12-2012

Role based access control and authentication for SCADA field devices using a dual Bloom filter and challenge-response.

Jacob Schreiver
University of Louisville

ROLE BASED ACCESS CONTROL AND AUTHENTICATION FOR SCADA FIELD DEVICES USING A DUAL BLOOM FILTER AND CHALLENGE-RESPONSE

By

Jacob Schreiver
B.S., University of Louisville, 2011

A Thesis
Submitted to the Faculty of the
University of Louisville
J. B. Speed School of Engineering
as Partial Fulfillment of the Requirements
for the Professional Degree

Master of Engineering

Department of Computer Engineering and Computer Science

December 2012

ROLE BASED ACCESS CONTROL AND AUTHENTICATION FOR SCADA FIELD DEVICES USING A DUAL BLOOM FILTER AND CHALLENGE-RESPONSE

Submitted by: ___________________________________

Jacob Schreiver

A Thesis Approved On

____

(Date)

by the Following Reading and Examination Committee:

____

Ahmed Desoky, Co-Director

____

Jeffery Hieb, Co-Director

____

James Graham

ACKNOWLEDGEMENTS

I would like to thank Dr Jeff Hieb for encouraging me to do this thesis, as well as his support throughout the entire thesis process. I would also like to thank my family who have supported me my entire life in everything I do, without them none of this would have been possible. Lastly I would like to thank all my friends who put up with me and help me through the long days and nights as I completed this endeavor.

ABSTRACT

Supervisory control and data acquisition (SCADA) systems are networked control systems used in many critical infrastructure areas such as power water and transportation. Many of these systems continue to use legacy field devices that lack cyber security features. The field device security preprocessor is a bump-in-the-wire security solution of legacy field devices. This thesis describes the design and analysis of a dual Bloom filter structure for use in a field device security preprocessor. A dual Bloom filter is a variant of the traditional Bloom filter, that performs role based access checks in O(1) time. It is shown this structure, which can produce false authentications is shown to be acceptable for this security use thought analysis and penetration testing. Analysis and testing shows that in spite of false positives this structure can provide the required level of security, while maintaining the required level of performance on low cost hardware.

TABLE OF CONTENTS

ACKNOWLEDGEMENTS ii
ABSTRACT iii
TABLE OF CONTENTS iv
LIST OF TABLES vii
LIST OF FIGURES viii
1. Introduction 1
  1.1. Background and Motivation 1
  1.2. Designing For Security, Designing for Feasibility 2
  1.3. Organization 4
2. Introduction to Bloom Filters 6
  2.1. Properties of Bloom Filters 7
  2.2. The Random Filter 12
  2.3. Parameter Selection for building a Bloom Filter 14
  2.4. Applications of Bloom Filters 18
  2.5. A Bloom Filter for Role Based Access Control 21
  2.6. Variations and Extensions of Bloom Filters 22

3. A Dual Bloom Filter Structure for Efficient Implementation of Role Based Access Control and Challenge Response for a Field Device Security Pre-Processor 24

3.1. The Need for a SCADA Field Device Security Pre-Processor 24

3.2. Access Control and Challenge Response in SCADA networks 25

3.2.1. Details of the Extended Modbus Function Codes 27

3.3. Dual Bloom Filters for Modbus Role Based Access Control (RBAC) 29

3.4. Creation and Analysis of an Example Dual Bloom Filter 32

3.5. Comparison of the Two Challenge Response Implementations 38

4. Implementation of the Field Device Security Pre-Processor for SCADA using Dual Bloom Filters for Access Control 48

4.1. The Field Device Security Pre-Processor 48

4.2. The Microkernel Architecture for the FD-SPP 49

4.3. The OKL4 Cell Structure used for the FD-SPP 52

4.4. Software Design of OKL4 Cells 55

4.5. Implementation of the Dual Bloom Filters for RBAC 62

4.6. An Example Access Check in the Dual Bloom Filters 64

4.7. Reduction of the False Positive Rate 65

4.8. Prototype of the Field Device Security Preprocessor 70

5. Testing of the Field Device Security Pre-processor 72

5.1. Java Modbus HMI Design 72

5.2. Initial performance data 74

5.3. Modbus compliance testing 76

5.3.1. Test Login Success 77

5.3.2. Test Login Invalid User 78

5.3.3. Test Login Invalid Password 78

5.3.4. Test Packet RBAC Challenge 79

5.3.5. Test Challenge-Response 80

5.3.6. Test Packet Reconstruction 81

5.3.7. Test RBAC Suspicious Mode 81

5.3.8. Test Write Coils 82
5.3.9. Test Read Coils 83
5.4. Penetration Testing 83
5.4.1. Write Coils Attack 85
5.4.2. Write Random Coils Repeatedly Attack 85
5.4.3. Theoretical Write Coils Attacks 86
5.4.4. Read Coils Attack 90
5.4.5. Attacks on the HMI 91
5.4.6. Read Coils Attack on HMI 91
5.4.7. Write Coils Attack on HMI 92
5.5. Digital Signatures for Return Packets 92
5.5.1. Creation of the Digitally Signed Messages 92
5.5.2. Timing Data for Signed Packets 93
5.5.3. Nonce Generation for the Signature 94
6. Conclusion and Future Work 97
6.1. Results Summary 97
6.2. Direction for Future Work 98

LIST OF TABLES

TABLE 3.2.1 Extended Modbus Function Codes 27
TABLE 3.3.1 – Dual Bloom Filter RBAC 30
TABLE 3.4.1 Example <Role, Modbus Packet> Entries 34
TABLE 3.5.1 Updated RBAC Responses 46
TABLE 5.2.1 Initial Timing Data 76
TABLE 5.4.1 Penetration testing results. 85
TABLE 5.5.1 Timing data for return message digital signatures 94

LIST OF FIGURES

FIGURE 2.1.1 - Adding elements A and B to a Bloom Filter 8
FIGURE 2.1.2 - Determining if C and D are in the set 9
FIGURE 2.3.1 Comparison of False Positive Rate for two different data sets......17
FIGURE 2.3.2 Comparison of Percent Difference from Theoretical False Positive Rate for two different data sets 17
FIGURE 3.2.1 - Typical elements of a Modbus message 25
FIGURE 3.2.2 - Standard Modbus exchange 26
FIGURE 3.2.3 - Extended Modbus Connection Request 27
FIGURE 3.2.4 - Packet Structure for Connection Request 28
FIGURE 3.2.5 - Packet Structure for Challenge Packet 29
FIGURE 3.3.1 - Dual Bloom Filter RBAC Check 32
FIGURE 3.5.1 Comparison of false positive rates of Challenge and Non-Challenge Implementations with p = .001 43
FIGURE 3.5.2 Comparison of false positive rates of Challenge and Non-Challenge Implementations with r = .75 44
FIGURE 3.5.3 Updated Flow Diagram for RBAC check 47
FIGURE 4.1.1 Placement of the FD-SPP in a simple SCADA system...................49
FIGURE 4.3.1 OKL4 cells for the FD-SPP 52
FIGURE 4.3.2 Cell Communication Flow 54
FIGURE 4.4.1 UART read flow chart 56
FIGURE 4.4.2 Communication Cell Circular Buffer 57
FIGURE 4.4.3 Master Cell second Modbus placement in circular buffer 57
FIGURE 4.4.4 Modbus message reconstruction 58
FIGURE 4.4.5 Flow in the UART write thread 59
FIGURE 4.4.6 Flow of a packet through the filtering cell 61
FIGURE 4.4.7 Modbus packet determination flow 61

FIGURE 4.5.1 Bloom Filter Creation Software Screenshot 63
FIGURE 4.6.1 Using SHA-256 to Add Entries to a Bloom filter 65
FIGURE 4.7.1 Probability Distribution of number of ones in the Bloom Filter 69
FIGURE 4.8.1 Field Device Security Preprocessor Prototype 71
FIGURE 5.1.1 Java Modbus HMI..74
FIGURE 5.2.1 Modbus packet propagation...75
FIGURE 5.3.1 Compliance testing system diagram 77
FIGURE 5.3.2 Compliance Checker Output..77
FIGURE 5.3.3 Digital output dialog displayed to Tester....................................82
FIGURE 5.3.4 Read coils dialog box...83
FIGURE 5.4.1 Penetration testing setup with FD-SPP.......................................84
FIGURE 5.4.2 Penetration testing setup control setup......................................84
FIGURE 5.5.1 Return Packet Digital Signature..93

1. INTRODUCTION

This thesis describes the design, development and testing of a field device security preprocessor for role based access control and challenge response using dual Bloom filters. The development of this device comes out of a need previously found and described by Hieb and Graham [1] in their research at the University of Louisville in the intelligent systems research lab (ISRL).

1.1. Background and Motivation

Supervisory control and data acquisition (SCADA) systems are at the heart of the critical infrastructure that includes the power grid and water treatment facilities. For a variety of reasons industrial control systems (ICS) have depended on "security by obscurity," however, in recent years these systems have become increasingly vulnerable to cyber security attacks. The Stuxnet virus discovered in June 2010 [2] is an excellent recent example of the thread to ICS. Vendors, operators, and the government are now aware of the need to protect ICS from cyber based attacks [3]. It has become obvious that this method of security is no longer enough. Protecting these systems has fallen to traditional network security techniques such as firewalls and intrusion detection systems (IDS), however, ICS have differences from traditional networks that require uniquely tailored solutions, protecting field devices is an example of this.

Protection at the field device level is necessary to ensure that these systems are not compromised or damaged, while operability and performance is maintained. Upgrading these systems directly to include the necessary security features such as access control, authentication, and integrity is not an option for most of the devices. SCADA field devices have long lifetimes typically measured in decades, these older devices do not have the processing power or memory to efficiently implement these security enhancements. Also due to the large number of different field devices, implementation of such security onto all of these legacy devices would be an unfeasible task.

1.2. Designing For Security, Designing for Feasibility

When thinking about a solution for security of these legacy SCADA field devices it is important to consider the feasibility in terms of cost and performance design requirements. Designing a system that has low cost typically means the hardware will be low performance as well; this means it is important to implement software that is efficient, minimal and secure. A solution that creates a large amount of overhead and time delay may cause performance issues in these critical systems, and any practical security solution must provide security enhancements within the time constraints of these systems.

Hieb and Graham[1] have recently proposed a field device security pre-processor (FD-SPP) using a microkernel based security architecture [4] and built on low cost commercially available hardware running. The effectiveness of the FD-SPP will eventually be measured using verification test, but current work is focused on a functional prototype implemented on low cost hardware. Selection of an operating system will play role in verification, without a verified operating system software running on the FD-SPP cannot be verified.

It is obvious that typical operating systems such as Linux or Windows, which contains several million lines of code are too large to be completely verified. Exploits in these operating systems are often found by attackers and require updates to patch these exploits. It is important to select an operating system that does require this frequent updating and has a kernel that can be trusted as bug free. For this reason, further discussed in Chapter 4, a micro-kernel based operating system called OKL4 was selected to serve as the operating system for this project.

Another important consideration in the design of the FD-SPP is a low foot-print in terms of overhead and performance impact on the ICS. Role based access control and authentication have been indicated by prior research as necessary security enhancements to SCADA devices[5]. The implementation of these is often costly in terms of computation time. Using a low foot print micro-kernel will help keep computation costs down, but it is important to implement these security features as efficiently as possible. Previous work at the University of Louisville by Brad Luyster worked to implement role based access control on a low footprint microkernel[6], this work aims to create an alternative method of role based access control with a low enough foot print to be implemented in SCADA systems. In order to achieve this, this thesis explores a variation on Bloom Filters (discussed in Chapter 2) to efficiently implement role based access control (RBAC) and make decisions on when to challenge ICS messages. This data structure also reduces the amount of space required by the RBAC lookup table. This data structure performs lookups in a time independent of the number of entries of the table allowing quick lookups for any number of entries. The low foot print created by this data structure in terms of storage allows it to be easily implemented on low cost hardware.

1.3. Organization

The second chapter of this Thesis provides a literature review and description of the data structure that is at the heart of the security feature implementation of this project, the Bloom filter. Chapter three describes the design of the security features for the FD-SPP using a Bloom filter as well as the required modifications needed to a SCADA communication protocol for these features to be utilized. The fourth chapter describes the implementation of a prototype field device security preprocessor (FD-SPP) for the purposes of initial evaluation of the dual Bloom filter structure. The fifth chapter describes the testing of the FD-SPP in a simple SCADA network. Conclusions and directions for future work are presented in the sixth and final chapter.

2. INTRODUCTION TO BLOOM FILTERS

A Bloom filter is a probabilistic data structure proposed by Burton H. Bloom in 1970. In the original paper[7], he proposed an alternative hash-coding technique which traded of a small amount of allowable error for a performance increase in both time and space. This hash-coding technique is now known as a Bloom Filter and is commonly used in a wide variety of applications.

A Bloom filter is used to determine whether or not a particular item is a member of a given set. When the Bloom filter is queried with respect to a given member, the Bloom filter always returns true if the member of the set, however if the queried value is not a member of the set, the Bloom filter will not always return false. The rate at which the Bloom filter returns true for elements not in the set is known as the false positive rate of the structure. A theoretical Bloom Filter does not have any false negatives.

2.1. Properties of Bloom Filters

The basic structure of a Bloom filter consists of a bit array of length *m*. All of the bits of this array are initially set to 0, which represents an empty Bloom filter. In order to add an item to the Bloom filter, the item is passed through *k* hash functions, each producing a different value, a_k $(0 \leq a_k \leq m-1)$, which represents one of the 1-m bits of the Bloom filter such that $a_1 \neq a_2 \neq \ldots \neq a_{k-1} \neq a_k$. The bit corresponding to each of these values is then set to 1. To check an entry, the entry is passed through the same *k* hash functions to produce a list of bit addresses. Each of the bit addresses are then checked in the filter's bit array; if all addresses have a 1, then the entry is said to be a member of the set represented by the Bloom filter [7]. Because different items may have bit address collisions false positives are possible, but unlikely, when an item not in the set has a hash address that has been set to 1 by adding multiple members to the filter. Because given entries are selected based on whether the addresses deemed by each of the hash functions given the particular entry are set to 1.

The time of the insertion is depends only on the *k* hash functions and is independent of both the size in bits of the filter, *m*, and the number of elements inserted, *n*. The insertion takes $O(k)$ time for each item [7].

Shown in Figure 2.1.1 are two elements A and B being added to the Bloom filter.

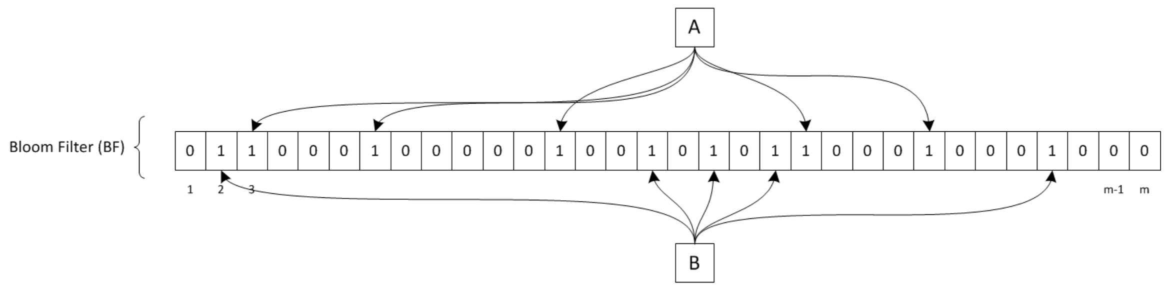


FIGURE 2.1.1 - Adding elements A and B to a Bloom Filter

To be added to the Bloom filter, the elements A and B are each passed through five hash functions (k=5) to produce five bit address in the Bloom filter. These bit values are then all set to 1. Additional items are inserted into the Bloom filter in the same way. Given a filter, checking if A and B are member is done similar to insertion; the values are passed once again through the hash functions to produce five bit address which are each checked in the filter; if every bit address is equal to 1, then the item is said to be in the set represented by the Bloom filter. The time of a membership check is also only dependent on the *k* hash functions and is independent of both the size in bits of the filter, *m*, and the number of elements stored in the filter, *n*. The membership check also takes $O(k)$ time [7]. Figure 2.1.2 shows the checking of two new elements, C and D are in the filter. Neither C nor D is a member of the set that is represented by the filter BF. D is determined not to be in the set, but C is a false positive.

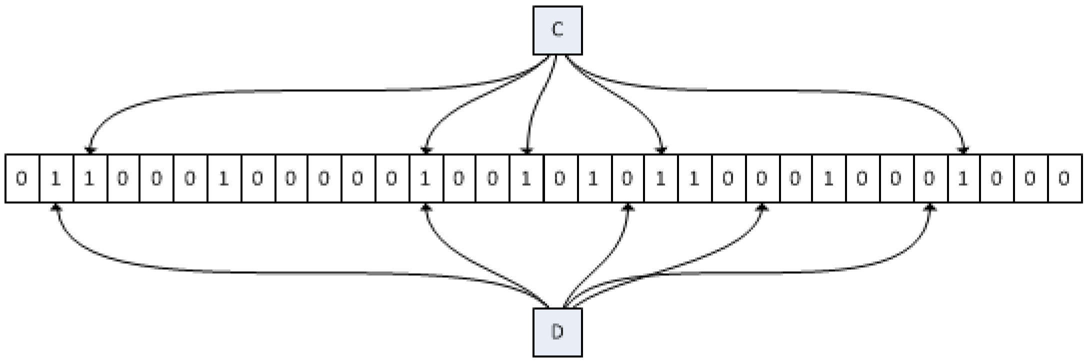


FIGURE 2.1.2 - Determining if C and D are in the set

In order to check if each of these values are in the filter, they are each passed through the five hash functions to generate corresponding bit addresses for C and D. When checking D, it is apparent that D is not a member of the set because not all the bit addresses contain 1's. According ti the filter, C is a member of the set because all the values from its hash functions are 1. Since C was not added to the set and it is not a true member of the set, this is a false positive. That Bloom filters have false positive rates is one of their limitations.

Since each of the hash functions should produce the address bits uniformly, the odds of any particular entry being a false positive is a function of the number of bits that are set to 1 and the size of the Bloom filter. After a given number of entries "n" have been added to the Bloom filter, the probability that a particular bit address is still 0 is give in equation 2.1.1.

$$p_0 = \left(1 - \frac{k}{m}\right)^n \quad [8] \tag{2.1.1}$$

Using the probability of each bit being set to 1, the false positive rate of the Bloom filter can be calculated. Using this probability, an estimate of the number of entries that are set to 1 can be calculated using equation 2.1.2.

$$Number\ of\ Bits\ set\ to\ 1 = (1 - p_0)m \tag{2.1.2}$$

This is important because it shows that the probability of an entry being falsely accepted is exponentially related to the number of bits set to 1's over the total number of bits, *m*. Since each bit is distinct, the probability of the bit at the second bit address being a 1 is slightly less than the probability the bit at the first bit address was since there is one less 1 and one less bit to select. The probability the bit at the first address is a 1 can be calculated using equation 2.1.3.

$$p_{11} = \frac{(1 - p_0)m}{m} = 1 - p_0 \tag{2.1.3}$$

The probability the bit at the second address is a 1 assuming the first address contains a 1 is calculated using equation 2.1.4.

$$p_{12} = \frac{((1 - p_0)m - 1)}{m - 1} \tag{2.1.4}$$

This trend continues for additional bit addresses for all of the *k* hash functions and the probability of the *ith* address containing a 1 assuming all the addresses *1* through *i-1* contain a 1 can be calculated using equation 2.1.5.

$$p_{1i} = \frac{((1 - p_0)m - (i - 1))}{m - (i - 1)} \tag{2.1.5}$$

For a given filter the false positive rate of the filter can be calculated as simply the product of all the probabilities of each of the bit addresses defined by the *k* hash functions. The formula to calculate the false positive rate (pb) for any given Bloom filter is shown in equation 2.1.6.

$$FP_{bf} = \prod_{i=0}^{k-1} \frac{((1 - p_0)m - i)}{m - i} \tag{2.1.6}$$

$$= \prod_{i=0}^{k-1} \frac{\left(\left(1 - \left(1 - \frac{k}{m}\right)^n\right)m - i\right)}{m - i}$$

For very large values of m such that m >> k, which is the case for Bloom filters which desire a low false positive rate, this equation can be shown to have a much simpler form as shown in equation 2.1.7.

$$FP_{bf} = \prod_{i=0}^{k-1} \frac{\left(\left(1-\left(1-\frac{k}{m}\right)^n\right)m-i\right)}{m-i} \cong \left(1-\left(1-\frac{k}{m}\right)^n\right)^k \ for\ k \ll m \ [6] \tag{2.1.7}$$

2.2. The Random Filter

Implementing a Bloom filter presents some specific challenges. Theoretically, a Bloom filter creates a data structure which will encodes a data set into a small data structure, set membership can be checked very quickly, but requires sacrificing a small amount of allowable error but when it comes to implementation. The properties of the Bloom filter discussed in section 2.1 are theoretical. The problem with implementation is with the hash functions; specifically identifying hash functions that produce *k* uniformly distributed distinct values from *k* hash functions [8]. In fact, most implementations of Bloom filters are not true Bloom filters, but a very similar data structure called a Random Filter [8].

A random filter is an adaption of the original Bloom filter proposed in 1998 by Wang, Yang and Tseug [8]. Unlike the Bloom filter, the random filter has hash functions that are completely independent of each other and are a permitted to produce the same output with any given key. Thus, in order to add an item to a random filter, the item will be passed through *k* hash functions, each producing a value a_k such that $0 \leq ak < m$ [8]. This produces a different false positive rate than the original Bloom filter, but is much easier to implement. All the other properties of the random filter are the same as the Bloom filter. Like the Bloom filter, the false positive rate of the random filter is related to the number of bits set to 1 in the random filter. After *n* entries into the random filter, the probability that a particular bit is still sit to 0 is given in equation 2.2.1.

$$p_0 = \left(1 - \frac{1}{m}\right)^{nk} \tag{2.2.1}$$

Since each of the hash functions produces a value that is independent and also uniformly distributed from 0 to *m-1*, the false positive rate of a random filter can be calculated using equation 2.2.2.

$$FP_r = \left(1 - \left(1 - \frac{1}{m}\right)^{nk}\right)^k \text{ [8]} \tag{2.2.2}$$

Besides being easier to implement, the random filter has another advantage over the Bloom filter: its false positive rate is less than that of the Bloom filter. Assuming $k > 1,\ n > 1,\ k \ll m,\ n < m$:

$$FP_r = \left(1 - \left(1 - \frac{1}{m}\right)^{nk}\right)^k < FP_{bf} = \left(1 - \left(1 - \frac{k}{m}\right)^{n}\right)^k \tag{2.2.3}$$

Proof, derived by [8] is given in Appendix A

This being shown, it is obvious that the random Filter is the better choice for implementation. Numerical implementations of these two filters also show that the random filter offers a slightly lower false positive rate than the Bloom filter [8]. More important than having a lower false positive rate is the ability to build it using independent hash functions. In computing literature, the random filter is often referred to as a Bloom filter and the false positive rates shown for Bloom filters are often those of the random filter. In order to adhere to common practice, the random filter will be referred to a Bloom filter for the remainder of this document, and all references to a Bloom filter unless specified can be assumed to be implemented as a random filter and not as originally proposed by Bloom in 1970.

2.3. Parameter Selection for building a Bloom Filter

The Bloom filter sacrifices a small false positive rate in exchange for both a small time and space constraint. For example, no matter how many elements are added to a Bloom filter it always stays the same size, and no matter how many elements are added to the filter both insertions and membership checks will always take the same amount of time. However, if too many items are added to the Bloom filter, its false positive rate will increase and eventually become 100%. So even though the space required for a Bloom filter is usually small relative to the actual data stored in it, it is important to make the Bloom filter large enough so that once all the elements are added the false positive rate is sufficiently low. In order to achieve this, the optimal values for *m* and *k* can be found given a desired false positive rate and the number of elements *n* that will be inserted into the set. As with most optimization problems, the location of the optimal values are found using a first derivative; however, the derivative of the false positive formula for the Bloom filter is one that is not particularly easy to solve. For this reason, an approximation of the false positive rate formula can be used.

$$p_r = \left(1 - \left(1 - \frac{1}{m}\right)^{nk}\right)^k \approx \left(1 - e^{\frac{-nk}{m}}\right)^k, m \gg 1 \quad [9] \qquad (2.3.1)$$

The properties of Bloom filters are well known and taking the derivative with respect to *k* and solving for *k* gives the well-known relationship between the optimal *k* and *m* and *n*.

$$k = \frac{m\, ln(2)}{n} \quad [9] \qquad (2.3.2)$$

This equation can be plugged into the approximated false positive rate and solved for *m*, which allows for the size of the Bloom filter to be determined given a particular number of entries and false positive rate. However, this value like the value for *k* cannot be used exactly because they most both be natural numbers.

$$m = -\frac{n \ln(p)}{\ln(2)^2} \tag{2.3.3}$$

This value in turn can be plugged back in to the optimal *k* equation so that the optimal number of hash functions can be found. This value as well must be turned into an natural number since it is not possible to have a non-natural number of hash functions. These formulas can still give a good estimate around the values for *m* and *k*, and then natural values can be checked into the false positive rate formula to find the one that produces the lowest rate. It is also important to note that the false positive rate returned by this function as well is merely an approximation. The actual false positive rate of the Bloom Filter requires the knowing the exact percentage of bits that are set to 1 to the size of the filter, which can only be found by building the Bloom filter. Different data sets will produce different number of 1's due to different collisions among the hash functions even for the same number of entries [10], but all of these rates will be fairly close (at least same order of magnitude) to the approximated false positive rate. Figures 2.3.1 and 2.3.2 show a comparison of false positive rates and percent difference from the theoretical false positive rate for two different data sets of size n=100, to several different Bloom filters. All Bloom filters use the same number of hash functions k=4. These two figures show that adding different values or using different hash functions each produce different false positive rates. More discussion on these differing false positive rates can be found in section 4.7.

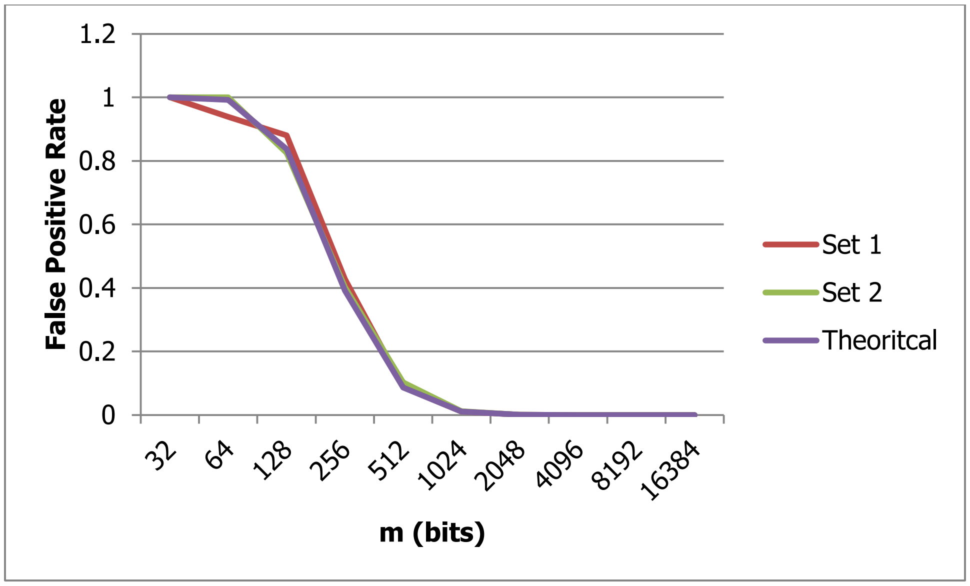


FIGURE 2.3.1 Comparison of False Positive Rate for two different data sets

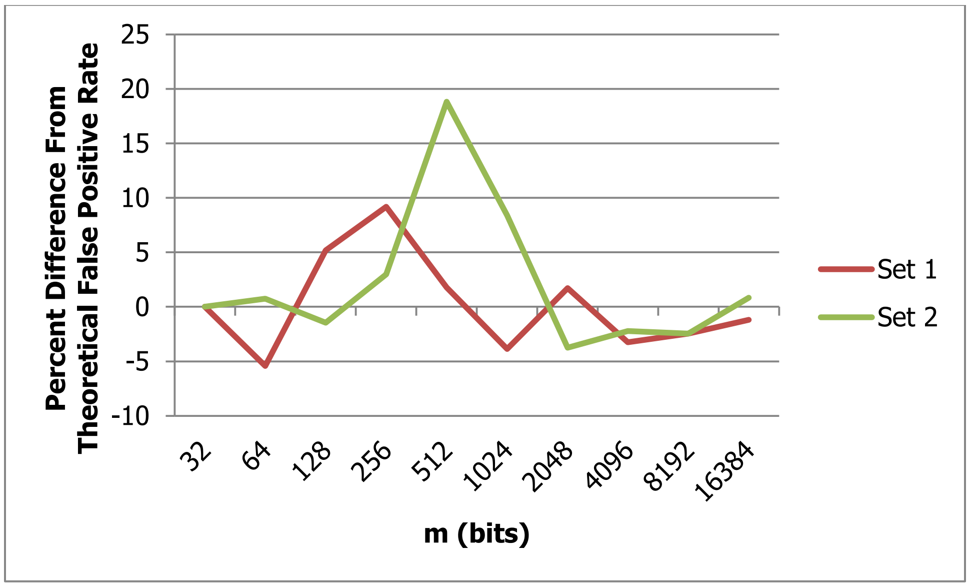


FIGURE 2.3.2 Comparison of Percent Difference from Theoretical False Positive Rate for two different data sets

2.4. Applications of Bloom Filters

Bloom filters are used in a large variety of applications, from networking applications like packet routing, to spell checkers, to helping with safe browsing. Spell checking is one of the most classic and historic use of a Bloom filter and were even used in early UNIX systems [11]. In these early computing systems, space was a scarce resource and using a Bloom filter allowed for a very compact data structure. Unlike most data structures, Bloom filters often can be stored in smaller spaces than the list could be. For example, a Bloom filter-based spell checker can store its dictionary in a much smaller space than the dictionary itself. An example Bloom filter was created for over 80,000 words, which took up over 680kb of disk space. Creating a bloom filter with a false positive rate of 2.38×10^{-8} % only takes around 520kb of space [12]. Similar to the spell checkers, Bloom filters have been proposed to store unsuitable passwords in security systems [11]. Basically the same concept that was used for spell checkers could be used to reject weak passwords [13]; add all the weak passwords to a Bloom filter, then when the user selects a password, it can be quickly rejected if it is in the weak password filter. Additionally, a false positive here and there really does not matter in this application since it will merely require the user to select a different password. This means a large dataset can be highly compressed using a Bloom Filter-based approach for this application. Also, the speed of a Bloom filter for any large dataset will mostly likely be faster than any look up for these large dictionaries.

More recently, networking applications have begun to use Bloom filters for a wide variety of topics. These applications include collaborating in overlay and peer-to-peer networks, resource routing, packet routing, measurement, and more [11]. Many network uses of Bloom filters involve the reduction of network traffic. For example, assume User A wants to send a large number of files to User B, but User B already has some of these files. Since sending all the files would waste network traffic, Bloom filters can be used to determine which files to send. Both user A and user B create Bloom filters for their file list, and user B sends their Bloom filter to user A. user A can then easily find the intersection of the two Bloom filters by performing a bitwise AND. Entries that are not in this new intersection filter are the files that need to be sent to User B. Because the Bloom filter may have false positives, not all the files in the original list may be sent. However, if this setting is used a distributed peer to peer system where the user is getting files from multiple agents to increase download time, the redundancy of the multiple users should allow for all the files to be transferred in most cases [11].

Google Chrome also uses Bloom filters in its safe surf features. According to Google®, Chrome downloads a list of sites that have been known to contain malware or are known to engage in phishing.

> "To save space and to avoid giving out URLs to malware and phishing websites, the lists contain enough information in most cases to verify that if a site is phishing or malware, but does not contain enough information to

definitively say if the site phishing or malware. If the URL of the site you're on matches anything in the list, your browser will contact Google’s servers for more information to make a decision. Your browser sends information that does not let Google uniquely determine what site you are visiting (for the technically savvy, the first 32 bits of a SHA-256 hash of the URL is sent). If your computer then decides that you’re visiting a risky site, it can warn you about it.” – Google Chrome Help [14]

Looking into the source of the chromium.org project [15], which is used for both the Google Chrome web browser and the Chromium OS, reveals the use of Bloom filters for the client side safe search check. In order to perform a site check, the requested URL is stripped down to its base and checked into a Bloom filter which contains a list of all the known URL’s which are associated with malware and phishing sites. If the checked site is in the Bloom filter, then a hash is sent to Google’s safe browsing service to verify that the site is indeed a known malicious site and not a false positive [14], [15]. The use of a Bloom filter serves multiple purposes for the Google Chrome safe search. For one, it is much smaller and faster than any type of lookup table that Google could provide with Chrome. Secondly, it allows checks to be performed client side, without distributing Google’s list of malicious URLs since there is no way to get the elements of the set back out of a Bloom filter. The speed at which the Bloom filter can make checks also makes these checks unnoticed to the user since network queries are only needed when a malicious site is visited or the occasional false positive. Avoiding these network queries improves the speed for the general user as well as reducing the load on Google's servers. This novel application of Bloom filter's shows how useful these data structures can be in situations where a small false positive rate can be traded-off for space and time.

2.5. A Bloom Filter for Role Based Access Control

Role based access control (RBAC) makes extensive use of sets, and set membership checks. Bloom filters can be used to implement a role based access control efficiently. An implementation of role based access was previously created by Tripuitara and Carbunar that used a modified Bloom filter known as a cascading Bloom filter as the primary role based access control mechanism [16].

To use a Bloom filter for RBAC, a list of the entire set of <role, operation> pairs must first be added to the Bloom filter. Adding a <role, operation> pair to a Bloom filter is the same as adding any other data object to the Bloom filter. The role and operation are combined into a single byte array, which is then passed through the 'k'-hash functions. The bits in the Bloom filter indicated by the hashes are set to "1" and the <role, operation> pair is inserted in the Bloom filter. Checks are done in the same way as a traditional Bloom filter check, using the same hash functions to identify bit locations check the bits for 1s in the filter. Using a Bloom Filter for access control has its draw backs due to the false positive rate property of the Bloom filter however it will be argued in this paper that the tradeoff for their speed can be worth the cost.

2.6. Variations and Extensions of Bloom Filters

The structure of the Bloom filter allows new data to be added but disallows the removal of any items because removal could create false negatives. If all of the 1's associated with any of the entries into the Bloom filter were removed, collisions that it shared with other entries could be removed as well, which creates false negatives for those entries. One of the simplest extensions of the Bloom filter, known as the counting Bloom filter[9], allows for data to be removed from the filter. The difference between a Bloom filter and a counting Bloom filter is when an entry is added, each hash function output corresponds to a counter instead of a single bit. Each of the counters at the positions selected by the hash functions is incremented during an insert, and decremented during a removal [9]. Also, though this certainly helps the false negative problem, it does not completely eliminate it, since these counters must have a finite maximum and will eventually become full. For example, if a counting Bloom filter has counters that go from 0-3, once a counter has four entries, the value of the counter will be 3. When three of the values are removed, the fourth will become a false negative. However, if the counter is large enough, it is unlikely that this will occur, and for some applications this low false negative rate may be acceptable.

Several expansions on the counting Bloom filter have been made to use Bloom filters with streaming data. Generally these streaming data counting Bloom filters decrement the counters periodically or based special functions to prevent the counting Bloom filter from filling and only show recent or specific trends in the data. One such expansion called the time-decaying Bloom filter uses counters that delay exponentially, which can be used to detect items that occur frequently in the data stream [17].

3. A DUAL BLOOM FILTER STRUCTURE FOR EFFICIENT IMPLEMENTATION OF ROLE BASED ACCESS CONTROL AND CHALLENGE RESPONSE FOR A FIELD DEVICE SECURITY PRE-PROCESSOR

3.1. The Need for a SCADA Field Device Security Pre-Processor

Industrial control systems have a number of known security vulnerabilities, and a large number of legacy control systems may no security in some places at the control system level. Some recent incidents such as the 2006 hacker attack on a water treatment plant Harrisburg, PA [4] has highlighted the significance of the cyber threat created by the lack of security in some of the most critical systems to the nation [4]. Legacy devices are one of the most significant of these vulnerabilities. Due to the long life times and high replacement cost of these legacy devices it is desirable to create a bolt-on appliance that can add security to these legacy devices with minimal cost and performance impact.

3.2. Access Control and Challenge Response in SCADA networks

The Implementation of a Security Pre-processer for SCADA security requires a protocol that allows for the implementation of such security. Modbus is an open and simple protocol commonly used in SCADA networks and commonly found being used by legacy field devices [18]. Modbus, by default does not offer any type of mechanisms for role based access control or for challenge response.

The basic structure of a Modbus message includes the address of the device the packet is intended for, a function code to tell the device what to do, a series of data bytes, and error detection bytes which are determined used cyclic redundancy check (CRC) algorithms [19].

ADDRESS	**FUNCTION CODE**	**DATA**	**CRC**

FIGURE 3.2.1 - Typical elements of a Modbus message

Function codes are predefined and specified in the Modbus protocol [19]. Typical function codes include read and write coils and read and write registers. A sample Modbus exchange is shown in FIGURE 3.2.2.

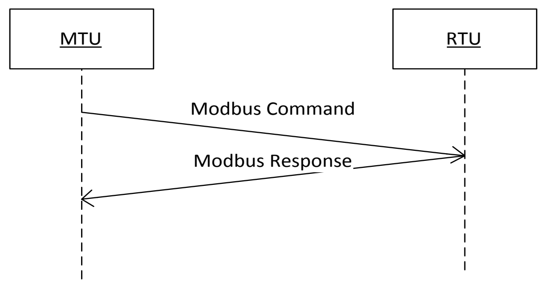


FIGURE 3.2.2 - Standard Modbus exchange

In order to allow for Modbus to support role based access control and challenge response a couple of small additions have to be made to the Modbus Protocol[4], [5]. The modified Modbus protocol will be referred to Extended-Modbus when differentiation between it and the original protocol is necessary. The first required extension to Modbus is to add the concept of a user, and a user's secret. This user needs to authenticate (initially and periodically) when communicating with the Remote Terminal Unit (RTU) or legacy field device. In order to support users the extended Modbus protocol includes a new function code, *Request Connection*, which includes a user ID in the data field to establish a connection, and allows the access control system to know which user is logging in. To verify that the user is the user specified in the login request an additional packet must be added. Extended-Modbus has a second new function code, *Challenge*. The HMI side of the system replies to this Extended-Modbus packet with a third new packet type with a new function code, *Response*. Keeping in tradition with typical Modbus protocol standards the request connection packet will be returned to the MTU upon a successful connection.

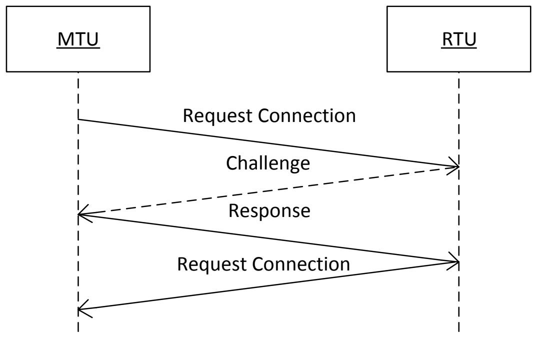


FIGURE 3.2.3 - Extended Modbus Connection Request

3.2.1. Details of the Extended Modbus Function Codes

Modbus has a large number of unused function codes which allowed the protocol to be expanded very simply. Shown in Table TABLE 3.2.1 are the function codes for the new commands that were added to the protocol.

TABLE 3.2.1 Extended Modbus Function Codes

Function Code	Description
40	Connection Request
41	Challenge
42	Challenge Response

The connection request packet was added as function code 40. This packet is responsible for sending the user id to the field device. Each user will have a unique user id which is used by the field device to identify them and look up their role and their secret. A connection request packet is always challenged by the field device. The successful completion of this challenge means that the user has successfully logged in and all packets that are challenged will be checked with that user's secret for the duration of the users session. The issue of a new login request automatically ends the previous user's session. Multiple users accessing the device at the same time is not supported at this time.

Byte	**1**	**2**	**3**	**4-5**
Data	ADDRESS	40	USER ID	CRC

FIGURE 3.2.4 - Packet Structure for Connection Request

The challenge packet is sent from the field device when a packet needs to be challenged or a user is logging in. This packet contains four 4 bytes of cryptographic nonce to be used as part of the hash for the response packet.

Byte	**1**	**2**	**3-6**	**7-8**
Data	ADDRESS	41	NONCE	CRC

FIGURE 3.2.5 - Packet Structure for Challenge Packet

The challenge response packet is sent from the MTU/HMI as a response to receiving a challenge from the field device. This packet contains a hash of the original packet, the cryptographic nonce from the challenge, and the user's secret. Construction of this packet is explained more later in Chapter IV.

3.3. Dual Bloom Filters for Modbus Role Based Access Control (RBAC)

The role based access control in this project is similar to the one mentioned previously in Section 2.5 in the fact that it uses a Bloom filter for access control by hashing <role, operation> pairs but the Bloom filter implementation and use is quite different. The RBAC for this project not only determines whether the packet is allowed, i.e. a user performing an allowed operation, but also determines whether that packet is critical and therefore requires a challenge to support integrity and authenticity. In order to achieve this extra feature the RBAC system uses dual Bloom filters. The Bloom filters have the exact same number of bits and use the exact same hash functions. When an entry is to be checked in the RBAC it only has to be passed through the k hash functions and then it can be checked in both the Bloom filters. The cost of using two Bloom filters instead of one in terms of computation time is nominal and results in an access check in O(k). The cost of using the two Bloom filters in space is double the cost of use a single Bloom filter since the Bloom filters will each take up m bits.

Both of the Bloom filters contain <role, operation> pairs, where each operation is a Modbus packet. The first Bloom filter contains all the allowable <role, operation> pairs and is responsible for determining if the packet should be allowed or rejected. The second Bloom filter determines whether the packet should be challenged. This second filter could contain either all the packets that need to be challenged or all the packets that do not, a comparison of these techniques can be found in section 3.5. For now assume the challenge Bloom filter contains all the entries that are challenged. In this case the following table illustrates, whether to allow the packet, challenge the packet, or reject the packet.

TABLE 3.3.1 – Dual Bloom Filter RBAC

Access Bloom Filter	**Challenge Bloom Filter**	
Yes	Yes	Challenge
Yes	No	Allow
No	Yes	Reject
No	No	Reject

Operations that are allowed are performed immediately after the packet is received; challenged operations are performed only after a successful challenge response is received. Operations that are rejected are ignored without are type of response. Since the access Bloom filter contains all the packets that can be accepted (Set A), and the challenge Bloom filter contains all the packets that can be accepted and must be challenged (Set C) if the packet is not in the access Bloom filter it will not be in the challenge Bloom filter.

$$C \subset A \ \therefore if\ x \notin A\ then\ x \notin C \tag{3.3.1}$$

Figure 3.3.1 shows how the dual Bloom filter RBAC is used to process Modbus messages, and determine whether to challenge, allow, or reject each packet.

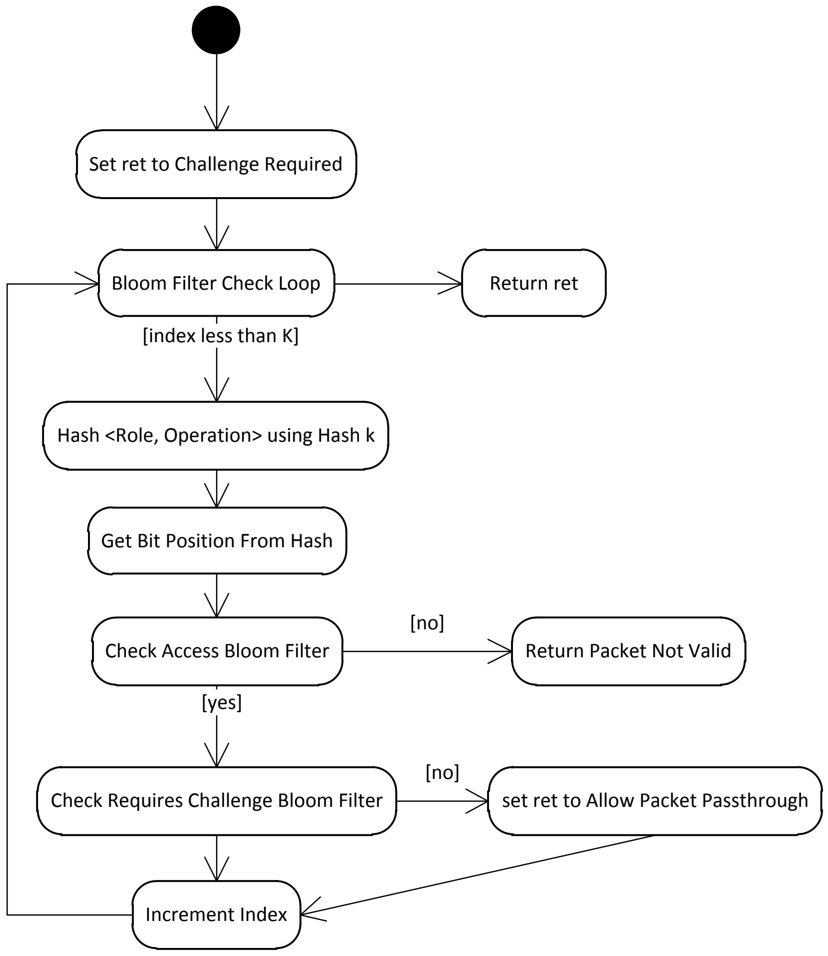

Set ret to Challenge Required
Bloom Filter Check Loop
Return ret
[index less than K]
Hash <Role, Operation> using Hash k
Get Bit Position From Hash
Check Access Bloom Filter
[no]
Return Packet Not Valid
[yes]
Check Requires Challenge Bloom Filter
[no]
set ret to Allow Packet Passthrough
Increment Index

FIGURE 3.3.1 - Dual Bloom Filter RBAC Check

3.4. Creation and Analysis of an Example Dual Bloom Filter

The first step to creating a Bloom filter is determining the number of bits required *m*, and the number of hash functions required *k*. These two numbers can be based on the estimated number of objects that will be added to the bloom filter and a desired false positive rate. From the properties of the Bloom filters these two values can be easily calculated using the well-known formulas described in section 2.2 using the estimated number of entries that will be added, *n*, and the desired false positive rate, *p*. For this example, let n = 100, and p = .01. Using the following formula an optimal value for *m* can be derived.

$$m = -\frac{n \ln(p)}{\ln(2)^2} = -\frac{100 \ln(.01)}{\ln(2)^2} = 958.53 \tag{3.4.1}$$

In order to make the code simpler and more efficient, a integer that is a power of 2 should be selected. This allows for x bits to be selected for each hash function where m = 2^x, and makes having uniform bit selection from the hash functions easier to achieve. For this example let m = 1024, this is the power of two above the *m* necessary for the false positive rate and will offer a lower false positive rate then the desired *p* of .01, while letting m = 512, the next closest power, would have a false positive rate of greater than .01 when 100 entries were added. Now that *m* is selected and optimal *k* can be selected using *m* and *n* as follows.

$$k = \frac{m\, ln(2)}{n} = \frac{1024\, ln(2)}{100} = 7.0979 \tag{3.4.2}$$

The optimal value of k is 7.0979, but it is not possible to have only a part of a hash function or select part of a bit, therefore an integer value must be selected for k. Since 7.0979 is between 7 or 8, we can test both of these to see which one is likely to have a lower false positive rate using our value of "m" and "n".

$$p = \left(1 - \left(1 - \frac{1}{m}\right)^{nk}\right)^{k} \tag{3.4.3}$$

$$p = \left(1 - \left(1 - \frac{1}{1024}\right)^{100*7}\right)^{7} = 0.0073198$$

$$p = \left(1 - \left(1 - \frac{1}{1024}\right)^{100*8}\right)^{8} = 0.0074848$$

The value of k should be selected as 7 since this produces a lower false positive rate, also using less hash functions will improve the speed of both Bloom filter entry additions, and access checks.

Now that the parameters have been selected entries can be added, to the dual Bloom filter structure. The following <role, operation> pairs were added to the dual Bloom filters. The first number is the role id, second is the hex data that represents the operation to be performed. The last value is a yes or no representing whether the packet should be challenged and therefore added to both filters.

TABLE 3.4.1 Example <Role, Modbus Packet> Entries

Role	**Operation**	**Needs Challenge**
1	01020000000C780F	No
2	01020000000C780F	No

1	010F0000000401003E96	Yes
1	010F000000040101FF56	Yes
1	010F000000040102BF57	Yes
1	010F0000000401037E97	Yes
1	010F0000000401043F55	Yes
1	010F000000040105FE95	Yes
1	010F000000040106BE94	Yes
1	010F0000000401077F54	Yes
1	010F0000000401083F50	Yes
1	010F000000040109FE90	Yes
1	010F00000004010ABE91	Yes
1	010F00000004010B7F51	Yes
1	010F00000004010C3E93	Yes
1	010F00000004010DFF53	Yes
1	010F00000004010EBF52	Yes
1	010F00000004010F7E92	Yes

After creating a Bloom filter is it's possible to more accurately determine its false positive rate. We have added 18 entries therefore n = 18. Using the previously derived equations for p given m, n, and k, the false positive rate of the Bloom filter can be shown as:

$$p = \left(1 - \left(1 - \frac{1}{1024}\right)^{18*7}\right)^{7} = 2.7975 * 10^{-7} \tag{3.4.4}$$

However this is merely the theoretical false positive rate of the approximation of the Bloom filter after 18 entries. Once we have actually created the Bloom filter the number of ones in the Bloom filter can be used to calculate the actual false positive rate of this specific Bloom filter. This is done by counting the number of bits in this Bloom filter that have been set to 1. This Bloom Filter is shown in the following byte array of hex values:

{ 0xc0, 0x41, 0x00, 0x04, 0x20, 0x00, 0x20, 0x28, 0x28, 0x04, 0x80, 0x14, 0x00, 0x01, 0x00, 0x00, 0x00, 0x00, 0x92, 0x08, 0x0a, 0x80, 0x00, 0x20, 0x04, 0x08, 0x02, 0x44, 0x22, 0x08, 0x08, 0x04, 0x00, 0x08, 0x05, 0x04, 0x00, 0x80, 0x08, 0x04, 0x04, 0x04, 0x00, 0x20, 0x00, 0x01, 0x81, 0x40, 0x02, 0x00, 0x04, 0x10, 0x20, 0x00, 0x20, 0x00, 0x10, 0x00, 0x40, 0x08, 0x12, 0x00, 0x29, 0x18, 0x00, 0x08, 0x0b, 0x01, 0x00, 0x00, 0x01, 0x20, 0x00, 0x20, 0x00, 0x11, 0x00, 0x20, 0x88, 0x00, 0x00, 0x04, 0x00, 0x00, 0x24, 0x60, 0x08, 0x06, 0x40, 0x00, 0x09, 0x08, 0x0a, 0x04, 0x20, 0x0c, 0x02, 0x00, 0x00, 0x00, 0x00, 0x00, 0x52, 0x00, 0x08, 0x01, 0x4a, 0x00, 0x01, 0x00, 0x08, 0x48, 0x00, 0x10, 0x00, 0x00, 0x80, 0x00, 0x42, 0x06, 0x00, 0x04, 0x41, 0x04, 0x01, 0x00, 0x04, 0x00 }

This Bloom filter of 1024 bits has 119 bits set to the value of 1. Therefore, the probability of any single bit being a 1 is simply 119/1024.

$$p = (\frac{119}{1024})^{7} = 2.8624 * 10^{-7} \tag{3.4.5}$$

since the number of bits in the second Bloom filter is known as well, the non-challenged false positive rate can be calculated as well. The non-challenged false positive rate is the odds of an attacker performing an operation that should be restricted without knowing the authentication secret. In order for a value not to be challenged it must be in the first Bloom filter but not in the second Bloom filter. Since all the bits in the second Bloom filter are also one in the first Bloom filter the only way for the entry not to be challenged is for the entry to require at least one of the bits that are only in the first Bloom filter. For this example the second bloom filter has 106 bits set to 1. This means that there are 13 bits not shared between the Bloom filters. Therefore the non-challenged false positive rate can be calculated as:

$$p = \left(\frac{13}{1024}\right)\left(\frac{119}{1024}\right)^6 = 8.1270 * 10^{-8} \tag{3.4.6}$$

As mentioned previously, the second filter could be designed to store the entries that do not need to be challenged as opposed to the entries that do need to be challenged. In this design, in order for an entry not to be challenged it must be in the second Bloom filter. If a value is in the second Bloom filter then it will also be in the first Bloom filter since all the ones in the second Bloom filter will be set in the first Bloom filter. Therefore this value can be calculated in the same way that the false positive rate for the original Bloom filter was. The theoretical value can be calculated using the false positive formula, using n=2 since there were two entries that were not challenged:

$$p = \left(1 - \left(1 - \frac{1}{1024}\right)^{18*2}\right)^{2} = 8.5411 * 10^{-14} \tag{3.4.7}$$

Also since this Bloom filter was actually created the number of ones can be counted and the actual false positive rate can be calculated.

$$p = \left(\frac{14}{1024}\right)^{7} = 8.9289 \times 10^{-14} \tag{3.4.8}$$

For this example the second Bloom filter used for challenge response has a much lower false positive rate than the first and therefore should be used if this data is used in an actual system.

3.5. Comparison of the Two Challenge Response Implementations

In the example above there was a difference in the two false positive rates of several orders of magnitude. This section will discuss and prove whether or not this is true for all data sets or if there is a cut off where the other implementation produces a smaller false positive rate. Start by expanding the formula to include an additional variable c; this will represent the number of entries that need to be challenged. In the second example above, operations that were in the Access

Bloom filter but were not in the challenge response Bloom filter were challenged. In this case, the false positive rate of non-challenged packets is the false positive rate of the challenge response Bloom Filter. This Bloom filter will have n - c entries therefore:

$$p(n-c) \approx \left(1 - e^{-\frac{k}{m}*(n-c)}\right)^{k} \tag{3.5.1}$$

For this example the value of k will be assumed to be the optimal value based on the access Bloom filter therefore:

$$k = \frac{m\, ln(2)}{n} \tag{3.5.2}$$

This allows the formula for p to be written as

$$p(n-c) \approx \left(1 - 2^{\frac{-n+c}{n}}\right)^{\frac{\ln(2)m}{n}} \tag{3.5.3}$$

The second Bloom filter strategy for deciding challenges, the first one implemented in the prior example, involves both Bloom filters. The first Bloom filter of containing n entries and the second Bloom filter containing c entries. In order for a non-challenged false positive to occur at least one bit must be in the first Bloom filter that is not in the second Bloom filter. In order to calculate this it is necessary to determine the number of bits that are one in the first Bloom filter that are not in the second Bloom filter.

First assume that c identical entries have been added to both Bloom filters. This means there are *n - c* entries to add to the first Bloom filter. The theoretical number of ones that is added to this filter when c entries already exist will be the number of ones in one filter but not in the other. The number of ones after n entries will be equal to:

$$ones(n) = m\left(1 - e^{-\frac{kn}{m}}\right) \tag{3.5.4}$$

And after c entries it will be:

$$ones(c) = m\left(1 - e^{-\frac{kc}{m}}\right) \tag{3.5.5}$$

Therefore formula 3.5.6 will calculate the number of ones not shared by the Bloom filters

$$\Delta ones = m\left(1 - e^{-\frac{kn}{m}}\right) - m\left(1 - e^{-\frac{kc}{m}}\right) \tag{3.5.6}$$

$$= m * \left(2^{-\frac{c}{n}} - \frac{1}{2}\right)$$

Using this, the false positive rate of non-challenged entries can be calculated by using the optimal value of k using the following formula:

$$p = \left(\frac{1}{2}\right)^{\left(\frac{m*\ln(2)}{n}-1\right)} * \frac{m * \left(2^{-\frac{c}{n}} - \frac{1}{2}\right)}{m} \tag{3.5.7}$$

$$= \left(\frac{1}{2}\right)^{\left(\frac{m*\ln(2)}{n}-1\right)} * \left(2^{-\frac{c}{n}} - \frac{1}{2}\right)$$

This formula uses the theoretical false positive rate for a bloom filter using k-1 hash functions and size m, with n entries, multiplied by the difference in the number of ones calculated previously over m.

To simplify the comparison of the two formulas the value of m that is based on n and the desired false positive rate can be used. This leaves the two formulas:

Non-Challenges Stored in Bloom Filter

$$p_{nc} = \left(1 - 2^{\frac{-n+c}{n}}\right)^{-\frac{\ln(p)}{\ln(2)}} \tag{3.5.8}$$

Challenges Stored in Bloom Filter

$$p_c = p\left(-1 + 2^{-\frac{-n+c}{n}}\right) \tag{3.5.9}$$

From these two formulas it can be noted that the false positive rate is dependent on the ratio between c and n, let this ratio be equal to r.

Non-Challenges Stored in Bloom Filter

$$p_{nc} = (1 - 2^{r-1})^{-\frac{\ln(p)}{\ln(2)}} = p^{-\frac{\ln(1-2^{r-1})}{\ln(2)}} \tag{3.5.10}$$

Challenges Stored in Bloom Filter

$$p_c = p(-1 + 2^{1-r})) \tag{3.5.11}$$

These two equations will always intersect minimally at r=0 and r=1, which produce false positive rates of p and 0 respectively. For a realistic example, p will be selected to be very small, p << 1, so as p gets smaller and smaller the false positive rates of the non-challenge method decreases exponentially while the challenge Bloom filter only decreases linearly. This means that the non-challenge Bloom filter will have a smaller false positive rate for any small value of p. Figure 3.5.1 is a plot showing that when p = .001, the non-challenge Bloom filter method produces a lower false positive rate for all r values.

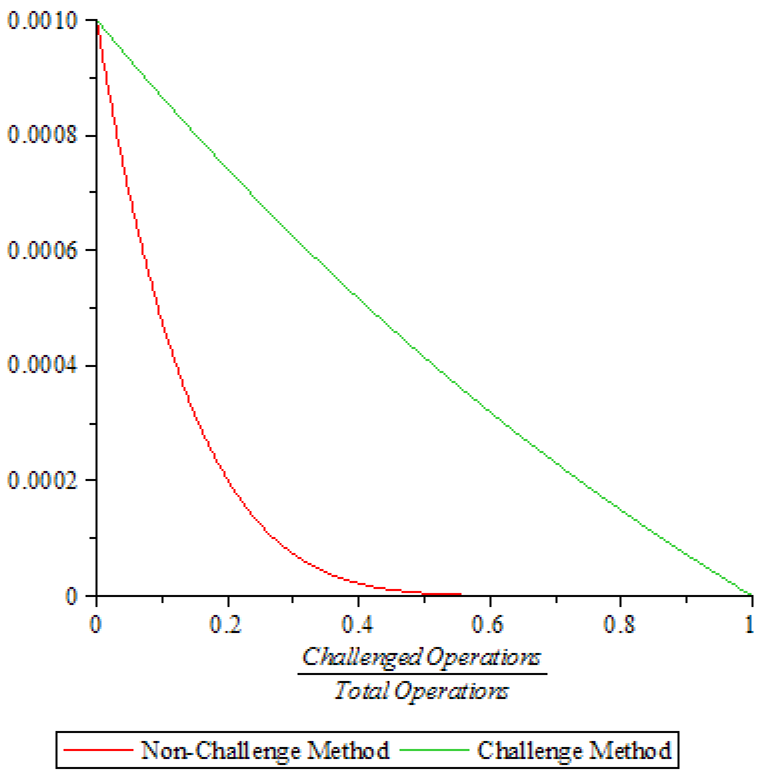


FIGURE 3.5.1 Comparison of false positive rates of Challenge and Non-Challenge Implementations with p = .001

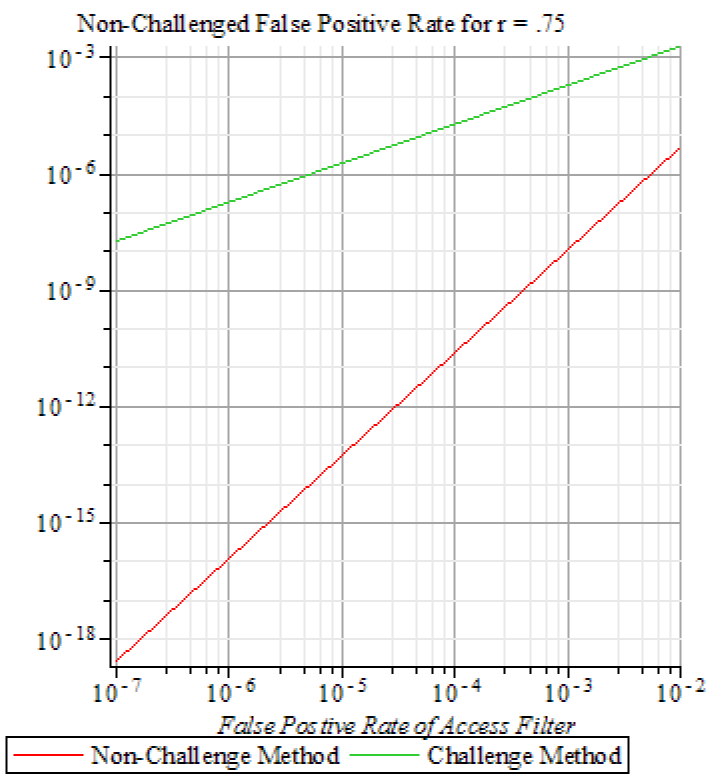


FIGURE 3.5.2 Comparison of false positive rates of Challenge and Non-Challenge Implementations with r = .75

Shown in figure 3.5.2 is a comparison of the non-challenged false positive rates for the two implementation methods for a ratio between challenged operations and total operations of .75. From here it can be seen that as the false positive rate of the access Bloom filter decreases the filter containing non-challenged operations decreases very rapidly, while the filter containing challenged operations only decreases linearly and is much greater than the non-challenged method.

Therefore if we select the non-challenge method for the second Bloom filter the false positive rate, which is the odds of a potential attack packet making it through the Bloom filter, is equal to

$$p_{nc} = (1 - 2^{r-1})^{-\frac{\ln(p)}{\ln(2)}} = p^{-\frac{\ln(1-2^{r-1})}{\ln(2)}} \tag{3.6.10}$$

where p is the desired false positive rate selected when creating the access Bloom filter and r is the ratio of challenged entries to all entries entered into the Bloom filter.

Now that the second filter contains <role, operation> pairs not to challenge, instead of pairs to challenge, the flow of the method that determines whether to allow, reject, or challenge an operation must be changed slightly. The following table, similar to the one shown previously 3.4.1 shows what to do when a pair is in or is not in each of the Bloom filters. Since the second filter still determines whether or not to challenge an operation, it is still listed as the challenge Bloom filter.

TABLE 3.5.1 Updated RBAC Responses

Access Bloom Filter	Challenge Bloom Filter	
Yes	Yes	Allow
Yes	No	Challenge
No	Yes	Reject
No	No	Reject

Once again it is not possible for an operation to be in the Challenge Bloom filter and not the Access one, since the Challenge Bloom filter contains a set of elements that is a subset of the ones in the Access filter. This change to the table forces the implementation to change slightly as well. The flow diagram for implementation of the updated RBAC policy is shown below.

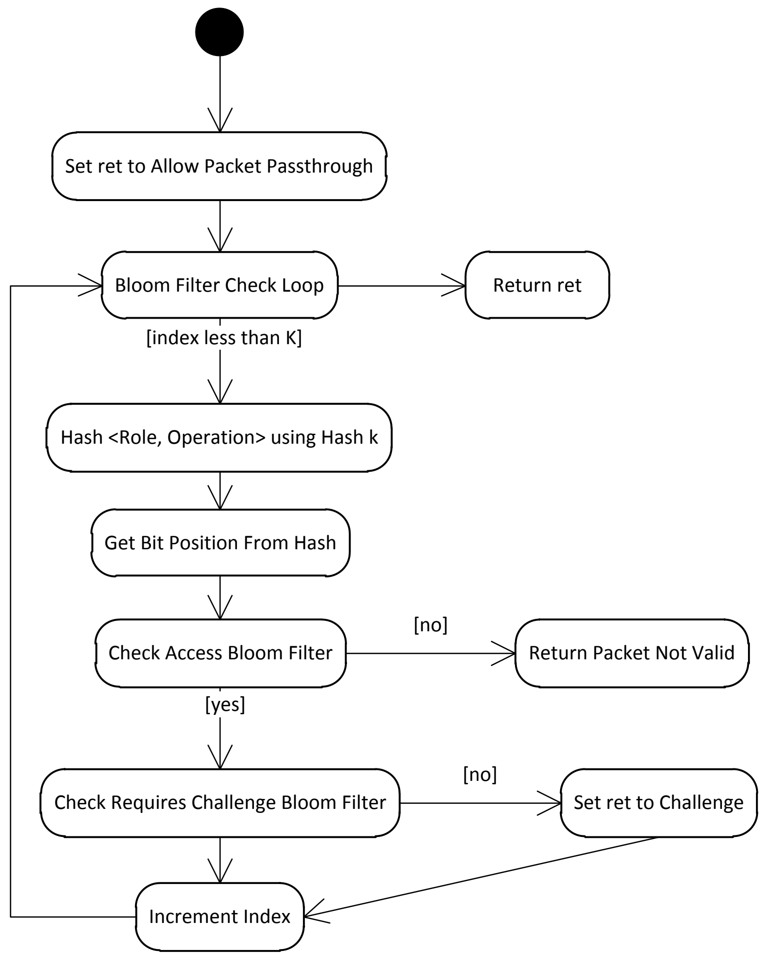

Set ret to Allow Packet Passthrough
Bloom Filter Check Loop
Return ret
[index less than K]
Hash <Role, Operation> using Hash k
Get Bit Position From Hash
Check Access Bloom Filter
[no]
Return Packet Not Valid
[yes]
Check Requires Challenge Bloom Filter
[no]
Set ret to Challenge
Increment Index

FIGURE 3.5.3 Updated Flow Diagram for RBAC check

In the next chapter the implementation and testing of this RBAC and challenge response policy for SCADA security will be discussed.

4. IMPLEMENTATION OF THE FIELD DEVICE SECURITY PRE-PROCESSOR FOR SCADA USING DUAL BLOOM FILTERS FOR ACCESS CONTROL

4.1. The Field Device Security Pre-Processor

The field device security preprocessor is a device being developed at the University of Louisville for the National Institute for Hometown Security (NIHS). It expands upon previously developed technologies developed at the University of Louisville for hardening legacy remote terminal units against cyber-attacks for HIHS [1], [4], [20]**.** The device will act as an add-on to existing legacy remote terminal units and can be added to existing industrial control systems with minimal hardware and software changes. Adding the field device security preprocessor (FD-SPP) to an existing unit such as a legacy remote terminal unit is performed by disconnecting the existing network connection from the remote terminal unit and connecting it instead to the master side of the field device, and then creating a connection between the remote terminal unit(RTU) and the slave side of the field device. This way the all the traffic that normally would be received by the RTU is now received instead by the FD-SPP. The FD-SPP adds support for new security features such as authentication, and role based access control to the existing terminal units, however the FD-SPP requires the master side of the system to support this mechanism as well. This can be done via a software upgrade on the HMI/MTU or with another hardware device similar to the FD-SPP.

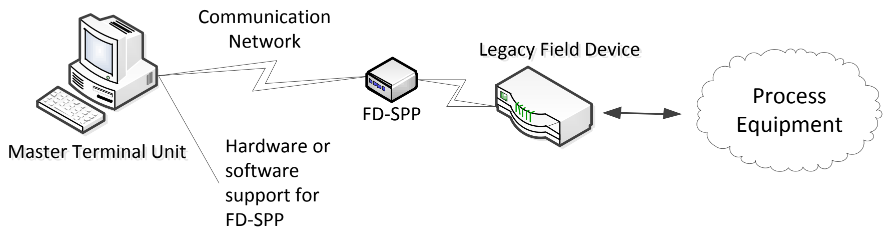


FIGURE 4.1.1 Placement of the FD-SPP in a simple SCADA system

Traditional industrial control systems or SCADA systems, do not provide any authentication or authorization[5], the FD-SPP will add these features to existing control systems. As described in the previous chapter these techniques will be implemented using the dual Bloom filter access control, which challenge response, and the extended Modbus protocol.

4.2. <u>The Microkernel Architecture for the FD-SPP</u>

The microkernel architecture described by Hieb and Graham [1], [4] isolates software components into isolated address spaces, separating networking device drivers, security enforcing software components, and field equipment interfaces and drivers. The microkernel enforces these isolations as well as provides limited communication channels between particular modules. The module containing the networking device drivers and the module containing the field device drivers or resources should be completely isolated from each other. These two modules should only communicate to the security modules creating a barrier between the input and output of the security device[1]. This barrier prevents attackers from leveraging an error in the communication driver to affect the field device [4]. The operating system possesses a critical role in security enforcement; a microkernel is used by the architecture to minimize the amount of code in the trusted computing base. The microkernel provides only the minimum necessary operations including memory abstraction (an address space), an execution abstraction (threads), and inter process communication (IPC) [1]. The microkernel must provide strong assurance that interaction between two address spaces is not possible, and that IPC is limited to specified threads only. [1]

The FD-SPP uses this microkernel based architecture to insure security and reliability, in order to achieve this architecture the OKL4 microkernel was selected. OKL4 is a member of the L4 family of operating systems. The L4 operating systems are second generation microkernel operating systems. OKL4 like all L4 kernels only provides the most basic essentials required for an operating system[21] and leaves the remainder of the design up to the developers. Another L4 operating system seL4 has been formally verified[22]. This means that there is a machine checked mathematical proof that the implementation in code of seL4 matches the code specifications. It also means that seL4 code is proven to be free from common programming errors such as buffer overflows and null pointer accesses[22]. Although this doesn't imply security, it is a starting point for building secure software. Additionally this allows the software above the kernel to be verified since the kernel is verified. OKL4 is in the same family of operating systems as seL4, the verified distribution of L4, it should be therefore a small step to make a port from the OKL4 to the verified kernel.

OKL4 allows the division of software above the kernel layer into cells. These cells each have their own virtual memory and are segregated from the other cells [21]. A buffer overflow in cell A cannot affect cell B. Communication between the cells is provided via IPC in the kernel layer. Additionally, like all microkernels, device driver level code is in the user application layer. This means that cells can have access to the hardware. However to prevent this from being a security issue only one cell can have write access to any given register. Cells can however share read access to a register [20]. All memory operations including access to hardware registers are performed using a virtual memory system provided by the OKL4 microkernel, the system calls required to access this virtual memory system were developed by Brad Luyster [20].

4.3. The OKL4 Cell Structure used for the FD-SPP

The FD-SPP software components are broken up into three cells: master Modbus communication cell, packet filtering security cell, and the slave Modbus communication cell. Each cell has its own responsibilities, and its own memory space. This design allows communication handling code and security code to be run in completely separate memory spaces, and creates a separation between the code communicating with the master and slave devices. Shown below is the model of the OKL4 cells for the FD-SPP.

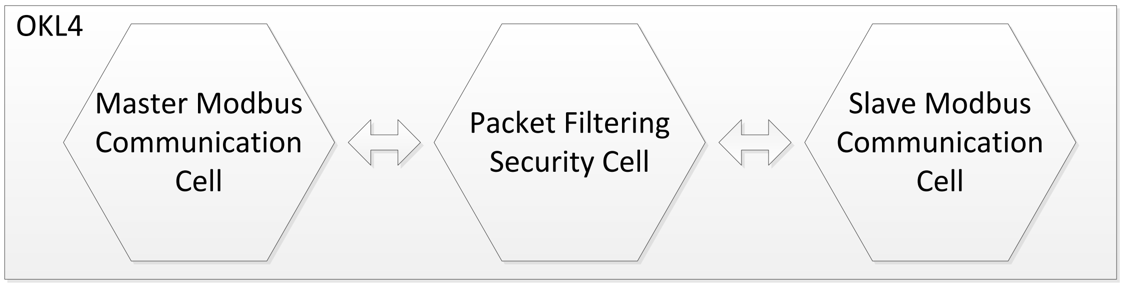

OKL4
Master Modbus Communication Cell
Packet Filtering Security Cell
Slave Modbus Communication Cell

FIGURE 4.3.1 OKL4 cells for the FD-SPP

The master Modbus communication cell is responsible for receiving and sending data to the Modbus master which in this case is the virtual Modbus serial device. This cell has a driver for one of the UARTs on the device, which allows it to communicate over RS-232 with the master. The master then scans the data received by the UART until it finds a valid Modbus packet. This packet is then sent to the packet filtering security cell via IPC. Since the master Modbus communication cell only sends valid packets to the packet filtering security cell, the packet filtering security cell is protected from attacks using invalid packets.

The packet filtering security cell can also send packets to the master Modbus communication cell which is responsible for forwarding these packets to the master.

The slave Modbus communication cell has very similar operations with the master Modbus communication cell. Additionally the code for the two cells is almost identical. The slave Modbus communication cell sends and receives data via RS-232 with the slave (RTU). When it receives data it automatically forwards the data to the security cell which then passes it through to the master. Therefore there is no filtering of data for packets being transferred from the slave to the master. Additionally the slave Modbus communication cell will get packets from the packet filtering security cell which it will be responsible for transmitting to the RTU.

The packet filtering security cell has several functions. Its most simple function is forwarding packets from the slave to the master. When the cell receives and IPC call from the slave it simply forwards this IPC call to the master. The primary task of the security cell is extracting the added security pieces out of the extended Modbus packets, and only sending packets that are verified to the Slave. This involves several tasks: creating and managing user connections, creating and validating challenge response packets, and performing role based access control for each packet and each user as defined in the previous chapter.

The OKL4 software was divided into three cells, which all have their own memory space which they can read/write and none of the other cells can access. In order for these cells to send data to each other they use IPC calls, provided by the microkernel. All IPC communication channels must be defined at compile time and cannot be changed during run time. This means that if there are no IPC communications channels defined between to cells they cannot directly communicate [20]. Shown below is the layout of the three cells as was shown previously, however now the threads and IPC calls of the system have been added.

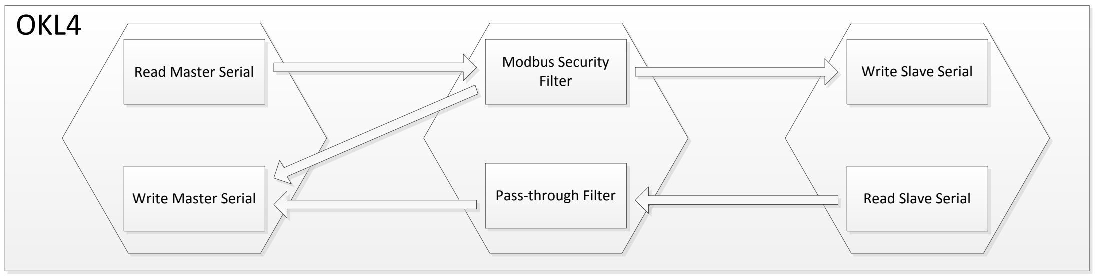


FIGURE 4.3.2 Cell Communication Flow

The most important concept is that the master communication cell and slave communication cell do not have any communication between them. This means that in order for a packet to be sent to the RTU it must be passed through two layers of IPC and two layers of validation checks. Do to the nature of OKL4 a bug in one of these layers cannot exploit the next layer this makes it very difficult for an attack packet to propagate through the OKL4 security device.

4.4. Software Design of OKL4 Cells

The slave Modbus communication cell and the master Modbus communication cell share much of the same source code and are almost identical. They both perform the operation of reading and writing to an RS-232 communication port. The STUART (Standard UART) is used for the master communications, and the BTUART (Bluetooth UART) is used for the slave communications. The BTUART although it is capable of being used as a Bluetooth device port is being used in the same way the STUART is as a standard RS-232 communications port [23]. Both the master and slave communication cells are broken up into two threads; One thread for polling the UART and forming Modbus packets to send to the filtering cell and a second for writing packets to the UART as they are received from the filtering cell. Shown below is the flow diagram for the read data from UART thread.

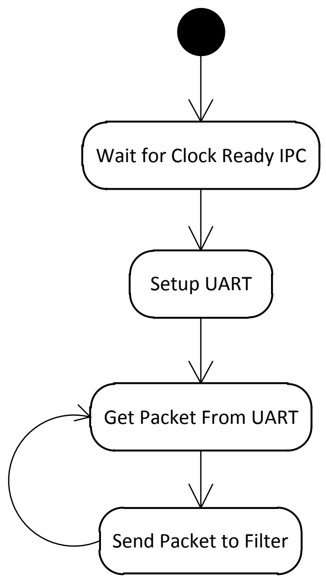


FIGURE 4.4.1 UART read flow chart

The first block on the diagram is a wait for the clocks to be set up for the UART. This is because only one cell is allowed write access to enable the clocks for the UART. The master communications cell sets up the UART clocks for itself and the slave communication cell. Once the thread is notified that the clocks are set up, it sets up its initialization parameters for the UART such as baud rate and flow control. After this it begins the process of gathering data from the UART and sending it to the filter cell. Each communication cell begins filling a buffer with data and waiting for the 3.5 character times between bytes, which represents the end of packet in the Modbus standard [24]. Once this 3.5 character times occurs the communication cell performs a Modbus CRC check on the packet. If the CRC check matches the packet CRC then this packet is valid and therefore sent to the filter, otherwise the beginning and ending point of the packet are marked in the buffer and the communication cell waits for the next packet. The communication cell additionally uses a circular buffer, in other words if the end of the buffer has been reached the bytes are added to the beginning of the buffer. The diagram below shows what the buffer looks like after the first packet is read in.

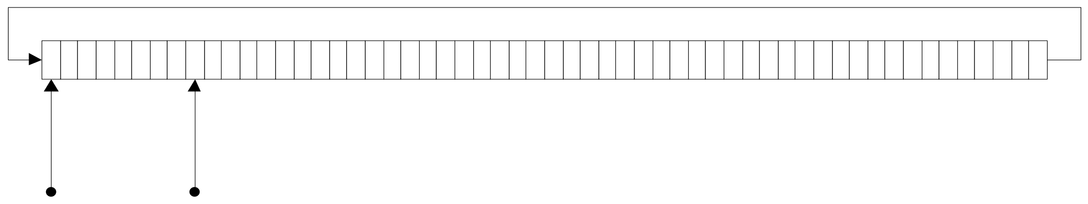


FIGURE 4.4.2 Communication Cell Circular Buffer

When the second packet is read in its data is placed after the first packet. The same CRC check is performed on this next packet to see if it is a valid Modbus packet. If so the packet is sent to the filtering cell. Whenever a valid packet is found and passed to the filtering cell, the buffer is cleared. This prevents unnecessary CRC checks with invalid data. The diagram below shows where the second packet is placed in the buffer.

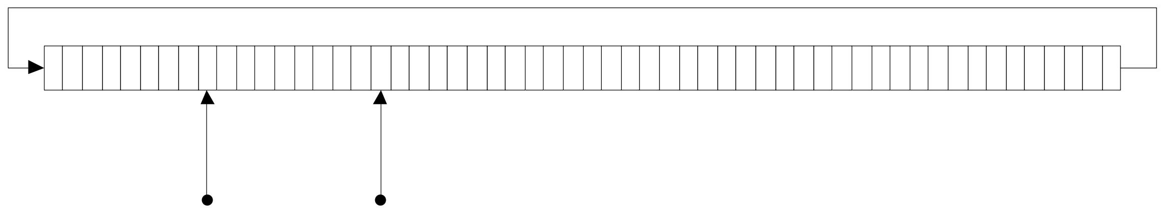


FIGURE 4.4.3 Master Cell second Modbus placement in circular buffer

If the data in the second packet is invalid, the two packets are looked at as a single packet. If this reconstructed packet is valid then it is passed on to the filtering cell. The diagram below shows the reconstructed packet.

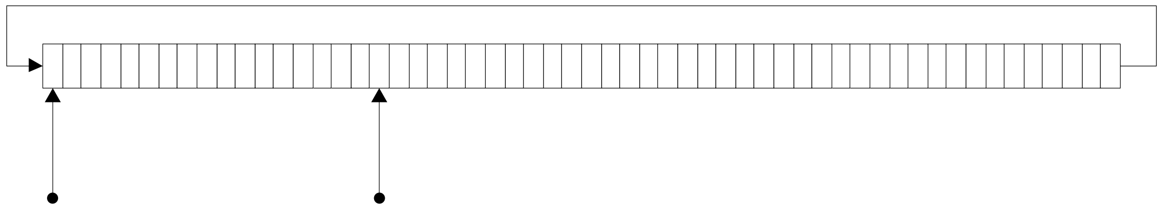


FIGURE 4.4.4 Modbus message reconstruction

If this packet is not valid then the process continues for the third packet. First the packet is checked by itself. If this fails it is checked in combination with the second packet. If this fails all three packets are combined and checked. This process is continued with a fourth, fifth or sixth packet. If a seventh packet is received the first packet is dropped so the system will never remember more than six packets. This packet reconstruction is necessary to guarantee valid Modbus packets are sent to the filtering cell from the communication cell.

Due to overhead created by the operating system there was no way to guarantee that the 3.5 character was enough to signify the end of a packet as in the Modbus. The 3.5 character times were used to segment groups of bytes received by the RS-232 port. These groups were then checked to see if they contained a complete Modbus packet with valid checksum. Using a wired connection a small number of packets were split in two and needed this reconstruction. When the communication was switched to wireless serial the packets were often split into 2 and sometimes more pieces, it was during the testing over the wireless lines it became apparent a mechanism was needed to reconstruct these packets to recreate the robustness that is required by SCADA systems.

The write data to UART thread is the same for both the Master and Slave with the only exception being they are writing to different UARTs. The Diagram below shows the flow of the write data to UART thread.

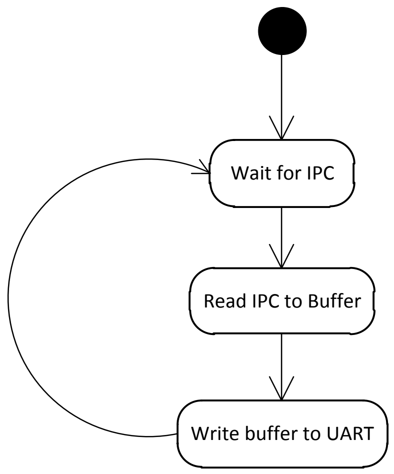


FIGURE 4.4.5 Flow in the UART write thread

The operations of this thread are fairly simple. Wait for a message from the filter then write that data to the UART.

Like the communication cells, which have two threads each the filtering cell also has two threads one for managing traffic from the slave to the master and one for managing traffic from the master to the slave. The first thread that manages traffic from the slave to the master is very simple. All it does is wait for packets from the slave cell then forward them to the Master cell. This thread is mostly needed to create isolation between the Master and Slave communication cells.

The thread that filters data from the master cell to the slave cell is much more complex. This cell is responsible for setting up communications with a user, performing role based access control, and validating packets via challenge response. This section will provide an overview of these operations but more details can be found in chapter 3.

Like all the threads in this system the general idea for this thread is wait for packet, perform operation, wait for next packet. So first the system waits for a packet from the Master cell. After receiving the packet, the packet is checked to see if a challenge or connection request is required before allowing this packet to pass through. If a connection request is required the filtering cell sends a connection request back to the master cell which forwards the packet to the master (HMI). Likewise, if a challenge is required, a challenge is sent to the master cell which forwards the packet to the Master (MHI). Otherwise the packet is rejected entirely or allowed through to the slave cell which forwards the packet to the slave device. The diagram below shows the summary view of the flow of the filtering cell.

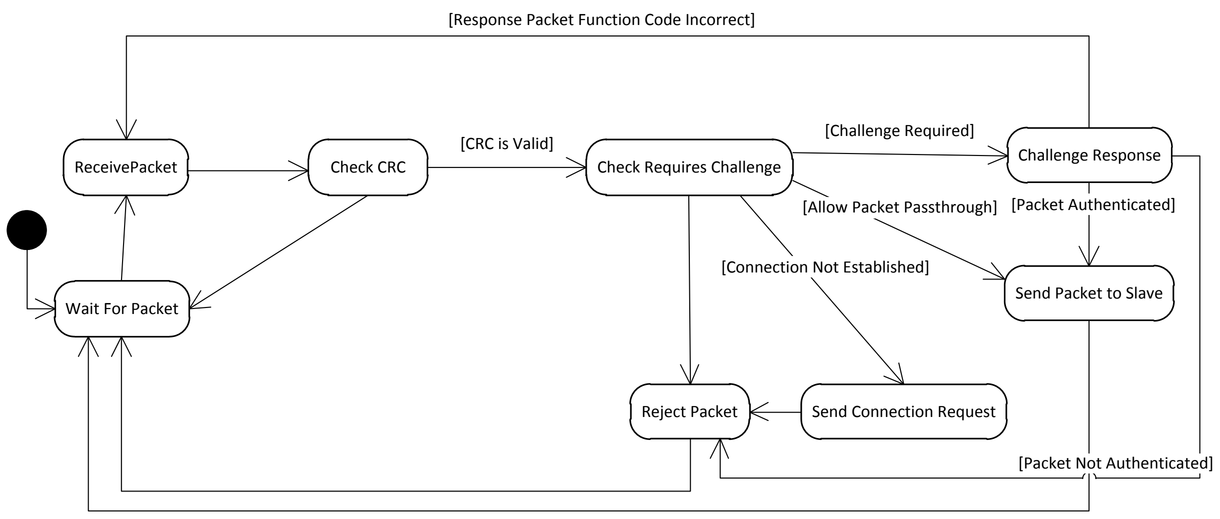


FIGURE 4.4.6 Flow of a packet through the filtering cell

The check requires challenge function plays a vital role in determining what happens to each packet. Show in figure 4.4.7 is a diagram that shows how the outcome of for each packet is formed.

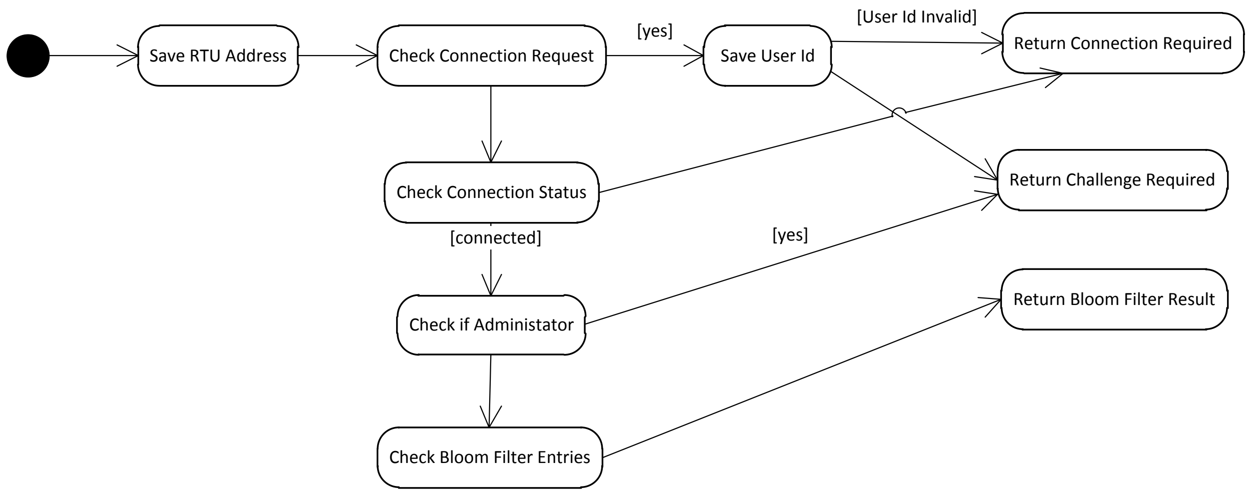
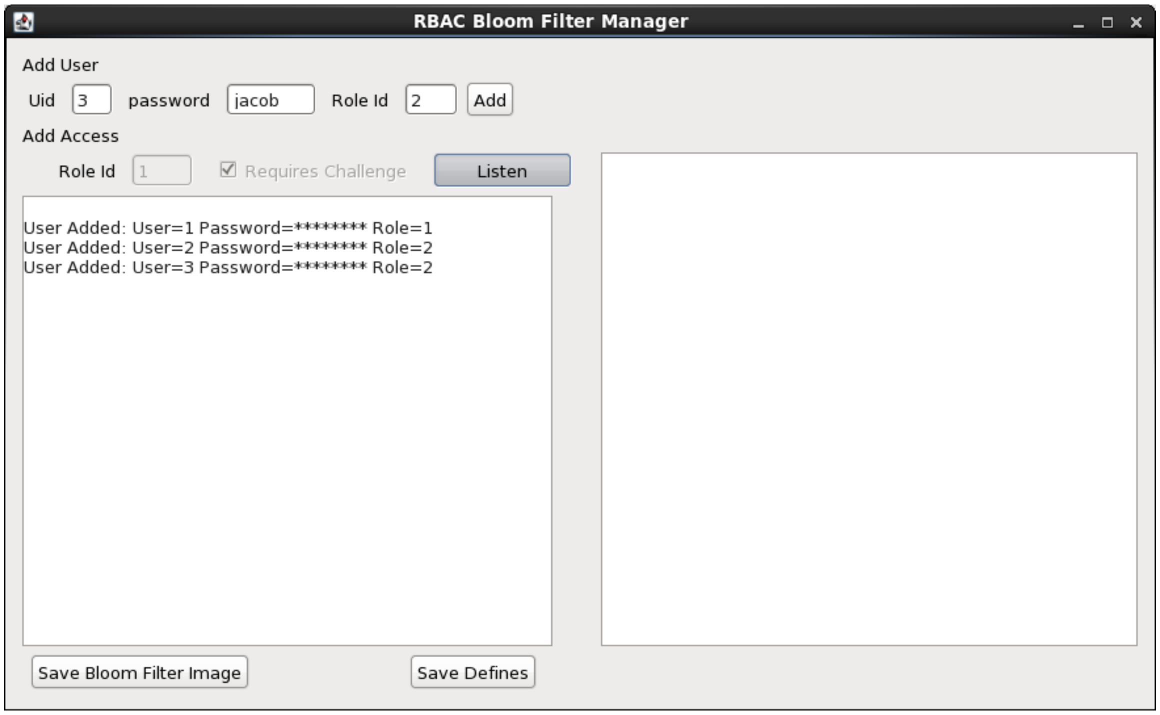


FIGURE 4.4.7 Modbus packet determination flow

The diagrams both show a very high level view of the operations of the filtering cell, to see more details see the challenge response and role based access control section below that specifies the detailed implementation of each of these functions.

4.5. Implementation of the Dual Bloom Filters for RBAC

As described in chapter 3, dual Bloom filters were created for the role based access control for the field device security preprocessor, in order to create the optimal Bloom filter, it must be known previously how much data, or in this case how many packets are going to be stored in the Bloom filter. In order to do this for an existing system the packets can simply be watched. A listener with a similar architecture to the FD-SPP can instead of filtering packets output them to a program that simple records them. It must also be known which user must be allowed to perform each operation. In order to find all the packets and create the Bloom filter in a simple easy to use way the following program was created. The recording system works as follows, first a system user is created, as well as a role for that system user. Then the user begins using the existing SCADA system as they would normally, the program listens to the packets used by the user and saves them. It can also be specified whether the operations or some of the operations require a challenge. After all the operations for one role, the next user can perform all of their operations, and the system can record those. After all the operations that the system should perform are recorded along with the users that can perform all those operations the dual Bloom filters containing those operations can be created. Shown below is a screenshot of the recording software capable of creating the dual Bloom filters for an existing SCADA system.



```
User Added: User=1 Password=******** Role=1
User Added: User=2 Password=******** Role=2
User Added: User=3 Password=******** Role=2
```

FIGURE 4.5.1 Bloom Filter Creation Software Screenshot

The Bloom filters created from this software can simply be placed into the field device security preprocessor, which will then use them for RBAC control. In this system all the packets must be known before the system is implemented, creating the Bloom filters before implementation and creating no method for updating them prevents an attacker from changing the role based access control policies. Future versions of the FD-SPP may allow for updating but this will not be done through a network update. The Bloom filters could be kept in removable flash memory, which can be physically swapped out in order to update the RBAC policy. In the current implementation however, the Bloom filters are hard coded and updated at compile time.

4.6. An Example Access Check in the Dual Bloom Filters

The Bloom filters created by capturing software are checked by the FD-SPP in order to implement its RBAC. Like described in the previous chapter this is done by appending the role of the currently logged in user with each Modbus packet received by the FD-SPP and passing it through a variety of hash functions. However the FD-SPP only passes the <role, operation> pair into a single hash function SHA-256. This 256-bit hash can be broken up into a large number of small hash functions. For example suppose Bloom filters of length 1024 bits are used, for this hash functions that produce 10 bits ($\log_2(1024)$) are required. The single SHA-256 hash can be used to create twenty-five 10-bit hash functions. Since SHA-256 is approved by NIST[25], which list random number generators as recommended use for its approved hash functions[26], the bits in the SHA-256 hash must uncorrelated be completely independent of each other[27]. This means that the sub-hash functions (10-bit chunks of the original SHA-256 bit hash) can be seen as independent hash functions that are all suitable for use for implementation of a Bloom filter. This technique of splitting a large hash function into smaller hash functions was also used by Tripunitara and Carbunar [16]. In order to ensure correctness are reduce coding time an open source c implementation of SHA-256 created by Aaron Gifford[28] and modified by Brad Luyster[20] for compatibility with OKL4 was used to generate the SHA-256 hashes required by the FD-SPP.

Assuming that k is selected to be seven, each hash function can be created from 10 bits any two bytes of the SHA-256. Figure 4.6.1 shows how 7 hash functions are created using the first 14 bytes of the SHA-256 Algorithm.

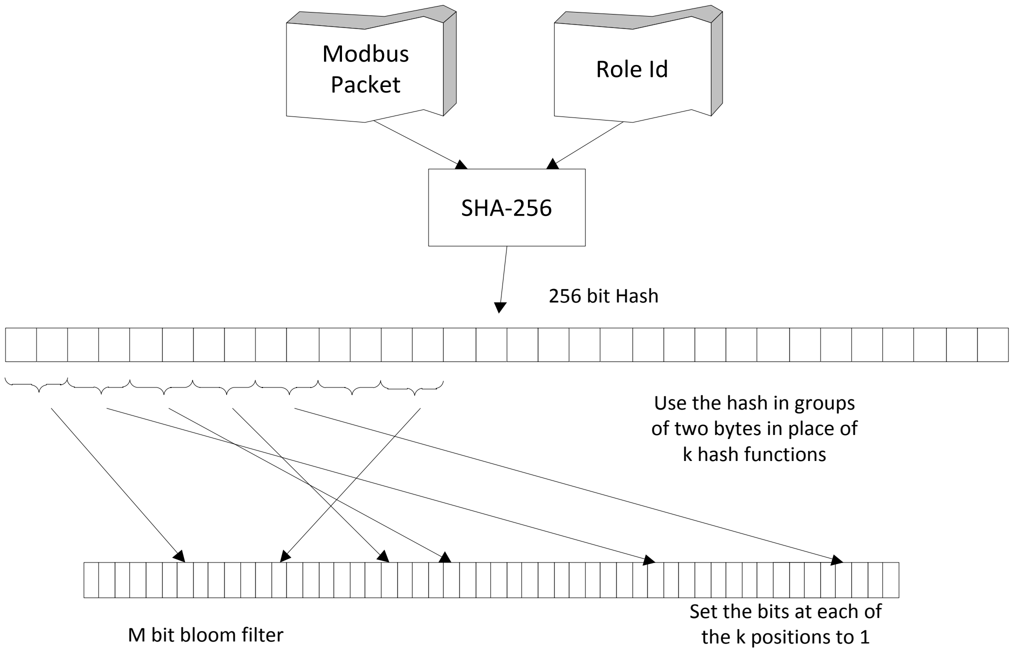

Modbus Packet
Role Id
SHA-256
256 bit Hash
Use the hash in groups of two bytes in place of k hash functions
M bit bloom filter
Set the bits at each of the k positions to 1

FIGURE 4.6.1 Using SHA-256 to Add Entries to a Bloom filter

4.7. Reduction of the False Positive Rate

In order to reduce the false positive rate it is important to understand what variables are related to the false positive rate of the Bloom filter. Simply put, if two Bloom filters have the same number of hash functions k, and the same number of bits m, then the only thing that can make the false positive rate any different is the number of ones. But how could one reduce the number of ones for a given Bloom filter without reducing the amount of data stored in the filter. Theoretically the number of entries into the Bloom filter is based on the number of entries times the number of hash functions minus the number of collisions. If the number of collisions is increased the Bloom filter will have a lower false positive rate [10]. It is important to note that this does not mean getting hash functions which generally create more collisions it means getting hash functions that collide for the specific values that are added to the Bloom filter. The hash functions must still have uniform results for any arbitrary input data otherwise the bias will allow for more false positives not less [10].

For example, assume two entries A and B are going to be placed in to a Bloom filter that uses 7 hash functions. When they are added they each add 7 bits to the Bloom filter for a total of 14 bits set. What if we had a list of hash functions where we could select the 7 hash functions that had the most collisions? First assume we can use the same number of hash functions but now there is one collision. This collision results in 13 bits added to the bloom filter instead of 14. Since this value is being taken to the k power, even adding a small number of collisions can have a large effect on the false positive rate.

$$\frac{13^7}{14^7} = .59526 \tag{4.7.1}$$

In this case the adding of a single collision reduced the false positive rate to 60% of its former value. Now for a more general case let x be the number of entries in the filter and c be the collision percentage that can be invoked, therefore $\left(\frac{x}{m}\right)^k$ can be reduced to $\left(\frac{x\,(1-c)}{m}\right)^k$. This means that we can reduce the false positive rate by $1-(1-c)^k$. For example creating a collision rate of 10% for the known entries of the Bloom filter reduces the false positive rate by over 50% when 7 hash functions are used.

In the example above the challenge response Bloom filter had 7 hash functions, 1024 bit length and 14 bits set to one. This produces a false positive rate for non-challenged entries of

$$p = \left(\frac{14}{1024}\right)^7 = 8.9289 * 10^{-14} \tag{4.7.2}$$

Using 74 hash functions to search for collision among the 2 entries, 7 new hash functions all with uniform output distributions were able to be selected that had 3 collisions for the 2 entries into the challenge response bloom filter. This reduces the number of ones from 14 to 11. This creates a collision rate of $c = \frac{3}{14}$, which allows for a reduction in the false positive rate of

$$1-\left(1-\frac{3}{14}\right)^7 = 0.81514 \tag{4.7.3}$$

This allows for over an 80% reduction in the false positive rate using the same number of hash functions, entries, and bit length. The new false positive rate for non-challenged false positives is

$$p = \left(\frac{11}{1024}\right)^7 = 1.65063 * 10^{-14} \tag{4.7.4}$$

Additionally all the hash functions used for this were based on the same SHA-256 hash as the previous design was therefore there is no additional cost for using these hash partitions as opposed to the original partitions. For example pairs of two bytes were taken to create each of the 7 hash functions, however only 10 of the 16 bits were needed. Hash functions could be created from any 10 of these 16 bits, thus creating a number of hash functions that can be easily used with nominal cost. 8008 different 10-bit hash functions, with 10 unique bits can be selected from the 2 bytes used above to create the sub-hash functions in the example above. Since all of the bits of the SHA-256 hash function should be uncorrelated and unbiased any 10 bits can be selected to make a 10 bit hash function that is suitable for Bloom filters. The hash functions created from the SHA-256 hash should not share any bits, to ensure they are not correlated in the general since. If the absolute minimum false positive rate is desired, it is possible to check the false positive rate of all the Bloom filters that could be created from all the allowable combinations of hash functions. Since the design of the FD-SPP requires that the entire set of <role, operation> pairs to known in advance it is not unfeasible for this to be done, however this would take a very long time and is probably not worth the effort. Figure 4.7.1 shows the probability of each possible number of bits being set to "1", in a 1024 bit Bloom Filter using 7 hash functions after 18 entries have been added created by simulating bits being set in Bloom Filter 10000000 times.

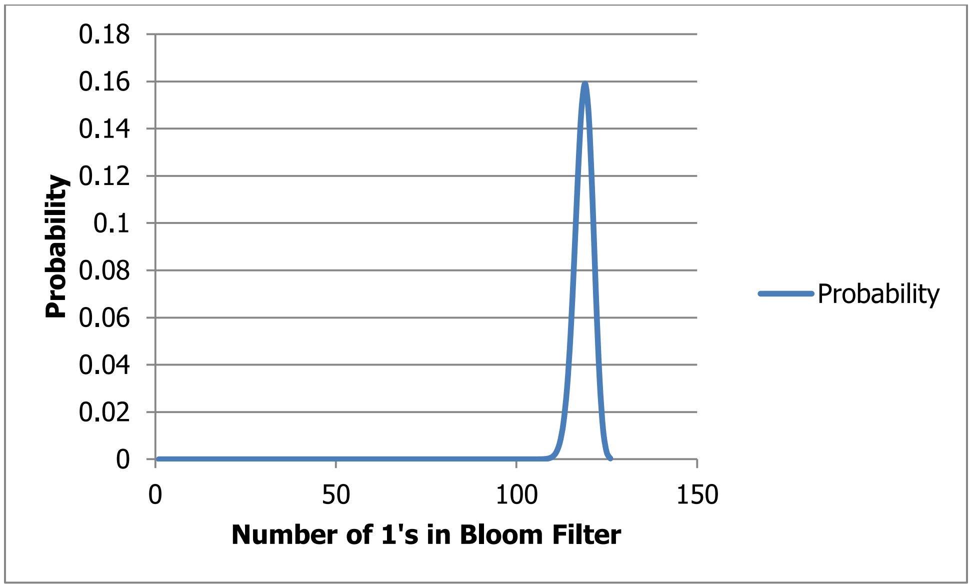


FIGURE 4.7.1 Probability Distribution of number of ones in the Bloom Filter

The average number of bits set to "1" for this Bloom filter is 118.6. In 10000000 runs of the simulation the lowest number of bits set to 1 in any Bloom filter was 103 out of the 1024, this happened 2 times. It is possible for a Bloom filter of this size to have much less bits set to "1", but the odds of this happening are very low since the odds of getting smaller number of bits shrinks exponentially. There is only around a 1% chance of getting a value of 112 or less. 112 bits set to "1" corresponds to a false positive rate of 1.87252E-07, which is the around two thirds of the average false positive rate for this Bloom filter of 2.7975E-07. For this size filter if 100 combinations of hash functions are created and tested there is a good chance one set will be found that will have a false positive rate of around two thirds of the estimated false positive rate. Searching 10000000 combinations of hash functions is likely to produce a Bloom filter that has a false positive rate of one third of the estimated false positive rate. As can be seen a short search can reduce the false positive rate of the Bloom filter, and searching for a long time can produce an even larger reduction. It is up to the implementer to decide how much time they are willing to use searching for hash functions that produce Bloom filters with large number of internal collisions, and thus low false positive rates.

4.8. Prototype of the Field Device Security Preprocessor

For the purpose of lab testing, a prototype FD-SPP was constructed using the previously described design. The prototype was built using the Gumstix® verdex pro™ XM4 COM single board computer. The OKL4 software system is designed to run on the Marvell® PXA270 with XScale® processor which was the primary motivation for choosing the Gumstix® verdex pro™ XM4 COM. The XM4 has 64 MB of RAM and 16 MB of Flash. Currently the FD-SPP doesn't use the Flash memory and all code is operated in RAM, however this may change in later revisions. The entire system uses less than 1 MB of RAM. The system also requires 2 or more serial communication ports, one to connect to the master device and one to talk to the slave. These can be added by connecting the Gumstix console-vx expansion board. Shown in figure 4.8.1 below is the Gumstix and the attached console-vx board that were used to create the prototype for the FD-SPP.

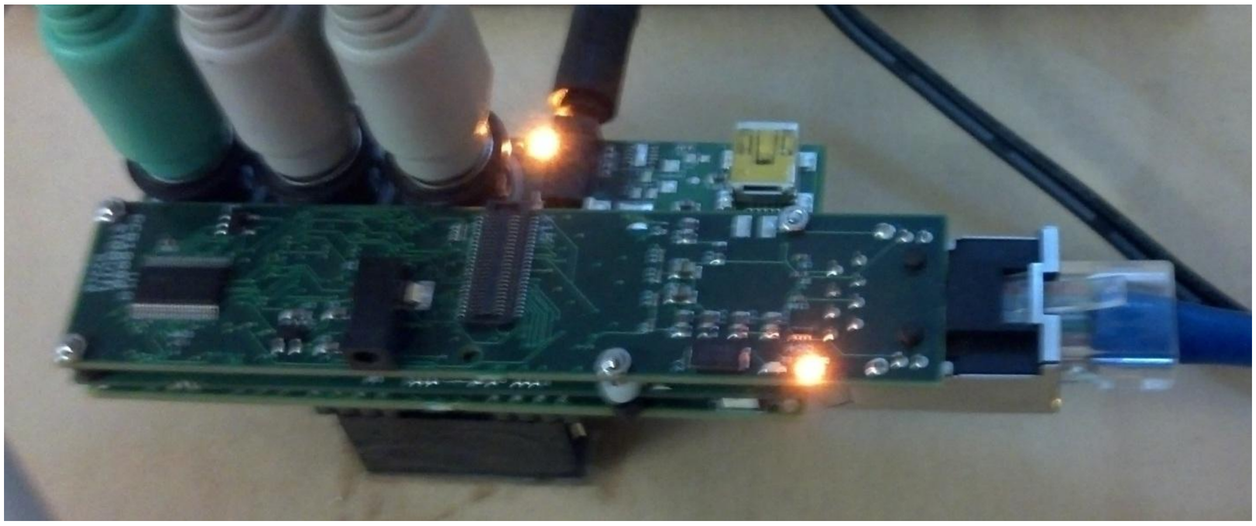


FIGURE 4.8.1 Field Device Security Preprocessor Prototype

5. TESTING OF THE FIELD DEVICE SECURITY PRE-PROCESSOR

This section describes testing and refinement of the prototype FD-SPP. An HMI/MTU test harness and a simulation environment were constructed for testing purposes. An actual legacy field device, a Sixnet mIPM RTU, was part of the test framework. The mIPM supports serial communication and the Modbus protocol. The mIPM has 24 IO points, and a simple HMI/MTU was built that could, via Modbus, read and write these "coils".

5.1. Java Modbus HMI Design

For this project the Modbus HMI and MTU were integrated into a single Java program developed using Netbeans IDE. The software uses the RXTX[29] package to communicate via RS232 to the FD-SPP. The program provided typical Modbus features such as reading and writing coils, as well as, support for security. The HMI provides four toggle buttons, which when pressed send Modbus write coils packets to the FD-SPP. The HMI also provides a "read" button which sends a Modbus read coils packet to the FD-SPP. Additionally the HMI provides the extended Modbus features required by the FD-SPP. The login request packet is sent by pressing the “login button”; this sends a login request packet containing the provided user id. The HMI also automatically replies to challenge responses using the password that is in the password field.

The HMI also monitors responses from the FD-SPP. As with most Master-Slave protocols when the master sends a message there will always be a response. If the HMI does not receive a response to any packet it sends, it will indicate this by turning the box next to the words “no reply” red, and force the user to issue a new login request. When a valid packet is received the box next to connected will turn green to indicate the communication channel between the HMI and FD-SPP is working properly. If this packet is a response to a read coils command the digital input indicators will show which digital inputs are on (red) and which are off (gray).

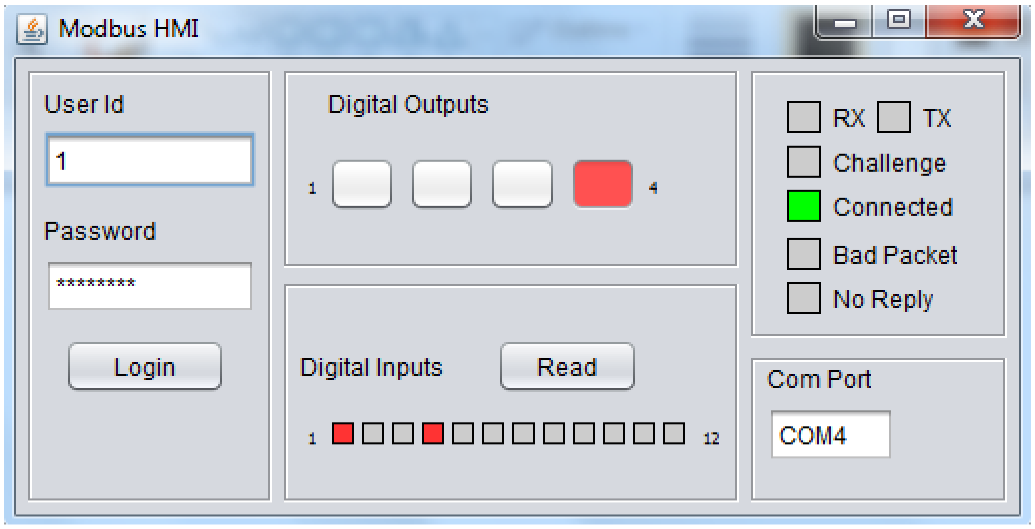


FIGURE 5.1.1 Java Modbus HMI

5.2. <u>Initial performance data</u>

Adding the FD-SPP increases the amount of time it takes for a packet to be sent from the master to the slave and from the slave to the master. A command line based communications timing program was created in C# to test the timing performance of the test SCADA network with and without the field device security preprocessor. The timing data below in table 5.2.1, collected by the program, shows the increase in time compared to the time without the security device in place. Two cases were tested for each, "with security" and "without security" implementations. The first case was reading coils. The second was writing coils, which included a challenge response for the "with security" implementation. For the purpose of this test, and the rest of the tests below, write coils commands are always challenged when the FD-SPP is in place, and read coils commands are not challenged.

Shown below in figure 5.2.1 is the diagram of how challenged Modbus packets would propagate through the system. The Timer software replaces the HMI. It sends a Modbus packet to either the RTU(to test without security), or the FD-SPP(to test with security) times how long it takes to receive the Modbus response for the sent message, therefore how long it takes the packet to propagate through the system, including the challenge response cycle if required.

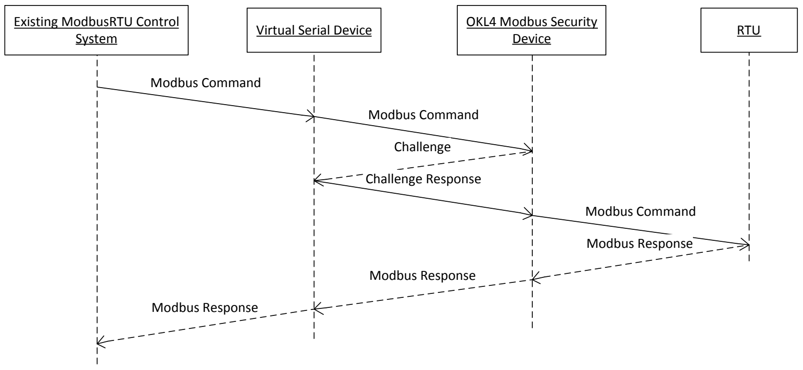


FIGURE 5.2.1 Modbus packet propagation

TABLE 5.2.1 Initial Timing Data

	Read Coils		Write Coils	
	Without Security	With Security	Without Security	With Security
Trails	500	500	500	500
Minimum	116ms	144ms	127ms	257ms
Median	119ms	163ms	129ms	281ms
Average	119ms	165ms	129ms	278ms
Maximum	140ms	207ms	140ms	304ms

This timing data was taken using 9600 baud serial connections. A Similar FD-SPP implementation using alternative communication protocol was also tested at a later time. Data from this alternative communication showed that packets with challenges could propagate through the system in less than 90ms.

5.3. Modbus compliance testing

This set of test was written using the Unit Testing Suite built into Visual Studio. The Test Suite works in place of the HMI, just like the timer program. Shown in figure 5.3.1 is the system diagram for these tests.

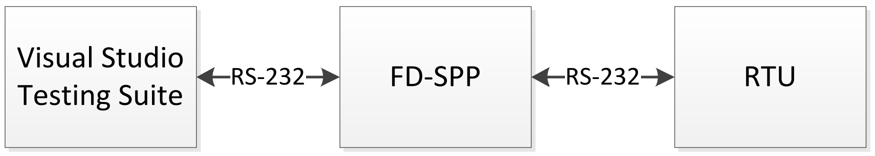
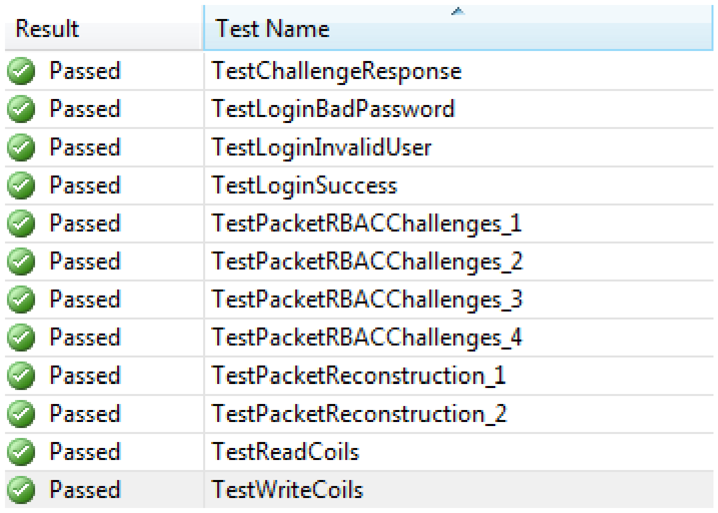


FIGURE 5.3.1 Compliance testing system diagram

These tests do not test security but test that the system operates as specified to when any given packet is encountered. Shown below is the test result output which was displayed by Visual Studio. The following subsections discuss each individual test in more detail.



Result	Test Name
✓ Passed	TestChallengeResponse
✓ Passed	TestLoginBadPassword
✓ Passed	TestLoginInvalidUser
✓ Passed	TestLoginSuccess
✓ Passed	TestPacketRBACChallenges_1
✓ Passed	TestPacketRBACChallenges_2
✓ Passed	TestPacketRBACChallenges_3
✓ Passed	TestPacketRBACChallenges_4
✓ Passed	TestPacketReconstruction_1
✓ Passed	TestPacketReconstruction_2
✓ Passed	TestReadCoils
✓ Passed	TestWriteCoils

FIGURE 5.3.2 Compliance Checker Output

5.3.1. Test Login Success

This test checks to make sure that the user login procedure works properly for a valid login. First the test sends a login request packet to the OKL4 Modbus security device. The test then collects data from the receive port for a small amount of time. This data is then checked to make sure it is a challenge packet. The test then sends the proper challenge response to complete the login. Once again the test waits to see if the security device response. Since currently there is no "login complete packet" the security device should not respond to the challenge response. A "login complete packet" may be created for a future revision of this system. After the system has verified that device did not send any data, it sends a read coils packet to confirm the connection. If the response to this packet is not the "Connection Required/Requested Packet" the connection is deemed successful.

5.3.2. Test Login Invalid User

This test checks to make sure that the user login procedure works properly when an invalid user id is sent in the connection request. First this test sends a connection request with an invalid user id to the OKL4 Modbus security device. Since the user id is invalid the OKL4 Modbus security device responds with a "Connection Required/Requested" Packet. The test checks to see if the packet it received is a "Connection Required/Requested" Packet, if so the OKL4 Modbus Security Device passes this test.

5.3.3. Test Login Invalid Password

This test checks to make sure that the user login procedure works properly for a login with a valid user but incorrect password. First, the test sends a login request packet to the OKL4 Modbus security device. The test then collects data from the receive port for a small amount of time. This data is then checked to make sure it is a challenge packet. The test then sends the challenge response formed with the incorrect password to complete the login process. Once again the test waits to see if the security device responds. Since currently there is no "login complete packet" the security device should not respond to the challenge response. A "login complete packet" may be created for a future revision of this system. After the system has verified that the device did not send any data, it sends a Read Coils packet to confirm the connection. If the response to this packet is not the "Connection Required Packet" the connection is deemed successful. In this case receiving the "Connection Required Packet" is a successful run of the test since this would imply that the login was unsuccessful.

5.3.4. Test Packet RBAC Challenge

This test is broken up into four separate tests. Each of the tests preforms a login with a particular user and then sends a packet. This test is verifying the proper response is returned by the RBAC. The three responses tested are: no-challenge, challenge, and reject. For this Test it is assumed the following permissions have been set up in the bloom filters. User 1 can read and write coils but challenges are required on writes. User 2 can read coils only, no challenges are required.

- The first test performs a login as user 1, and sends a read coils packet. The test then verifies that there is a response to this packet, and that the response is a valid read coils response and not a challenge request.
- The second test performs a login as user 1, and sends a write coil packet. The test then verifies that the response to this packet is a challenge request.
- The third test performs a login as user 2, and sends a read coil packet. The test then verifies that the there is a response to this packet, and that the response is a valid read coils response and not a challenge request packet.
- The fourth test performs a login as user 2, and sends a write coil packet. The test then verifies that there is no response to this packet, therefore it was rejected.

5.3.5. Test Challenge-Response

This test is responsible for testing the full challenge response exchange as shown in figure 5.2.1. This test first performs a login as user 1 then the sends write packets to the RTU which causes a challenge to be made. The proper response is then sent to these challenges. The test then waits for the response of this challenge. If this response confirms that the coils have been written then the challenge response protocol is working properly. This test also tests improper challenge responses to confirm that the proper response is required for a packet to propagate to the slave RTU.

5.3.6. Test Packet Reconstruction

This test is responsible for testing the packet reconstruction abilities of the OKL4 Modbus security device. This test sends packets that have been split by a small amount of time, to confirm that the OKL4 Modbus security device is properly combining them into a single valid packet. This reconstruction is necessary because the separation of bytes into packets is done in software by the OKL4 Modbus security device and can often not be precise enough to separate packets based on the 3.5 character stop time specified in the Modbus protocol standard. This test uses unchallenged read commands split in to pieces and verifies that a read response was sent in return. The test also sends garbage packets in an attempt to trick the OKL4 device. The test makes sure that only consecutives packets pieces that combine to a single valid packet are accepted by the device.

5.3.7. Test RBAC Suspicious Mode

This test is responsible for making sure the system properly enters and leaves suspicious mode. This is performed by creating a user that can only read coils. The test logs in as this user, and attempts to write to a coil. Since this operation is not allowed by the RBAC this should cause the system to enter suspicious mode which cause all packets to be challenged. The test then attempts to read a coil. The response of this should be a challenge if not the test fails. If the response is a challenge the proper response is sent sending the system back into normal mode. A second read packet can then be sent to verify this transition. If the packet is not challenged, the devices has properly reentered normal operating mode. Various amounts of read packets are sent after a write packet to confirm proper operations of the suspicious mode, and the switching between modes.

5.3.8. Test Write Coils

Unlike all of the tests above, this test requires a tester to supply input to the test. This test logins as a user and begin writing coils. After each write the test pops up a dialog asking what the current state of the RTU digital outputs are. The test then checks to see if which of the DO on the Sixnet RTU are active, and inputs this to the dialog. If the input to this dialog matches the write coils packet that was sent to the RTU then the write was a success. Shown below in figure 5.3.3 is the dialog presented to the tester.

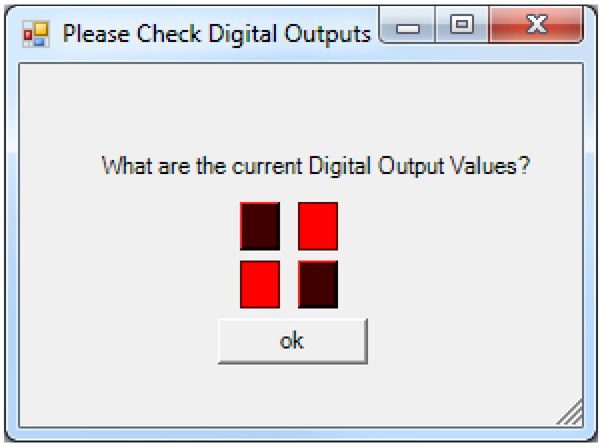

Please Check Digital Outputs
What are the current Digital Output Values?
ok

FIGURE 5.3.3 Digital output dialog displayed to Tester

5.3.9. Test Read Coils

This test is responsible for making sure reading coils is working properly. This test is very similar to the write coils test; it asks a tester what the current states of the digital inputs are the Sixnet RTU are and then performs a read. If these two values match then the coils were read properly. It then allows the user to change the coils and read again to further confirm the read is working correctly. Shown below is the dialog for the read coils test.

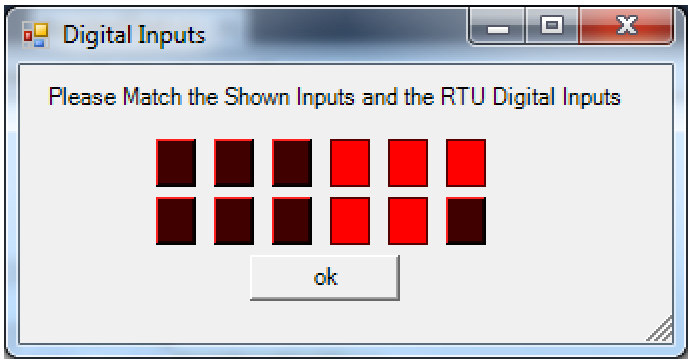


FIGURE 5.3.4 Read coils dialog box

5.4. Penetration Testing

For the purpose of penetration testing two lab setups were conceived. The first of which did not have the FD-SPP, and the second did. Shown below are the two lab configurations.

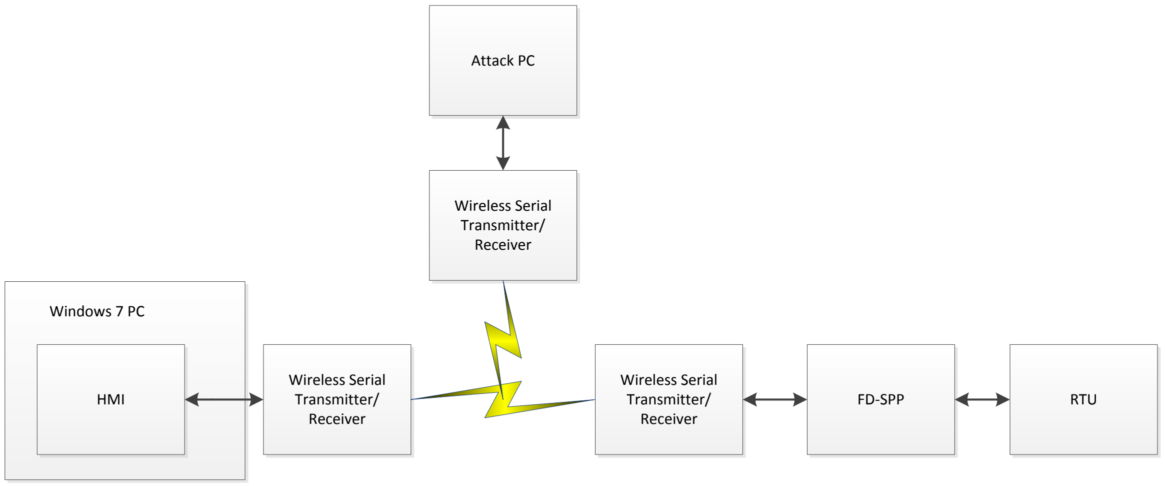

Attack PC
Wireless Serial Transmitter/ Receiver
Windows 7 PC
HMI
Wireless Serial Transmitter/ Receiver
Wireless Serial Transmitter/ Receiver
FD-SPP
RTU

FIGURE 5.4.1 Penetration testing setup with FD-SPP

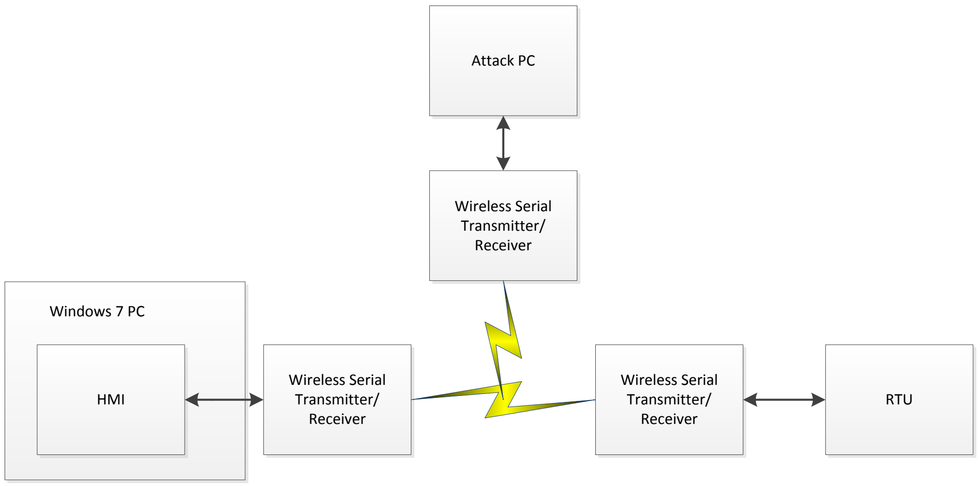

Attack PC
Wireless Serial Transmitter/ Receiver
Windows 7 PC
HMI
Wireless Serial Transmitter/ Receiver
Wireless Serial Transmitter/ Receiver
RTU

FIGURE 5.4.2 Penetration testing setup control setup

The attack PC was used to monitor, and inject Modbus packets onto the wireless serial connection. A number of different attacks were carried out. The different attacks and the results are described in the following sections. Table 5.4.1 summarizes the results of the penetration testing.

TABLE 5.4.1 Penetration testing results.

Attack	**Without Security**	**With Security**	**With Security and Signatures**
Write Coils	***Success***	***Failed***	***Failed***
Write Random Coils	***Success***	***Failed***	***Failed***
Read Coils	***Success***	***Success***	***Success***
HMI Read Attack	***Success***	***Success***	***Failed***
HMI Write Attack	***Success***	***Success***	***Failed***
DOS Attack	***Success***	***Success***	***Success***

5.4.1. Write Coils Attack

Since there is no built in security anyone on the network can send a packet. For this test a simple write multiple coils command were transmitter onto the network. The packet sent on the network (in hex) was *010F000000040105fe95*. For this attack the FD-SPP protected the RTU from the attack, while without the security the attacker was very easily able to write to the coils.

5.4.2. Write Random Coils Repeatedly Attack

This attack takes the previous attack one step farther. It sends randomly generated write coils attacks very rapidly in an attempt to get the security software confused, or in an incorrect state. For this attack the security piece protected the RTU from the attack, while without the security the attacker was very easily able to write to the coils. The security piece did not allow any of the unauthenticated packets to go through and after the attack was able to return to normal operating mode and allow authenticated write coil packets to pass through.

5.4.3. Theoretical Write Coils Attacks

Since with all security it is not whether or not an attack is possible that matters, but a measure of how difficult the attack is to perform, this section will cover the theoretical attacks that could be used. This section covers writing coils but can apply to any packet that is a challenged packet and in the Bloom filter as such for a particular user. There are two mechanisms that control security, and in order to defeat the security the attack must exploit at least one of these mechanisms.

The first of which is the challenge response. When a packet, such as a *Write Coils* packet is challenged, the attack could attempt to create the proper response packet. Since each challenge contains 4 bytes of nonce the user cannot simply apply a replay attack. If the user simply records traffic they can begin to accumulate challenge responses. Eventually since there are a finite number of nonce values the attacker will eventually be able to perform a replay attack using the saved nonce values. A size of four bytes means that the nonce can take the form of 4294967296 different bit sequences. However the attacker would not need to see every bit sequence to get a single packet through, using similar equations to the Bloom filter ones in chapter two the average amount of packets an attacker would have to see before getting a successful replay can be calculated by solving equation 5.4.1, where m is the total number of combinations, and n is the average number of bit sequences it takes to see a single repeated sequence.

$$m\left(1-\left(1-\frac{1}{m}\right)^{n}\right)=n-1 \tag{5.4.1}$$

Solving for n numerically using Maple with m = 4294967296, gives n = 92682.73335. Assuming the attacker gets the best case scenario, seeing the same command sent continuously, they will see on average one packet they need approximately every 275ms it would take over 7.5 hours for the attacker to get a single packet they can replay. However the attacker would probably want a much higher number of packets to perform an actual attack. For example for the attacker to have a 10% chance of sending a packet and receiving a challenge packet they must have seen 10% of the total number of packets that are different. Solving equation 5.4.2 for n gives the number of packets that must be seen for the attacker to have a 10% chance of getting an attack to succeed.

$$\left(1-\left(1-\frac{1}{m}\right)^{n}\right)=\ .10 \tag{5.4.2}$$

Solving for n numerically with m = 4294967296, gives n = 452519969. Again assuming the attacker gets the best case scenario as describe previously it would take almost 4 years to get enough packets. Additionally, it is unlikely the same user will be sending the same packet over and over on any SCADA system the attacker would want to attack. This being showing it is highly unlikely an attacker would ever succeed in a replay attack on the FD-SPP.

The second part of the challenge response the user could attempt to guess is the password. The password is stored in 8 bytes, which means there are $2^{(8*8)}$ combinations of passwords. In order to check passwords the attacker would have to write coils and use the password on the challenge then check the coils. Without being able to see the coils this would be difficult. The other strategy would be to try to login as a user using the password. Since there is no successful login response the attack would then have to perform another operation to verify that the login was successful. Combined either of these strategies would take at least 300ms. In order to check all the passwords that could be used it would take $1.75*10^{11}$ years. Even if only half this time was needed (the time on average to find the correct password) the attacker would still be searching for a time much greater than their lifetime. Additionally this type of attack would also act as DOS on the network and be quickly detected by any user attempting to use the system.

The second system an attacker could attack is the RBAC Bloom filters. This attack is even more unlikely to be successful than the previous attack. This attack involves creating a packet that will successfully make it through the non-challenged Bloom filter. This means the packet would be directly sent to the RTU. However most regular Modbus packets are unlikely to be in the non-challenged Bloom filter and it is most likely that they are checked to make sure they are not in this Bloom filter. This means that a special attack packet must be crafted to write the coils, or be performed when a user that cannot write coils is logged in.

For the attack to work using a user that cannot write coils a false positive must occur whenever that user writes coils, since there are only a small number of packets which this can occur they can all be checked to make sure none exist. This special packet would contain a Modbus write coils packet as the first part of the packet, and then data would be appended to the packet to make it in the Bloom filter. The proper CRC would also have to be added to the packet a valid Modbus packet. If the packet was able to pass through the Bloom filter and be sent to the RTU it would require that the RTU be checking the packets for a CRC match byte by byte, or an error occur splitting the write coils packet from the rest of the packet to create a successful write.

This attack is more of a series of unfortunate events more than it is a planned Modbus attack. Also since after one failure the RBAC Bloom filters enter suspicious mode, the attacker would only have one attempt before they would have to wait for regular traffic to reset the RBAC back to normal mode. This attack, like the one above, would also be detected way before the attacker could succeed. Since the attack needs a bit error to occur, when the attacker finally finds a packet that is a false positive the attack may still not be successful because there is a very low probability of getting the exact bit error the attacker needs. Without offline access to the Bloom filter, the attack will likely never know if they have found a false positive. Therefore the attack success rate will be much less than the bloom filter non-challenged false positive rate. Attack packets like the ones discussed here were sent at the RTU directly without the security to test the likelihood of a this type of bit error occurring; however in the 1000 packets sent 0 were successful in writing the coils. Therefore it is extremely unlikely an attack of this kind could succeed.

5.4.4. Read Coils Attack

In the system that was used for testing reading coils was considered non critical. The challenge response was not required to read coils. Therefore once a user logs in the coils can be read in the same way they would be in a non-secured system. In order to attack this system a read coils command was transmitted by the attack, the attacker could then read the resulting packet. If the attacker wanted to be less obvious about their attacks the attacker could just wait until a read coils command is issued by the user then read the results. Since the data is not encrypted, an attacker would be able to perform this second attack even if challenge response was applied on issuing the command. Both with and without the FD-SPP this attack was successful.

5.4.5. Attacks on the HMI

In current configuration packets going from the HMI to the FD-SPP can be challenged; however packets going from the FD-SPP to the HMI are not protected. Two attacks on the HMI were performed by injecting traffic to the wireless serial connection to the HMI. Both these attacks work on the system with and without the security additions.

5.4.6. Read Coils Attack on HMI

In this attack the attacker waits for a read coil command to be sent from the HMI to the FD-SPP, once the packet is sent the attacker sends a packet to collide with the read coils response sent by the FD-SPP. Since the packets collide no valid data is received. The attacker then sends their own read coils response code. The HMI sees this faked packet as the real read coils response. This allows the attacker to make the user think different things are happening in the network then what really are.

5.4.7. Write Coils Attack on HMI

In this attack the attacker waits for a write coil command to be sent from the HMI to the FD-SPP, once the packet is sent the attacker sends a packet to collide with the write coils packet to prevent it from getting to the FD-SPP, if this fails making a packet collide with the challenge will perform the same thing. The idea is to stop the write coils packet from ever making it to the RTU by any means necessary. Once the packet is stopped the attacker can then send a write coils response to the serial connection of the HMI. In this way the attacker can make the user think they are writing coils when they are not.

5.5. Digital Signatures for Return Packets

After penetration testing it was clear a second layer of security was needed. An attacker was able to trick the HMI into thinking they were the RTU. In order to prevent this attack a digital signature was added to each return packet sent from the FD-SPP to the HMI.

5.5.1. Creation of the Digitally Signed Messages

The digital signature is created the same way the challenge response hashes are. They are created by hashing the Modbus packet without the CRC, with 4 bytes of nonce, with the 8 byte pre-shared secret using the SHA-256 hash algorithm.

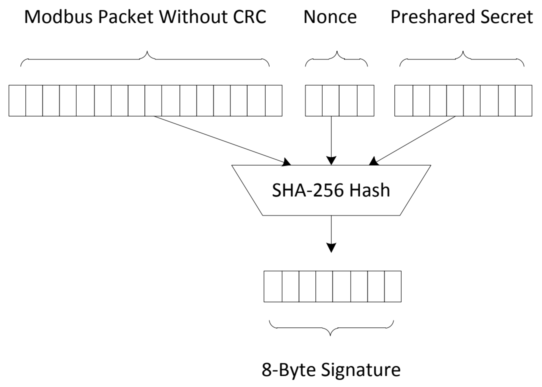


FIGURE 5.5.1 Return Packet Digital Signature

The digital signature is then appended to the Modbus Packet without CRC. A Modbus CRC for this new combined packet is then appended to the end to create a valid Modbus Packet. Using a signature instead of a challenge-response reduced the overhead created by the extra message passing that is required for a challenge-response. This way the only time that is added to the process is the time it takes to sign the packet and a small increase in packet propagation through the network since packet sizes are increased by 8 bytes.

5.5.2. Timing Data for Signed Packets

Shown in Table 5.5.1 is the table for packet propagation times after the addition of the packet signing routines.

TABLE 5.5.1 Timing data for return message digital signatures

	Read	Write
Trails	50	50
Minimum	169ms	288ms
Median	192ms	304ms
Average	189ms	307ms
Maximum	201ms	325ms

Comparing this table to the original time data reveals that adding the Digital Signature only adds a few milliseconds. Since the difference between reading and writing coils is a challenge response, it can be seen that the challenge response adds a lot more time than the digital signature.

5.5.3. Nonce Generation for the Signature

Since the Nonce is typically generated by a random number generator and sent in the challenge, the removal of this challenge presents a problem with nonce generation. It is important that the nonce be different on each message sent in order to prevent replay attacks. Additionally this nonce cannot be sent in the message because then the attacker could choose what nonce to send therefore allowing an opportunity for a replay attack.

The nonce therefore must be different each time it is generated and able to be predicted by both the sender and the receiver of the message. In order to achieve this, a pseudo random number generator was created to serve the nonce. The 32-bit (4 bytes) pseudo random number generator was creating using two independent 16-bit random number generators. The first was a linear congruential generator with period of 53124 and the second was a linear feedback shift register with period 65535 [30]. Together these pseudo random number generators combine to create a single 32 bit pseudo random number generator with a period of 1160493780. Additionally the pseudo random number generator is reseeded by a separate random number generator which adds entropy from a counter on the FD-SPP during every challenge and a separate random on the HMI during every challenge response. Since it is highly unlikely that 1160493780 packets in a row will be unchallenged it is likely that entropy will be added before the pseudo random generator repeats, and the nonce values for the digital signature should be random enough to ensure a high amount of protection from replay attacks.

Each time a nonce value was required to create a digital signature the last used nonce values are used as seeds to the generator. Since both systems know the old nonce values and have the same generator they will get the same new nonce values without have to communicate them. This allows for a shared source of nonce without having to communicate the nonce, and prevents replay attacks.

In order to inject more randomness into the nonce the nonce values from the challenge response will also be used. When a challenge response generates new random nonce both systems will set this as their old nonce to be used as a seed the next time they sign a packet. Two of these bytes will come from the 4 nonce bytes sent in the challenge and another two bytes are included in the response packet. This way both systems create and control the nonce. This prevents an attacker from being able to choose all the nonce values, and then perform replay attacks. With the addition of the digital signatures and the new nonce creation, the attacker can no longer perform either of the previously discussed HMI Attacks on the system.

6. CONCLUSION AND FUTURE WORK

This thesis has provided the details of the design and development of a Bloom filter authentication module field device security preprocesser as a security solution for legacy SCADA field device based on the previous work of Hieb and Graham [1]. This device provides several missing security features to legacy SCADA field devices using the Modbus protocol, and could easily be adapted for other SCADA protocols. The field device uses a micro-kernel operating system called OKL4, which allows for a high level of security by abstracting memory and execution spaces. The security features implemented on the field device include role based access control and challenge response. The focus of this thesis has been the design, implementation, analysis of these security features using Bloom filters.

6.1. Results Summary

Although Bloom filters have false positive authentication it was shown through analysis as well as penetration testing that this structure is acceptable for many security applications. Additionally it was shown that by increasing the space used to store the Bloom filter and by finding hash functions that produce high internal collision rates for the given data set, the false positive rate can be reduced to the point where it is more than or as secure as an n-bit symmetric encryption key.

Through performance testing it was shown that the propagation delay added by inserting a field device security pre-process around 275ms which is an acceptable level for use in the Department of Homeland Security Water Sector. The dual Bloom filter structure allows for role based access checks to be performed at a very high speed, only costing around 18µs, lowering the overhead cost of the FD-SPP on the SCADA network.

6.2. Direction for Future Work

The future work for the field device security preprocessor can go a number of directions. One direction is to add support for more protocols and interfaces. Modbus is not the only SCADA protocol being used in industry and in order to protect legacy systems, the protocols that are being used by these systems must be supported. Also, RS-232 is not the only interface SCADA systems are using. Modbus even has its own protocol for communicating over TCP/IP. In order to support these SCADA networks, this interface must be supported by the FD-SPP. Additionally the FD-SPP could act as an adapter that changes the interface from RS-232 to Ethernet or vice versa. The cell structure of FD-SPP makes adding new protocols and changing interfaces very easy; once cells are created they can be simply swapped in and out to create new variants of the FD-SPP. Preliminary testing shows that, as expected, a FD-SPP using TCP/IP is a lot faster than one using serial communication, and the overhead created by the FD-SPP is much smaller.

The OKL4 micro-kernel operating system has a close relative known as seL4, which has been formally verified [22]. Future development will need to consider porting this software to the verified kernel. Once the software is running on seL4, the software itself must be verified. Possibly, in a similar manner to seL4 using machine assisted and machine checked formal proof [22]. Although verification does not guarantee that the FD-SPP is secure, it would confirm that the software behaves completely as specified and any exploits would be of the design, not software bugs. A verified version of the FD-SPP could go a long way to reduce the vulnerabilities of legacy systems.

An alternative to verifying the FD-SPP software running on seL4 would be to implement it on a FPGA and verify the VHDL code. Verification should be much easier since there has been some prior research into the formal verification of VHDL[31]. The software only has a few simple parts and could be implemented on an FPGA with little difficulty. The Bloom Filter does not require complex software other than the SHA-256 hash. Since it is known that SHA-256 can be implemented on a FPGA, it should be a small step to implement the dual Bloom filter on a FPGA. Additionally, the challenge response routine also uses

SHA-256 and a small state machine; these two should be possible to implement on a FPGA. By implementing the FD-SPP in hardware, it can be made faster than using the general purpose hardware, further reducing the overhead added to the network.

Works Cited

[1] J. Hieb, J. Graham, and B. A. Luyster, "A Prototype Security Hardened Field Device for Industrial Control Systems," *Proceedings of the International Conference on Advanced Computing and Communications*, pp. 95–100, 2010.

[2] D. E. SANGER, "Obama Order Sped Up Wave of Cyberattacks Against Iran," *New York Times*, New York, New York, USA, p. A1, 01-Jun-2012.

[3] K. Stouffer, J. Falco, and K. Scarfone, "Guide to Industrial Control Systems (ICS) Security," *National Institute of Standards and Technology*, vol. Special Pu, 2011.

[4] J. Hieb and J. Graham, "Designing Security-Hardened Microkernels for Field Devices," *IFIP International Federation for Information Processing*, vol. 290, no. Critical Infrastructure Protection II, pp. 129–140, 2008.

[5] J. Hieb, S. C. Patel, and J. Graham, "Security Enhancements for Distributed Control Systems," *IFIP International Federation for Information Processin*, vol. 253, no. Critical Infrastructure Protection, pp. 133–146, 2007.

[6] B. H. Bloom, "Space/time trade-offs in hash coding with allowable errors," *Communications of the ACM*, vol. 13, no. 7, pp. 422–426, Jul. 1970.

[7] B. H. Bloom, *Space/time trade−offs in hash coding with allowable errors*, vol. 13, no. 7. p. 422−426.

[8] W. Ching-Yuan, Y. Wei-Pang, J. C. R. Tseng, and H. Meichun, "Random filter and its analysis," *Signals, Systems and Computers, 1989. Twenty-Third Asilomar Conference on*, pp. 1031–1035.

[9] J. Almeida and A. Z. Broder, "Summary cache: a scalable wide-area Web cache sharing protocol," *IEEE/ACM Transactions on Networking*, vol. 8, no. 3, pp. 281–293, Jun. 2000.

[10] F. Hao, M. Kodialam, and T. V. Lakshman, "Building high accuracy bloom filters using partitioned hashing," *ACM SIGMETRICS Performance Evaluation Review*, vol. 35, no. 1, p. 277, Jun. 2007.

[11] A. Broder and M. Mitzenmacher, "Network Applications of Bloom Filters: A Survey," *Internet Mathematics*, vol. 1, no. 4, pp. 485–509, Jan. 2004.

[12] J. Schreiver, K. Underwood, J. Hieb, A. Desoky, and J. Graham, "Visualization of Multidimensional Bloom Filters: An analysis of Differing Bloom Filters and their Visual Representations," *International Conference on Computer Applications in Industry and Engineering [Accepted]*, 2012.

[13] E. H. Spafford, "OPUS: Preventing weak password choices," *Computers & Security*, vol. 11, no. 3, pp. 273–278, May 1992.

[14] "Phishing and malware detection - Google Chrome Help." [Online]. Available: http://support.google.com/chrome/bin/answer.py?hl=en&answer=99020.

[15] "The Chromium Projects Source Code." Google, p. bloom_filter.h, 2011.

[16] M. V. Tripunitara and B. Carbunar, "Efficient access enforcement in distributed role-based access control (RBAC) deployments," in *Proceedings of the 14th ACM symposium on Access control models and technologies - SACMAT '09*, 2009, p. 155.

[17] M. Iwaihara and M. M. Mohania, "Time-Decaying Bloom Filters for Data Streams with Skewed Distributions," *15th International Workshop on Research Issues in Data Engineering: Stream Data Mining and Applications (RIDE-SDMA'05)*, pp. 63–69, 2005.

[18] Modbus Organization, "Modbus FAQ." [Online]. Available: http://www.modbus.org/faq.php.

[19] "Modicon Modbus Protocol Reference Guide Modicon Modbus Protocol Reference Guide," North Andover, Massachusetts, 1996.

[20] B. A. Luyster, "A PROTOTYPE SECURITY HARDENED FIELD DEVICE FOR SCADA SYSTEMS," University of Louisville, 2011.

[21] G. Heiser, "Secure embedded systems need microkernels," *National ICT Australia and University of New South Wales*, 2005.

[22] G. Klein, M. Norrish, T. Sewell, H. Tuch, S. Winwood, K. Elphinstone, G. Heiser, J. Andronick, D. Cock, P. Derrin, D. Elkaduwe, K. Engelhardt, and R. Kolanski, "seL4," in *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles - SOSP '09*, 2009, p. 207.

[23] "Intel® PXA27x Processor Family Developer's Manual," 2004.

[24] "Modicon Modbus Protocol Reference Guide Modicon Modbus Protocol Reference Guide," North Andover, Massachusetts, 1996.

[25] NIST, "SECURE HASH STANDARD," *Federal Information Processing Standards Publication*, vol. 180–3, 2008.

[26] Q. Dang, "Recommendation for Applications Using Approved Hash Algorithms," no. NIST Special Publication 800–107, 2012.

[27] A. Rukhin, J. Soto, J. Nechvatal, M. Smid, E. Barker, S. Leigh, M. Levenson, M. Vangel, D. Banks, A. Heckert, J. Dray, and S. Vo, "A Statistical Test Suite for

Random and Pseudorandom Number Generators for Cryptographic Applications," *NIST Special Publication*, vol. 800–22, 2010.

[28] A. Gifford, "Aaron Gifford's Implementations of SHA-1, SHA-224, SHA-256, SHA-384, and SHA-512." 2004.

[29] T. Jarvi, "RXTX." 1997.

[30] B. Schneier, *Applied Cryptography: Protocols, Algorithms, and Source Code in C*, 2nd ed. New York, New York, USA: John Wiley & Sons, 1995.

[31] R. Reetz, T. Kropf, and K. Schneider, "Formal Specification in VHDL for Hardware Verification," in *Design, Automation and Test in Europe*, 1998, pp. 257–263.