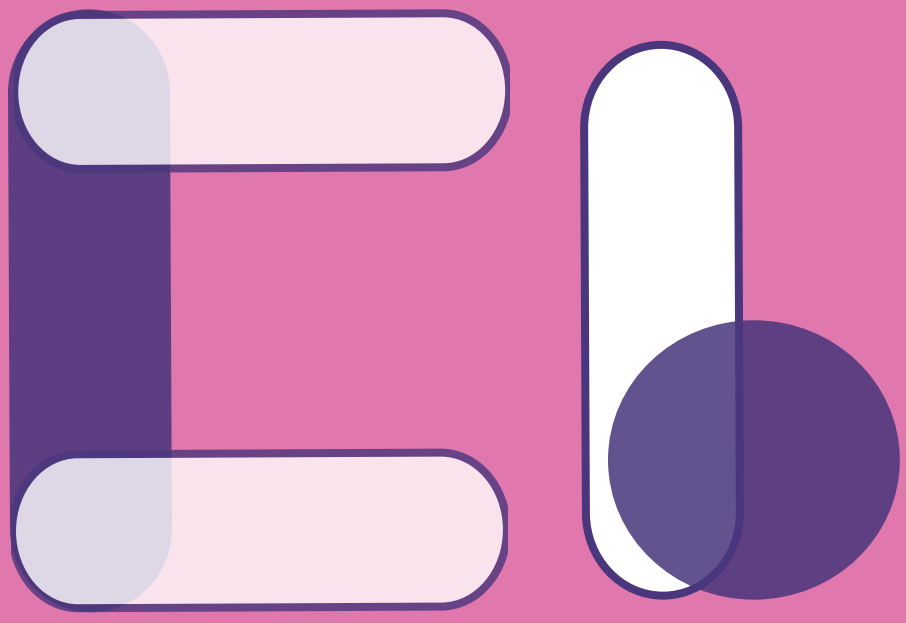
A Security Guide for COBOL Developers



kiuwan
An Idera, Inc. Company

INDEX

COBOL

Overview

For the past several decades, businesses and governmental agencies have been using the **COBOL programming language** for numerous applications. Despite its age, COBOL remains popular and widespread across industries such as banking, finance, and commerce. This is in part due to COBOL's highly user-friendly design and imperative orientation. Today, COBOL is commonly used in the development of applications for mainframe computing operations, particularly for operations such as batch processing and transaction processing that are commonly used in businesses across the world.

However, though COBOL offers businesses and governmental agencies many upsides in terms of ease of use and convenience, it also has a few drawbacks of which users need to be aware. In particular, COBOL programming languages often come with potential security vulnerabilities that may compromise critical data contained within a COBOL application. These vulnerabilities are particularly concerning given the areas where COBOL is most common. For example, banks and other financial institutions that use COBOL language applications may risk exposing the financial and personal data of customers to hackers if COBOL security risks are ignored. Many of these COBOL vulnerabilities can result in significant data breaches that compromise customer or user data, and cost companies millions of dollars in lost income and resource expenditures.

Of course, with a programming language as convenient and engrained as COBOL is, developers and IT professionals cannot expect it to go away any time soon. Therefore, DevSecOps professionals, cybersecurity experts, and other IT managers need to understand the most common security risks associated with COBOL programming languages. With a firm understanding of these risks, IT professionals who work with COBOL systems can beef up their data security and reduce the risks of critical data breaches. A cybersecurity partner like Kiuwan can also provide comprehensive security solutions for professionals working with COBOL systems. In this guide, we walk you through just that.

The COBOL *Legacy Language*

The Common Business-Oriented Language, commonly known as COBOL, is one of the oldest extant programming languages that sees widespread use today. COBOL's origins go back to the 1950s when it was developed as part of the Conference/Committee on Data Systems Languages (CODASYL) as an offshoot of the earlier FLOW-MATIC programming language developed by Grace Hopper. COBOL's development came in conjunction with the United States Department of Defense, which sought to create a more portable programming language that could be used for easy and user-friendly data processing.

Based on its origins at the intersection of business and government, it should come as no surprise that COBOL's continued popularity today comes mainly from the corporate and governmental sectors. Since COBOL's initial design centered on easy data processing for user-friendly applications, businesses across several industries were able to quickly implement COBOL languages into their own transaction processing needs.

Over time, these programs became ingrained in the digital sediment used by many companies and governmental agencies, even as newer programming languages arrived in the subsequent decades. Today, many businesses and government agencies prefer to maintain COBOL applications that still sufficiently serve their needs, rather than expend the time and resources needed to upgrade to a new programming language. Given its age and the fact that it still sees widespread use, COBOL is commonly referred to as a legacy language.

One area where COBOL stands out among other programming languages is the fact that its development and standardization arose from the spheres of business and government. COBOL's initial development in the 1950s came about in part because computer scientists at the time were more concerned with academic issues in mathematics and the physical sciences, and were not dedicating sufficient energy to the development of programming languages that could meet the data processing needs of businesses and governments.

COBOL and *Software Security*

The lack of input from computer scientists in COBOL's development has both upsides and downsides. On the one hand, the fact that it was designed by business professionals and government employees means that it is much more user-friendly for IT professionals in these areas than many programming languages designed by computer scientists. However, the drawback of this is that COBOL may be more vulnerable to certain security risks than more computer science-oriented languages.

Another related issue is that fewer and fewer new programmers and computer scientists have specific training with COBOL, given its age and lack of status within the computer science world. This can mean that businesses and agencies that still rely on COBOL applications may have a hard time finding IT professionals with the specific COBOL expertise needed to mitigate COBOL's security risks. Still, as a legacy language ubiquitous in business and government, COBOL will likely remain in use for the foreseeable future. Therefore, professionals working with COBOL systems should be proactive in understanding the security risks they are most likely to face, and how to stop them before they happen.

SQL Injections

One of the most common security vulnerabilities found in COBOL applications is the risk of SQL injections. SQL injections are a particular type of data attack that takes advantage of weaknesses in an application's Structure Query Language (SQL). When an SQL injection occurs, an attacker will access the application via the client end. Once attackers can access the application, they will insert malicious SQL code into the application's entry field for further execution. The insertion of malicious SQL code can allow an attacker to access data contained within the program itself.

If an *SQL attack is successful*, attackers can access, change, or destroy any data within the applications. For example, a hacker who uses an SQL injection attack can change financial information within a bank's COBOL application on financial transactions. An attacker can also mimic the identity of authorized users in the system to access or change critical data on the system's end. In some cases, an attacker can use an SQL injection on a COBOL system to gain authorization as a system administrator. This can allow them to lock out other users, change security settings, steal data, or otherwise alter the application's main functions.

For an SQL injection to be successful, an application must have a preexisting security vulnerability for the attacker to exploit. This is because SQL injection attacks require exploitable security flaws. These include failures to prevent weak-type input and insufficient filters for SQL statements inputted into the system. For this reason, COBOL systems are particularly vulnerable to SQL injection attacks.

COBOL systems usually use frequent SQL statements for the data that is inputted and outputted within the system.

However, COBOL languages often do not have sufficient parameters for their systems' SQL input. In these instances, the program requires more user input to build functional statements for essential system operations. Though this is a source of the very user-friendly status that makes COBOL so popular in business and government, it also means that a COBOL application's SQL statements often lack sufficient parameters. Attackers can exploit this vulnerability to inject malicious SQL code into the application and cause a significant data breach or security issue.

Fortunately, SQL injections are also one of the easier security risks to mitigate, even with an older legacy language like COBOL. IT professionals working with a COBOL system should be proactive in making sure that their applications have appropriate parameters. In these cases, the input would be based on the parameters, rather than user input. This would go a long way in preventing opportunities for malicious actors to induce SQL injections into the system. Firewalls, though unable to fully prevent SQL injection attacks, can make the process of injecting malicious SQL code much more difficult, and potentially discourage any would-be attacker.

Insufficient Input Validation

Another major security vulnerability in COBOL systems is the potential for a lack of sufficient input validation controls on the user end of COBOL applications. Though all sorts of programming languages can experience insufficient input validation controls, COBOL systems may be particularly vulnerable to this due to a combination of the language's age, its high degree of user input, and its origins and development outside of the sphere of computer science.

All web and mobile applications that involve user input must have sufficient controls to sanitize the data that gets entered into the system. Without proper input validations in place, COBOL applications can be vulnerable to a wide range of different types of attacks and security breacInsufficientcient input validation in COBOL systems can be a significant cause of the aforementioned SQL injection attacks. Additionally, insufficient input validation controls can cause a COBOL system to experience other types of attacks, such as cross-site scripting and code injection. When these occur, attackers can access critical data within the system, abuse authentication controls, and shut out other users from the application.

The good news here is that these types of vulnerabilities are relatively easy to fix in COBOL sysOnet. One of the main reasons why COBOL systems may lack sufficient input validation controls is simply because of the lack of new IT professionals with specific training in the COBOL programming language. Companies, government offices, and other organizations that still rely on COBOL applications can go a long way in securing this particular vulnerability with a bit of proactive education in input validation control programming in COBOL systems. With users more aware of what validations need to be in place for all...

data within the system, and with IT professionals more aware of protocols for programming input validations within COBOL systems, these applications can easily become much more secure from input validation attacks.

Non-Critical Code Smells

In the world of computer programming, code smells are a more generalized issue that nonetheless may point to an underlying vulnerability in a code. In essence, a code smell is a symptom of a deeper issue within the coding itself, one that may not be a critical flaw in and of itself but may point to a deeper bug or issue within the code.

Code smells that are non-critical may, in a certain sense, be more pernicious than critical code smells that induce system failures. If a code smell causes a continuous annoyance but does not rise to the level of significant system failure, users and administrators may be more likely to ignore the underlying issues and let the problem fester. But this means that the system will remain vulnerable to whatever bug or error is causing the smell, and data within the system may be at a higher risk of being exposed in an attack or security breach.

Common code smells that may affect COBOL systems are:

Duplicate Code

Duplicate codes often occur when bits of code are copied and pasted multiple times into one application. When similar or identical lines of code exist in more than one location in a program, the program's operations may fail to function properly, and essential security protocols may fail to prevent malicious attacks from outside the system.

Bloaters

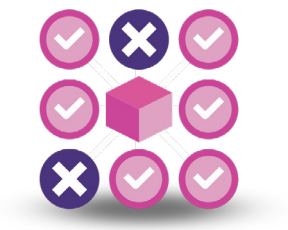
Bloaters happen when certain methods or classes of code in a particular program get bloated beyond their normal capacity. This tends to occur when a line of code is not properly designated within an application and ends up doing too many things at once, or operating without a clear function. Bloaters can cause the entire application to suffer delays and system errors. These, in turn, can hinder an application's security controls and make it more vulnerable to attacks and data breaches.

Lazy Classes

Lazy classes of code are codes that do not have a particular function within the current system's operations. Oftentimes, these classes of code were designed in anticipation of future features that don't yet exist within the current system. Since they do not serve a clear purpose within the operation as it exists now, these lazy classes often end up slowing down the program as a whole and making the operations much harder to work with on the user end. Lazy classes can also result in security vulnerabilities by compromising things like input validations and code parameters.

Though non-critical code smells can affect all sorts of programming languages, COBOL programs tend to be particularly vulnerable to them. This is another consequence of COBOL's origins outside of the realm of computer science, where coding precision is a greater emphasis during the process of programming language design. In general, though, these issues can be more easily solved with some simple proactive measures for all users. When a company or agency uses COBOL systems, it's important for all users to recognize the signs of code smells and what kinds of coding issues they may point to. It's also important for IT professionals working with these systems to respond to all code smells, even though they are not critical, as a way to prevent future security vulnerabilities.

Insufficient Security Validation Controls



Similar to the issue with insufficient input validation, insufficient security validation controls stem from broader failures in COBOL system administrators and users to maintain sufficient security protocols in the system validation process. Users who access data within a COBOL application should have specific authentication in place. However, this means that the application's security validation controls need to be sufficient for authenticating all authorized users who access the system's data. Without sufficient security validation controls, attackers can more easily bypass the system's security protocols and access critical data within the system itself.

COBOL applications can be particularly vulnerable to a few specific types of security validation control issues. For example, many COBOL systems may suffer from issues with duplicate validation forms. These occur when multiple validation forms within the system's security validation have the same names. If this happens, the system will usually choose one validation form arbitrarily, and discard the other. This can result in issues with users accessing the system, or failures in the system responding to requests for data or other operations. It can also result in attackers exploiting duplicate validation forms to access critical data or shut down the system entirely.

Another security validation control issue that may occur in COBOL systems is erroneous validation methods. When these kinds of issues occur, normal user validation may be hindered or compromised entirely, as users fail to gain access to the application through the normal validation process. The main risk of this type of issue is that the system administrators will disable the COBOL system's security validation procedures entirely rather than fix the underlying issues with the security validation controls. This would obviously leave the system much more vulnerable to outside attacks and data breaches.

Security Vulnerabilities Due to Defective Code

One of the consequences of a long-standing legacy programming language like COBOL is that it has accumulated a degree of defective code over the decades in which it has been in use. Despite COBOL's resiliency and user-friendly orientation, COBOL systems still may contain defective code that leaves them vulnerable to broader security risks or data breaches.

Today, researchers estimate that there are about 220 billion lines of COBOL code across the public and private sectors. Most experts estimate that somewhere between 0.5% and 0.7% of this code is defective in some way. While this may not sound like a lot, it would still represent millions of lines of defective code embedded in critical programs in banks, software companies, and governmental agencies. If each line of defective code represents one potential security vulnerability, users may experience around 27.5 million such vulnerabilities across 220 billion lines of COBOL code.

Of course, all programming languages have some degree of defective code. But COBOL stands out somewhat in the field of programming languages for its legacy status and its embedment within various businesses. When working with COBOL applications, IT professionals can mitigate the security vulnerabilities brought on by these tens of millions of lines of potentially defective code by investing in data-centric security protocols. Kiuwan's appsec solutions are a great resource for implementing the necessary security protocols into legacy languages like COBOL. Data-centric security solutions can separate data security from vulnerable software, and minimize the risk of security vulnerabilities resulting from defective COBOL code.

The *Cost of Not* Securing COBOL Applications

COBOL's ubiquity across important industries such as banking, as well as its role in governance, means that COBOL-related security breaches can be particularly harmful. In particular, COBOL's widespread use in the U.S. government can lead to important data being exposed. Recent oversight has found that security breaches in U.S. government COBOL systems have risen by more **than 1,000% between 2006 and 2014**. In 2015, the U.S. government's Office of Personnel Management **experienced a major security breach** in its COBOL systems that exposed the data of around 18 million government employees.

These types of attacks against legacy language systems like COBOL can end up costing significant amounts of money. In 2016 alone, the U.S. government spent around 75% of its $80 billion IT budget responding to security vulnerabilities in its extant legacy language systems, including many that use COBOL.

Despite these frustrations, the fact remains that most of these security vulnerabilities in COBOL can be easily mitigated with simple proactive implementations of security protocols, as well as a bit of education among all system users. Small upfront investments in modernized COBOL security solutions can save government agencies and private companies millions of dollars that they would otherwise lose in costly data breaches. Given the general lack of contemporary IT training in COBOL systems, user education in appropriate COBOL security protocols is another great way for companies, governmental agencies, and other organizations to work together to make their data and software much more secure.

Securing COBOL Data and Preventing Breaches

Though COBOL remains vulnerable to these and other types of security risks and data breaches, its status as a legacy language means that it will remain a critical part of the digital infrastructure of both businesses and governments. Fortunately, the convenient nature of COBOL programming means that COBOL security solutions can be quite convenient as well. Kiuwan's cybersecurity solutions offer an excellent path forward for all IT professionals—in both the public and private spheres—to secure their COBOL applications and prevent costly data breaches.

Kiuwan offers exceptional SAST and SCA solutions, as well as user-friendly system add-ons that can mitigate the types of security risks that COBOL systems may be vulnerable to. So, if you are an IT professional working with a legacy language like COBOL, get in touch with Kiuwan today to connect with our team of cybersecurity experts. You can also request a demo of Kiuwan's security solutions, start a free trial, or read our blog for more helpful information.

*YOU KNOW **CODE**, WE KNOW **SECURITY!***

GET IN TOUCH:

Headquarters
2950 N Loop Freeway W, Ste 700
Houston, TX 77092, USA

United States **+1 732 895 9870**
Asia-Pacific, Europe, Middle East and Africa **+44 1628 684407**
contact@kiuwan.com
Partnerships: **partners@kiuwan.com**