Mississippi State University
Scholars Junction

Theses and Dissertations | Theses and Dissertations

1-1-2013

Minimal Trusted Computing Base for Critical Infrastructure Protection

Arun Velagapalli

Follow this and additional works at: https://scholarsjunction.msstate.edu/td

Recommended Citation

Velagapalli, Arun, "Minimal Trusted Computing Base for Critical Infrastructure Protection" (2013). *Theses and Dissertations*. 3109.
https://scholarsjunction.msstate.edu/td/3109

This Dissertation - Open Access is brought to you for free and open access by the Theses and Dissertations at Scholars Junction. It has been accepted for inclusion in Theses and Dissertations by an authorized administrator of Scholars Junction. For more information, please contact scholcomm@msstate.libanswers.com.

المنارة للاستشارات

www.manaraa.com

Minimal trusted computing base for critical infrastructure protection

By

Arun Velagapalli

A Dissertation
Submitted to the Faculty of
Mississippi State University
in Partial Fulfillment of the Requirements
for the Degree of Doctor of Philosophy
in Computer Science
in the Department of Computer Science and Engineering

Mississippi State, Mississippi

August 2013

المنارة للاستشارات
www.manaraa.com

Copyright by

Arun Velagapalli

2013

للاستشارات المنارة

www.manaraa.com

Minimal trusted computing base for critical infrastructure protection

By

Arun Velagapalli

Approved:

Mahalingam Ramkumar
Associate Professor of Computer Science
and Engineering
(Major Professor)

David A. Dampier
Professor of Computer Science
and Engineering
(Committee Member)

Yoginder S. Dandass
Associate Professor of Computer Science
and Engineering
(Committee Member)

Thomas H. Morris
Assistant Professor of Electrical and
Computer Engineering
(Committee Member)

Edward B. Allen
Associate Professor of Computer
Science and Engineering,
(Graduate Coordinator)

Jerome A. Gilbert
Interim Dean of the Bagley College of
Engineering

المنارة للاستشارات
www.manaraa.com

Name: Arun Velagapalli

Date of Degree: August 17, 2013

Institution: Mississippi State University

Major Field: Computer Science

Major Professor: Dr. Mahalingam Ramkumar

Title of Study: Minimal trusted computing base for critical infrastructure protection

Pages of Study: 195

Candidate for Degree of Doctor of Philosophy

Critical infrastructures like oil & gas, power grids, water treatment facilities, domain name system (DNS) etc., are attractive targets for attackers — both due to the potential impact of attacks on such systems, and due to the enormous attack surface exposed by such systems. Unwarranted functionality in the form of accidental bugs or maliciously inserted hidden functionality in any component of a system could potentially be exploited by attackers to launch attacks on the system.

As it is far from practical to root out undesired functionality in *every* component of a complex system, it is essential to develop security measures for protecting CI systems that rely *only* on the integrity of a small number of carefully constructed components, identified as the *trusted computing base (TCB)* for the system. The broad aim of this dissertation is to *characterize* elements of the TCB for critical infrastructure systems, and outline *strategies* to leverage the TCB to secure CI systems.

www.manaraa.com

A unified provider-middleman-consumer (PMC) view of systems was adopted to characterize systems as being constituted by *p*roviders of data, untrusted *m*iddlemen, and *c*onsumers of data. As the goal of proposed approach is to eliminate the need to trust *most* components of a system to be secured, most components of the system are considered to fall under the category of "untrusted middlemen." From this perspective, the TCB for the system is a minimal set of trusted functionality required to verify that the tasks performed by the middle-men will not result in violation of the desired assurances.

Specific systems that were investigated in this dissertation work to characterize the minimal TCB included the domain name system (DNS), dynamic DNS, and Supervisory Control and Data Acquisition (SCADA) systems that monitor/control various CI systems. For such systems, this dissertation provides a comprehensive *functional specification* of the TCB, and outlines *security protocols* that leverage the trust in TCB functionality to realize the desired assurances regarding the system.

المنارة للاستشارات
www.manaraa.com

DEDICATION

To my mother Alice Mary Veda Kumari, father Samuel Rajababu, sister Jessie Ratan, and wife Ramani for their support and encouragement.

المنارة للاستشارات

www.manaraa.com

ACKNOWLEDGEMENTS

I would like to thank my major professor and mentor Dr. Mahalingam Ramkumar. I am grateful for his advise on every step of my research. I simply cannot imagine a PhD degree without his help and guidance all the way. I also thank him for the financial support provided during my doctoral research.

I would like to thank Dr. Sivakumar Kulasekaran for his advice on choosing the right advisor.

I would like to thank Pujitha Jammula for her contribution in systems design.

I would like to thank Dr. David A. Dampier, Dr. Yoginder Dandass, Dr. Thomas H. Morris for serving as members in my PhD Dissertation Committee, and for their suggestions and feedback.

I would like to thank Department of Computer Science and Engineering for providing the resources required to perform my research.

I would like to thank National Strategic Planning & Analysis Research Center (nSPARC) at Mississippi State University for funding my Master's degree. I would like to thank CVM at Mississippi State University for their partial funding provided during my PhD program.

I would like to acknowledge the source of funding for conducting this research from the Department of Homeland Security (DHS)-sponsored Southeast Region Research Initiative (SERRI) at the Department of Energy's Oak Ridge National Laboratory.

المنارة للاستشارات
www.manaraa.com

TABLE OF CONTENTS

DEDICATION ii

ACKNOWLEDGEMENTS iii

LIST OF TABLES ix

LIST OF FIGURES x

CHAPTER

1. INTRODUCTION 1

1.1 Hypothesis and Research Goals 2
1.2 Specific Contributions of this Dissertation 3

2. LITERATURE SURVEY 6

2.1 Domain Name System 6
2.1.1 DNS Records 8
2.1.1.1 Query-Response Process 9
2.1.2 Securing DNS 11
2.1.3 Link-Security Approaches 12
2.1.4 DNSSEC 13
2.1.5 Authenticated Denial 15
2.1.5.1 DNS-Walk 16
2.2 SCADA systems 17
2.2.1 SCADA System Components 17
2.2.2 SCADA Security 19
2.2.3 Vulnerabilities 21
2.2.3.1 Network Layer Vulnerabilities 22
2.2.3.2 Application Layer Vulnerabilities 24
2.2.4 Attacks on SCADA Systems 25
2.2.5 Related Work 27
2.2.5.1 Collaborative, Trust-Based Security Mechanism 27
2.2.5.2 DNP3 Secure Authentication 28

المنارة للاستشارات
www.manaraa.com

2.2.5.3 IEEE SCM 29
2.3 Merkle Hash Tree 30
2.3.1 Merkle Tree Limitations 31
2.3.2 Index Ordered Merkle Tree 33

3. TCB FOR UNTRUSTED MIDDLEMAN 36

3.1 TCB 37
3.2 Minimal TCB 38
3.2.1 Read-proofing and Write-proofing 38
3.3 TCB Models 40
3.3.1 Security Model 41
3.4 Critical Infrastructures from the perspective of module T 42
3.4.1 Static vs Dynamic Data 42
3.4.2 Static vs Dynamic Data Identifiers 43
3.4.3 TCB Tasks 43
3.4.3.1 Stateless TCB 43
3.4.3.2 Stateful TCB 44

4. ATOMIC RELAY FOR DNS 46

4.1 Extending Link-Security Approaches 46
4.2 Principle of Operation 47
4.2.1 Atomic Relay 48
4.2.2 "Intelligent" Atomic Relay 49
4.3 Computing Link Secrets 51
4.3.1 MLS 52
4.3.2 Key Distribution for TCB-DNS 53
4.3.3 Multiple KDCs 54
4.3.4 Renewal 56
4.4 The TCB-DNS Protocol 57
4.4.1 The Atomic Relay Algorithm 57
4.4.2 Preparation of TCB-DNS Master File 60
4.4.3 Verification of RRSets 62
4.4.3.1 Events at ANS with TM A 62
4.4.3.2 Events at the PNS with TM P 63
4.4.3.3 At the Stub-Resolver C 66
4.4.4 Proof of Correctness 67
4.5 Practical Considerations 68
4.5.1 Ideal TMs 70
4.5.2 Leveraging TPMs 72
4.6 TCB-DNS vs. DNSSEC 74
4.6.1 Authenticated Denial 76

المنارة للاستشارات
www.manaraa.com

4.6.2 Overhead 79
4.6.3 Replay Attacks 81
4.6.4 DNSSEC with TSIG 81
4.6.5 NSEC3 Opt-out 83

5. MINIMAL TCB FOR DATA DISSEMINATION SYSTEM 86

5.1 Organization 87
5.2 A Generic Data Dissemination System 87
5.2.1 Related Work 89
5.2.1.1 Limitations of Existing Approaches 91
5.2.2 Salient Features of the Proposed Approach 92
5.2.2.1 Query-Response Authentication 93
5.2.2.2 Opportunistic Shared Secrets Between Users 94
5.2.2.3 Potential Applications 94
5.2.3 TCB Functions 95
5.2.3.1 Conveying User Secret 96
5.2.3.2 Inserting and Deleting IOMT leaves 97
5.2.3.3 Updating Records 99
5.2.3.4 Querying Records 101

6. MINIMAL TCB FOR SCADA SYSTEM MONITOR 105

6.1 Problem Statement 106
6.2 Minimal TCB 107
6.2.1 Principle of Operation 108
6.3 Background 109
6.4 Trustworthy SCADA Monitor 110
6.4.1 Overview of Proposed Approach 111
6.4.2 Example of the Utility of the Proposed Approach 112
6.4.3 Auxiliary Sensor Data 114
6.4.3.1 Initializing Sensor Data 115
6.4.4 Updating a Sensor Record 116
6.4.5 Interface Update() 117
6.4.6 Interface FProof() 118

7. EVAULATING SCADA SYSTEM STATE 120

7.1 Principle of Operation 120
7.2 Sensor leaves and Synthetic Leaves 121
7.3 Operation of module 123
7.3.1 Module Interfaces 124

المنارة للاستشارات
www.manaraa.com

8. A SECURITY ARCHITECTURE FOR SCADA SYSTEMS 127

8.1 Overview of STCB Approach 128
8.1.1 STCB System Components 130
8.1.2 Evaluating $\mathcal{F}()$ 132
8.1.2.1 Merkle Trees in the STCB Approach 133
8.1.3 STCB Designer and Deployer 134
8.2 STCB Design 135
8.2.1 STCB Design Tree 135
8.2.2 Inputs and Outputs of $\mathcal{U}_i$ 136
8.2.2.1 Synthetic Sensors 138
8.2.2.2 Constants and Look-Up Tables 139
8.2.3 Instruction Set $\mathcal{A}$ 140
8.3 STCB Deployment 141
8.3.1 STCB Operation 144
8.3.1.1 STCB Interfaces 146
8.4 STCB Architecture 147
8.4.1 Module Registers 148
8.4.2 Initializing Peer Parameters 150
8.4.3 Self Certificates 151
8.4.3.1 Binary Tree Certificates 151
8.4.3.2 Offset Certificates 152
8.4.4 Initializing STCB Modules 154
8.4.5 Sensor and State Reports 156
8.4.6 Sensor Updates and Incremental State Evaluations 158
8.5 STCB Protocol 160
8.5.1 Generation of Offset Certificates 161
8.5.2 Generating Static Binary Tree Certificates 162
8.5.3 Initialization and Regular Operation 162
8.5.4 STCB Design Example 166

9. CONCLUSIONS AND FUTURE RESEARCH 171

9.1 Contributions 171
9.2 Future Research 172

REFERENCES 174

APPENDIX

A. STCB INSTRUCTION SET 181

A.1 Opcodes 183

المنارة للاستشارات
www.manaraa.com

A.1.1 Internal Functions 186
A.2 Illustration of STCB System Design 189
A.2.1 Design Steps 189
A.2.1.1 Valid and Invalid States 190
A.2.2 Design Tree Leaves 192
A.2.3 Instructions for Leaves PP and HH 193

المنارة للاستشارات
www.manaraa.com

LIST OF TABLES

4.1 Comparison of TCB-DNS and DNSSEC 79

8.1 Instruction Set for Thermal Plant 170

A.1 Memory Layout of **Y** 183

A.2 A Partial Listing of Opcodes and their Interpretation 184

A.3 Examples Illustrating 3-byte Instructions. 185

A.4 Karnaugh Maps for Water Tank System. 191

المنارة للاستشارات
www.manaraa.com

LIST OF FIGURES

2.1 The domain name system (DNS) tree 7

2.2 SCADA System Architecture 19

2.3 A Binary Merkle tree -16 leaves. 32

3.1 PMC Model 36

3.2 Simplified Trusted Computing Model 39

3.3 Security Model 42

4.1 Atomic Relay Function 58

4.2 Original Configuration 69

4.3 Bump-in-the-Wire (BITW) Implementation 71

8.1 STCB components 131

8.2 Information flow in the STCB model 131

8.3 Static Descriptor ξ_{sp} – a specification for an STCB system 143

8.4 Example: Simplified version of thermal power plant 166

A.1 Internal function $f_{exec}()$ 186

A.2 Internal function $f_{eoc}()$ 188

A.3 Water tank SCADA system 190

المنارة للاستشارات
www.manaraa.com

CHAPTER 1

INTRODUCTION

The functioning of modern societies rely heavily on the functioning of several complex systems. Some such systems are seen as *critical*, as their failure can have a significant impact on the security of a nation. Examples of critical infrastructure systems include power grids, water supply systems, oil and gas production/transportation systems, mass transportation systems, and increasingly, the Internet.

From the perspective of beneficiaries of a system, the primary requirement for any system is to effectively provide services. From a perspective of the providers/operators of the system, the system should be profitable, and inexpensive to operate/maintain. In pursuit of effective and profitable systems, some systemic risks to the operation of the system are often ignored.

The components of complex systems are often themselves complex systems. Very often, in the design and operation of any system, it is simply assumed that all components of the system can be trusted to faithfully perform their assigned tasks. More specifically, while the design of many systems do often cater for *accidental* failure of components, failures resulting from carefully orchestrated attacks on components are ignored.

www.manaraa.com

The likelihood of hidden malicious/accidental functionality in any hardware/software component, especially in complex components of questionable provenance, is a serious threat to the security of critical national infrastructures. The very importance of crucial systems make them good targets for attackers — both domestic and foreign. With every passing day, more and more exploitable vulnerabilities in various components of critical infrastructure systems are discovered. Patching such vulnerabilities may not always be easy, or even possible (especially in legacy systems). Furthermore, such patches/fixes themselves may introduce fresh vulnerabilities.

1.1 Hypothesis and Research Goals

Due to the substantial risks posed by failure of critical infrastructure systems, we simply can not afford to blindly trust all components of the system. For any system with a desired set of assurances, the *trusted computing base* (TCB) [35] is a small amount of hardware/software that needs to be trusted in order to realize the desired assurances. It is crucial, especially for critical systems, to clearly identify a *minimal set of components* that constitute the TCB for the system, and focus our efforts on ensuring that the TCB is indeed *worthy* of trust.

The hypothesis of the proposed research is that resource limited hardware modules capable of performing only logical and hash operations will be able to serve as the TCB for the critical infrastructures. The main goals of the proposed research are

1. Identifying the minimal TCB for some sample critical infrastructure systems, in terms of a set of simple functions executed inside the confines of trustworthy hardware modules.

المنارة للاستشارات
www.manaraa.com

2. Identifying efficient mechanisms to amplify the trust in the TCB in order to provide the desired assurances.

As examples of critical infrastructure systems, we address the domain name system (DNS) [43] which is vital to the functioning of the Internet, and Supervisory Control and Data Acquisition (SCADA) systems [60] responsible for controlling various critical infrastructure systems.

1.2 Specific Contributions of this Dissertation

At the core of the proposed approach is a unified view of systems as constituted by providers of data, *untrusted* middle-men, and consumers of data. Under the provider-middlemen-consumer (PMC) model, most components of any system to be secured are assumed to be associated with the untrusted middle-men.

For purposes of arriving at a functional specification of the TCB components our research began by investigating various applications/system under the PMC model. Some of the characteristics of the system that were considered for classification of various systems under this model included

1. dynamics of the provider (static or dynamic number of data sources)
2. dynamics of data (does data have a pre-specifiable life time, or is it necessary to prematurely invalidate some data) and
3. the nature of the tasks performed by middle-men (does middle-men merely relay data to consumers or do they have to process data before making them available to consumers)

Under the PMC model, the domain name system (DNS) can be classified as one with dynamic number of providers, and static data (DNS records), where the middle-men simply relay data. SCADA systems can considered as composed of static providers of data

www.manaraa.com

(sensors and actuators of SCADA systems) which provide dynamic data (current sensor/actuator states) to middle-men (SCADA system components) that monitor the state of the SCADA system. The middle-men perform some system dependent tasks to determine the state of the SCADA system and report the state of the system to the consumer (stake-holders of the SCADA system).

A characterization of the minimal TCB for the domain name system was obtained, leading to the publication of a journal article [71]. Specifically, the TCB was identified to be a stateless atomic relay function.

That DNS employs static DNS records is a well appreciated limitation of DNS. If DNS records are dynamic, then the TCB functionality required to assure the integrity of DNS records needs to be state-ful. More specifically, even while the module executing the TCB functions is constrained to possess very little memory, it is required to "track" potentially unlimited amounts of data. This was catered for by adding to the TCB functionality, the ability to maintain an index ordered Merkle tree described in Section 2.3.2. A generic TCB for a dynamic *look-up* server was published in a conference proceeding [70].

For effectively monitoring the state of a SCADA system a first pre-requisite is to ensure *completeness* of inputs — or that a fresh snapshot of current states of all sensor/actuators are available. For this purpose the TCB functionality was expanded to include the ability to maintain an ordered merkle tree [69]. A conference paper outlining this approach was presented in [69].

Once completeness of inputs is guaranteed, the next step is to ensure that the system dependent algorithms for computing the overall state of the system (the output provided to

www.manaraa.com

the stake-holder, based on sensor states) can be executed inside a resource limited trustworthy boundary. Due to the fact that the number of the inputs could be different for different SCADA systems, it is necessary to possess mechanisms to handle *any* number of sensor inputs. This was handled by representing sensor data as leaves of a merkle hash tree [42]. The concept of "synthetic sensors" was introduced represent states that are functions of multiple sensor states. A conference paper outlining this approach was presented in [68].

Our work in [69] and [68] merely addressed some components of the TCB functionality required to secure SCADA systems. A comprehensive security architecture for SCADA systems was outlined in [72]. This architecture, based on a trusted hardware module — which we refer to as a SCADA TCB (STCB) module — is intended to be usable for any SCADA system, irrespective of the nature and size of the system. The STCB based security architecture includes specifications for a) the functionality of STCB modules; b) processes to be adopted by the *designer* and the *deployer* of the system; and c) an STCB protocol, for updating the state of STCB modules, and obtaining SCADA state reports.

The rest of the document is organized as follows:

Chapter 2 includes a brief introduction to DNS and SCADA systems and a survey of current efforts to secure such systems. The importance of identifying a minimal TCB for critical infrastructure systems is discussed in Chapter 3. Chapters 4 and 5 outline work towards securing DNS. Chapters 6 and 7 outlines work towards identifying TCB components for SCADA systems. Chapter 8 outlines a comprehensive architecture for securing SCADA systems. Chapter 9 offers conclusions and discusses future research.

المنارة للاستشارات
www.manaraa.com

CHAPTER 2

LITERATURE SURVEY

2.1 Domain Name System

The domain name system is a tree-hierarchical naming system for services that can be accessed over the Internet. At the top of the inverted DNS tree (see Figure 2.1) is the *root*. Below the root are generic top level domains (gTLD) like com, org, net, edu, etc., and country-code top level domains (ccTLD) like ca (Canada), in (India), etc. A leaf named b.cs.univ.edu in the DNS tree is a server-host in a branch cs.univ.edu, which stems from a thicker branch univ.edu, which stems from an even thicker branch .edu, stemming from the root of the DNS tree.

A branch of the tree (including its sub-branches and leaves) under the administrative control of an authority, is a DNS zone. The authority for a zone is responsible for i) assigning names for branches and leaves under the zone; ii) creating DNS resource records corresponding to such names, and/or iii) delegating an entity as the authority for a branch within the zone.

The authority for the root zone has delegated a gTLD zone like .edu to a .edu-gTLD authority, who has in turn delegated the zone univ.edu to another authority, who may have delegated a zone cs.univ.edu to yet another authority (say Z). All DNS records for the zone cs.univ.edu (or all DNS records with names ending with cs.univ.edu)

المنارة للاستشارات
www.manaraa.com

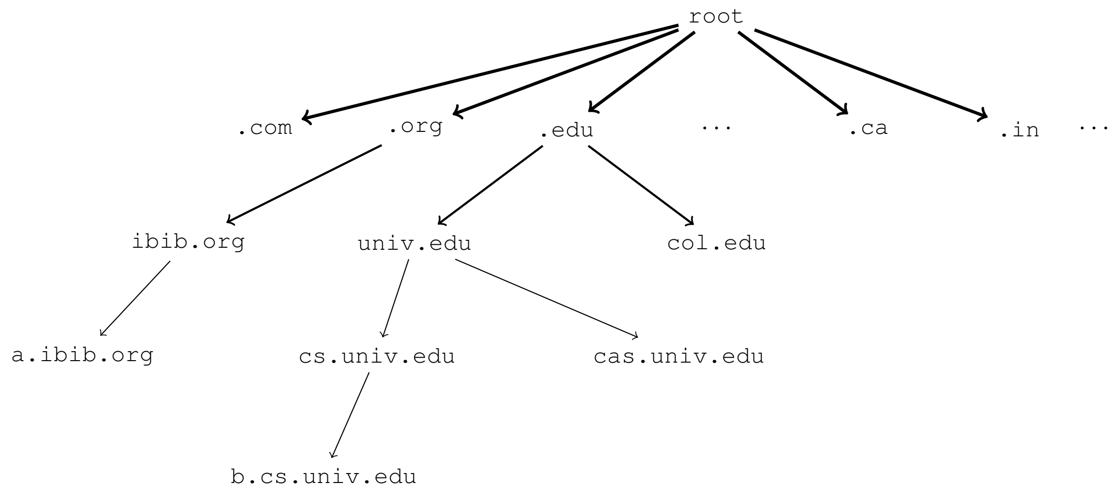

Figure 2.1

The domain name system (DNS) tree

are created by Z. The zone authority also specifies the names of *authoritative* name servers (ANS) for the zone. A "zone master file" which includes the set of all DNS RRs pertaining to the zone is then provided every ANS of the zone. ANSs of the root zone are also referred to as *root name-servers*.

A client desiring to access a HTTP service `www.cs.univ.edu` requires the IP address of the web-server with a domain name `www.cs.univ.edu`. This information is in an A-type record in the master file for a zone under which the name `www.cs.univ.edu` falls, and can be obtained by querying any ANS for the zone. To obtain this information, the querier only needs to know the IP address of a root name-server. While the root name server cannot directly provide the answer to the query "`www.cs.univ.edu, A`," it can provide the names and IP address of the ANSs for gTLD and ccTLD zones. In this case, the root server will respond with the names and IP addresses of all `.edu` ANSs.

المنارة للاستشارات
www.manaraa.com

The querier can now send the same query to any of the `.edu` ANS, which will respond with the name and the IP address of the ANSs for the zone `univ.edu`. When the same query "`www.cs.univ.edu, A`" is the directed to an ANS for the zone `univ.edu`, the response includes the names and IP addresses of ANSs for the zone `cs.univ.edu`.

Finally, any of the ANSs for the zone `cs.univ.edu` is queried to obtain the desired the A-type record. If the zone `cs.univ.edu` had not been sub-delegated, then the ANS for the zone `univ.edu` would have directly provided the response. Thus, knowing only the IP address of one root name server, any one can obtain any DNS record by specifying the name and type, and performing a series of queries.

2.1.1 DNS Records

Every DNS resource record (RR) is a five-tuple consisting of i) name, ii) class, iii) time-to-live (TTL), iv) type, and v) value; for example, `name=www.cs.univ.edu, IN, TTL=2345, type=A, value=159.43.7.82`. The class is always IN (for Internet RRs); the field TTL is specified in seconds, and indicates how long a RR can be cached. In the rest of this document, to keep notations simple, we shall ignore the fields "class" and "TTL."

A-type RRs indicate an IP address in the value field. An NS-type record `name=cs.univ.edu, type=NS, value=ns1.dserv.net` indicates that a name-server with a domain name `ns1.dserv.net` is an ANS for the zone `cs.univ.edu`.

A set of records with the same name and type, but with different value fields, is collectively referred to as an RRSet. For example the NS-type RRSet for the name `cs.univ.edu`

المنارة للاستشارات
www.manaraa.com

may include two NS records - one indicating the ANS ns1.dserv.net, and the other indicating another ANS named ns1.cs.univ.edu.

The NS type records are used for delegation. An RRSet of NS records for a delegated zone (say) cs.univ.edu can be found in the master file of the parent zone univ.edu. Similarly the NS RRSet for univ.edu can be found in the master file of the zone .edu, and so on. Along with NS records which specify ANSs, the A-type records for the ANSs are also included in the master file as *glue* records[1].

The creator of RRs for a zone, viz., the zone authority, is always off-line. Once the master file for zone has been provided to the ANSs, and the names of ANSs conveyed to authority of the parent zone (and included as NS records in the master file of the parent), the zone authority simply expects the ANSs to faithfully accept and respond to DNS queries regarding the zone.

2.1.1.1 Query-Response Process

DNS queries and responses are typically payloads of UDP packets and have the same packet format. They include a header, and four sections: QUESTION, ANSWER, AUTHORITY and ADDITIONAL. In a query packet QUESTION section indicates the queried name and type (all other sections are empty). The response has an identical QUESTION section. The ANSWER section contains the desired RRSet. The AUTHORITY section includes

[1]Note that zone univ.edu (or even edu) cannot be authoritative for the zone dserv.net. Thus, while univ.edu can provide an authoritative response regarding the *name* of the ANS for the child zone cs.univ.edu, it cannot provide an authoritative A-type record for the server ns1.dserv.net. To avoid possible circular dependency problems, the necessary *non authoritative* A-type records are included as glue records.

www.manaraa.com

NS records indicating the authoritative zone and the ANS for the zone. The `ADDITIONAL` section contains A-type glues for the NS records.

In practice, clients initiate queries using *stub-resolvers* running on their own host machine. Stub-resolvers do not directly query ANSs. Instead, they use preferred name-servers (PNS) as intermediaries. PNSs are also referred to as local DNS servers or local recursive resolvers or caching-only name-servers, and are typically operated by Internet service providers (ISP).

An application requiring the IP address of (say) `www.cs.univ.edu` queries a stub-resolver running on the same host. The stub-resolver redirects the query to a PNS. To do so, the host (or the stub-resolver) should know the IP address[2] of at least one PNS.

All PNS needs to be aware of the IP address of at least one root server. The PNS queries a root-server for `(www.cs.univ.edu,A)`, and receives NS records (with glued A-type records) for ANSs of `.edu`. The PNS then queries a `.edu` ANS to receive NS records of `univ.edu`, and so on. Finally, an ANS of the zone `cs.univ.edu` responds to the query with the desired A-type RRSet, which is relayed back to the stub-resolver.

PNSs may cache RRs for a duration specified by the TTL field in the RR, and may respond to queries from stub-resolvers using cached RRs. Similarly stub-resolvers may also cache RRs and respond to queries from applications running on the same host using the cached RRs.

[2]Typically, IP addresses of PNSs are provided to a host by a DHCP server. In UNIX-like machines the IP addresses of PNS are stored in a file /etc/resolv.conf.

المنارة للاستشارات
www.manaraa.com

2.1.2 Securing DNS

The main goal of attacks on DNS is to simply divert traffic away from genuine services, or more often, to divert such traffic to impersonators phishing for personal information from unsuspecting clients. A common strategy for attackers is to impersonate ANSs to provide fake DNS responses to PNSs, thereby "poisoning the cache" of the PNS, and consequently the caches of many stub-resolvers which employ the poisoned PNS.

The header of a DNS query includes a 16-bit transaction ID t_{id}; the UDP packet carrying the query indicates a 16-bit source port p chosen by the querier. A DNS packet carrying the query will be accepted only if it is addressed to port p. The DNS response in the UDP packet will be accepted only if it indicates an expected transaction ID t_{id}.

To create a fake response that will be accepted by an PNS, an *out-of-path* (or external) attacker, who does not have plain-sight view of the query packet, will need to guess the values t_{id} and p. A typical strategy for an out-of-path attacker is to register a domain, run her own ANS for the domain, and query the targeted PNS for a name under her domain. When the query from the PNS is ultimately directed to the attacker's ANS, the attacker learns enough information to narrow down the two values t_{id} and p within small range.

Recently, Kaminsky [32] pointed out that DNS cache poisoning attacks can have even more severe consequences. Instead of attempting to poison RRs corresponding to a specific zone, the attacker can impersonate a root server and send fake glue records for "IP addresses of gTLD name servers." Thus, queries to every `.com` zone, for example, will then be directed to a computer under the control of the attacker, which could redirect such queries to other "ANSs" under her control.

المنارة للاستشارات
www.manaraa.com

2.1.3 Link-Security Approaches

While properly randomizing the two 16-bit values (t_{id} and p) is a good first step, they offer no defense against *in-path* attackers. In-path attackers who may be in the same LAN as the server or the resolver, or lie in-between the resolver and the server, have plain-sight access to the values t_{id} and p in the UDP DNS packets, and can thus easily fake responses. Securing links between DNS servers (for example, by using a secret shared between a resolver and the server queried by the resolver) can prevent such attacks.

Specifically, two entities A and B who share a secret K_{AB} can prevent even in-path attackers from impersonating them by i) encrypting the message sent over the link using the shared secret K_{AB}, or ii) appending a message authentication code (MAC) $h(V \parallel K_{AB})$ where $h()$ is cryptographic hash function, and V may be the message, or a cryptographic hash of a message $\mathcal{M}$ (or $V = h(\mathcal{M})$); as long as $h()$ is pre-image resistant, only an entity with access to the secret K_{AB} can compute a valid MAC for a message.

Strategies like SK-DNSSEC [56] and DNSCurve [12] adopt such an approach. In symmetric key DNSSEC [56] all PNSs have the ability to establish secure channel with the root servers. ANSs higher in the hierarchy act as trusted servers and facilitate establishment of secrets with ANSs lower in the hierarchy, using the Needham-Schroeder protocol [45] (which is the basis for the Kerberos [46] authentication protocol). When a PNS queries the root server for "`cs.coll.edu`, A", the root server's response includes a Kerberos-like ticket which permits the PNS to establish a secure channel with a `.edu` DNS server. The `.edu` DNS server then issues a ticket for securely communicating with an ANS for the zone `coll.edu`.

المنارة للاستشارات
www.manaraa.com

DNSCurve [12] employs a Diffie-Hellman scheme over a special elliptic curve $\mathcal{C}$ for setting up a private channel between DNSCurve enabled DNS servers. A DNSCurve enabled server A chooses a secret a.

The secret between two DNSCurve enabled servers/resolvers A and B (where B's secret is b) is $K_{AB} = \mathcal{C}(b, \alpha) = \mathcal{C}(a, \beta)$, where $\alpha = \mathcal{C}(a, S)$, $\beta = \mathcal{C}(b, S)$, and S is a public parameter.

Link-security approaches assume that the DNS servers themselves are trustworthy. Note that while link-security approaches protect DNS RRs from out-of-path attackers (who do not have access to values t_{id} and p) *and* in-path attackers (those with access to t_{id} and p), there is nothing that prevents an entity controlling the DNS server from modifying an RR. In practice, such an attacker can be the operator of a DNS server, or some other entity who has somehow gained control of the DNS server. Such an attacker can receive RRs over protected links, illegally modify RRs, and relay fake RRs over "protected" links.

2.1.4 DNSSEC

Ideally, the "middle-men" should not be trusted: only the authority of a zone should be trusted for providing information regarding the zone. This is the approach taken by DNSSEC [9], where every RRSet in the zone master file is individually signed by the zone authority.

Every DNSSEC-enabled zone authority has an asymmetric key pair. The public portion of the key pair is certified by the authority of the parent zone. For example, the public key of the zone `cs.univ.edu` is signed by the authority of zone `univ.edu`. The public

المنارة للاستشارات
www.manaraa.com

key of the zone cs.univ.edu can be obtained by querying for a DNSKEY-type RR for the name cs.univ.edu. To authenticate the public key in the DNSKEY RR, the parent zone univ.edu introduces two RRs in its zone file: a delegation signed (DS) RR which indicates a key-tag (a hash) for the public key of its child, and an RRSIG(DS) record which is the signature for the DS record.

For verifying the RRSIG(DS) record the public key of the parent zone univ.edu is required - which is the DNSKEY RR for the name univ.edu. To authenticate the public key of the parent, it is necessary to obtain the DS and RRSIG(DS) record from *its* parent zone - .edu, along with the DNSKEY RR for .edu. Finally, the public key of .edu can be verified by obtaining DS and RRSIG(DS) records from the root zone (by querying any root server). The public key of the root zone is assumed to be well publicized.

To summarize, for every RRSet in the zone cs.univ.edu is a RRSIG(RRSet) record which contains the digital signature for the RRSet. In response to a query for an RRSet, the corresponding RRSIG record is also included in the response. To verify the RRSIG, the required DNS RRs are

1) DNSKEY RR of cs.univ.edu

2) DS, RRSIG(DS) corresponding to DNSKEY RR of cs.univ.edu, and DNSKEY RR of the parent zone univ.edu (fetched from the parent zone univ.edu);

3) DS, RRSIG(DS) corresponding to DNSKEY RR of univ.edu, DNSKEY RR of .edu;

4) DS, RRSIG(DS) corresponding to DNSKEY RR of .edu, from the root zone.

المنارة للاستشارات
www.manaraa.com

2.1.5 Authenticated Denial

Consider a scenario where the zone authority for the domain `wesellstuff.com` outsources its DNS operations to `dnsnet.net`. It is indeed conceivable that a competitor `wealsosellstuff.com` could bribe some personnel in `dnsnet.net` (or any entity who has acquired control over the ANS) to remove the record for `wesellstuff.com` (thereby driving the competitor out of business).

To ensure that DNS servers and/or their operators need not be trusted, DNSSEC demands a pertinent response from an ANS for *every* query that falls under the zone. If the queried name exists, the ANS should provide a signed RRSet. If the queried name does *not* exist, the ANS is expected to provide *authenticated denial* by providing some information signed by the *zone authority*[3] which demonstrates that the queried record does not exist. If the ANS ignores the query, or provides a non pertinent response, the resolver will send the query again, or will query another ANS for the zone, till it receives a pertinent response.

For example, in response to a query for name `abc.xyz.fgh` the querier expects a signed RRSet by the authority for the zone under which the name `abc.xyz.fgh` falls, or alternately, expects

1) a signed response from the authority of the root zone that no record for a name `.fgh` exists; or

2) a signed response from the authority of the zone `.fgh` that no record for the name `xyz.fgh` exists; or

[3] Only the zone authority is trusted to provide information regarding the zone - even information indicating that a record does *not* exist.

المنارة للاستشارات
www.manaraa.com

3) a signed response from the authority of the zone `xyz.fgh` that no record for the name `abc.xyz.fgh` exists.

As the zone authority is off-line, a response denying every possible (as yet unknown) query, regarding the almost infinitely many possible names and types that can fall under the zone, should somehow be signed by the zone authority and included in the master file provided to ANSs. This is accomplished cleverly through NSEC records [76]. A signed NSEC record `abc.example.com, NSEC, cat.example.com` indicating two *enclosers* is interpreted as an authenticated denial of all enclosed names: viz., names that fall between `abc.example.com` and `cat.example.com` in the dictionary order. For example, if queried for a record named `cab.example.com`, this NSEC RR signed by the authority of the zone `example.com` (the signature included in a RRSIG(NSEC) RR) is proof that no such record exists.

2.1.5.1 DNS-Walk

Even while DNS RRs are not meant to be private they should only be provided when explicitly queried by name and type. NSEC permits one to query random names and learn about unsolicited names of enclosers that *do* exist in the zone master file. For example, a querier may send a query for a random name like `axx.example.com` and get to know the two enclosers `abc.example.com, cat.example.com` that actually exist. The attacker can then query for a random name like `cate.example.com` and obtain its enclosers, say `cat.example.com, data.example.com`, and so on.

المنارة للاستشارات
www.manaraa.com

The ability to easily enumerate all services under a zone is obviously a useful starting point for any attacker. An attacker wishing to obtain all DNS records for a zone can easily "walk-through" all records in the zone master by simply making a sequence of random queries. Such supercilious queries also have the ill-effect of further burdening the DNS infrastructure.

2.2 SCADA systems

Process Control Systems (SCADA systems) play a major role in present day critical infrastructures like power grid, water management, petrochemical, oil and natural gas distribution systems etc. While modern technologies provide solutions for better performance, management, reliability of systems, they also expose the critical infrastructure to crucial attacks which could endanger lives of people along with economy.

2.2.1 SCADA System Components

A typical SCADA system includes the following components:

Master Terminal Unit (MTU): A master terminal unit is a server that is supervised by some trained personnel through a Human Machine Interface (HMI). A MTU usually comprises of racks of programmable logic controllers (PLCs). PLCs are programmable, configurable systems that run continuously. The PLCs read inputs from some addressed interfaces, processes the inputs and writes the output to some addresses. The MTU is responsible for i) collecting the data from different Remote Terminal Units (RTUs) and ii) sending commands to them through a wide variety of communication channels (Ethernet/wireless/radio/proprietary). A HMI is software that runs on the MTU provides a graph-

المنارة للاستشارات

www.manaraa.com

ical representation of the supervised system, and is also a central point for configuring the functionality of the SCADA system - either manually or automatically.

Remote Terminal Unit (RTU): RTUs are generally on the field close to the sensors which sense various physical processes. The sensors on the field report the data to RTUs. Most RTUs house some PLCs and analog-to-digital (A-D) converters. Typically RTUs are periodically polled by MTUs for collecting data, and for sending dynamic instructions/commands. In turn, PLCs at the field send commands to actuators on the field to control some physical process.

Sensors: A sensor is an analog or digital device that measures some parameter of a physical process and provides analog or digital data to an RTU.

Actuator: An actuator is a mechanical device for moving or controlling a mechanism or system. It takes energy, usually transported by air, electric current, or liquid, and converts that into some kind of motion [14].

Alarm: Alarms play a major role in SCADA systems by notifying the personnel about any hazardous situations and/or failure of devices etc. Alarm conditions are programmed in PLCs that poll an alarm in continuous intervals. A PLC sends a command to raise an alarm when an abnormal condition is met.

A typical SCADA system consists of all the above mentioned subsystems which collectively work for some very critical systems like Oil/ Gas/ Water / Electricity distribution where failure could result in hazardous effects on surroundings.

المنارة للاستشارات
www.manaraa.com

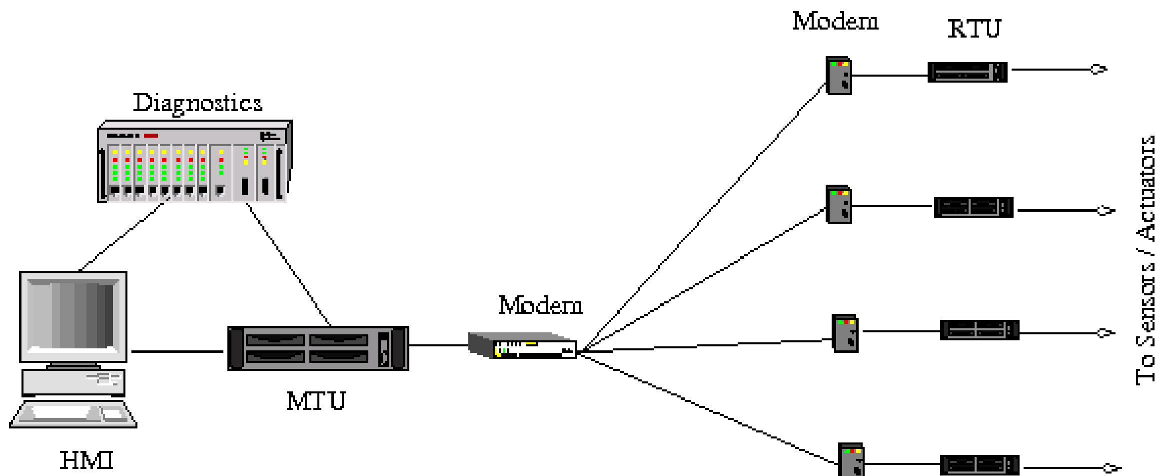

Figure 2.2

SCADA System Architecture

2.2.2 SCADA Security

Several issues concerning our ability to secure SCADA systems have been addressed by many researchers for various types of SCADA systems [60], [30], [1], [51]. Practical solutions to securing SCADA systems have been marred by i) the lack of well accepted standards (both for hardware and communication protocols), and ii) the uniqueness of each system, demanding specific domain knowledge for addressing security issues in each system.

For many industrial control systems security is not seen as a priority, or even a feature. Vendors who develop the control systems for the clients do not provide security unless it is asked for. Clients on the other end do not request for security features as the risk assessments do not show up any security related risks. Many legacy SCADA systems do

المنارة للاستشارات
www.manaraa.com

not employ any explicit security mechanism on the assumption of an established secure perimeter through isolation, and their obscure nature.

The SCADA systems that are used today have different sensor devices, data acquisition and processing units such as RTUs/ PLCs at field level and servers off the field usually monitored through HMI software by maintenance personnel. The unauthenticated data sent by the devices to a control system (a RTU/PLC) is delivered to the Master Terminal Unit through communication channels like Ethernet/Radio modem. As the communication channel between the master terminal unit and a field device do not typically employ any kind of authentication, it can be subjected to spoofing attacks like man in the middle attack and message injection attacks.

Apart from a wide range of easy attacks on the communication channels, several security loopholes can be exploited by attackers to gain control of the supervisory system. For example, operators of a plant may use e-mails, through which Trojan horses and worms can be planted in applications that can manipulate data or send commands [48]. More generally, such loop-holes include defects in the SCADA software, bugs in the underlying operating system, and even untrusted hardware in general purpose computers which run the software.

Efforts to secure SCADA systems have also been marred by its substantially different requirements, compared to better know information technology (IT) systems [18]. "SCADA systems and corporate IT systems focus on very different information assurance priorities. IT systems priorities are confidentiality, authentication, integrity, availability, and non-repudiation, SCADA systems on the other hand emphasize reliability, real-time

المنارة للاستشارات
www.manaraa.com

response, and tolerance of emergency situations, personnel safety, product quality, plant safety, usually to the exclusion of any security mechanism that might hinder these [18]". It is no surprise that standard IT security mechanisms are not appropriate for all SCADA systems as each system has its own requirements and dependencies. Over "90% of major SCADA and automation vendors have all of their manuals and specifications available on-line to the general public [48] ". Notwithstanding the fact that firewalls are used to isolate the SCADA LANs from the Internet, it is not impractical for a cyber attacker who invests time to learn the specifications from readily available manuals to create a far reaching disasters on critical infrastructures. In particular, the human machine interface (HMI) software which servers as the nerve center for many SCADA applications runs on untrusted general purpose platforms, and is often reachable over a wide area network. Relying on the integrity of this nerve center to control the system is a dangerous approach.

2.2.3 Vulnerabilities

In a SCADA system several individual components (sensors, actuators, RTUs, MTU , etc.) are bridged through a communication network. While each component plays a prominent role in effective functionality of the entire system, any component that is vulnerable to common cyber attacks will have a cascading effect on the whole system. For most of the SCADA networks are protected by software firewalls that act as perimeter [61], a simple misconfiguration of firewall can easily open up back doors for various components connected through the network.

المنارة للاستشارات
www.manaraa.com

Although the workstations inside the perimeter are protected by individual software firewalls, a rogue employee may plug a flash drive that carries worm/ virus. Apart from the firewalls, the critical infrastructure systems are monitored by intrusion detection systems IDS that continuously monitor for any malicious activity over the network. Most advanced IDS system can only watch for known vulnerabilities. The ineffectiveness of standardized security schemes in detecting worms like Stuxnet could cost a lot of fiscal damage.

Most of the SCADA systems demand high availability, reliability, timeliness. The vendors of the SCADA systems compete more over the mentioned demands and ignore the security measures like confidentiality, integrity despite of their usage in critical infrastructure. The SCADA vendors have little to no response to the known critical public exploits. A very less number of administrators care about patching their systems against exploits. A study done by ICS-CERT shows 60% [6] of failure rate in patching process.

Different layers of a Distributed Control System (DCS/ SCADA) system have distinct known vulnerabilities. These vulnerabilities can be broadly categorized as

1. Network Layer Vulnerabilities
2. Application Layer Vulnerabilities.

2.2.3.1 Network Layer Vulnerabilities

A SCADA network comprise of two major subnets

1. Corporate Network.
2. Control Network.

The components in the control network are entrusted with data acquisition from sensors/ actuators, data logging, data storage in historian, processing the data to issue commands

المنارة للاستشارات
www.manaraa.com

etc. On the other hand the business network / corporate network is involved with the IT issues like management, planning, customer service, inventory control etc. A proper communication channel wired/ wireless is in place for data exchange between both of these subnets. For effective monitoring vendors also provide remote access to the system, which today can be done using a mobile app over a smart phone.

Several different entry points to the control / corporate network area usually guarded by a single or multi-layered firewall. In a typical scenario, the control network is protected by two firewalls,

1. Corporate network firewall
2. Control network firewall

The corporate network is connected to internet which indirectly connects every component in the control network to the internet, but through one or more firewalls and IDS systems. Any component in control network (SCADA) that is virtually connected to internet is prone to all the known / unknown cyber attacks probably with an increased level of difficulty to penetrate for attackers.

The most widely used RTU technology in SCADA systems vendors utilize the dial up feature offered to perform a mass / individual update of firmware on these components. Gaining access to vendor's internal resources would reduce the complexity for an attacker to compromise the components in the SCADA system [3]. The communication schemes widely in practice in today's industry are a. MODBUS, b. DNP3, c. Zigbee.

The communications protocol widely used in legendary SCADA systems- MODBUS does not offer any security features. Even the most recent protocol DNP3 offers little to no

المنارة للاستشارات
www.manaraa.com

security. The DNP3 protocol is mostly used by utilities like power industry for communication of master and remote telemetry units. Zigbee is a wireless communication standard for SCADA systems that is gaining momentum with little evaluation of its security. In this protocol a smart energy device is tied with a certificate with serial number and manufacturer. Each device authenticates itself to other using its private key based on the asymmetric key cryptography [11]. Authors in [27] show different tools like KillerBee [81] to expose the flaws in this communication protocol.

2.2.3.2 Application Layer Vulnerabilities

The SCADA systems are integrated with various software applications and hardware peripheral devices at application layer such as

1. Database systems like SQL Server for data historian, that are used to store/ retrieve data for reporting, trending and analysis.
2. Human Machine Interface (HMI) that provides entire / partial state of the system.
3. Remote desktop applications for remote control/ monitor.
4. Hardware includes USB, CD-ROM, multimedia cards etc.

The SCADA systems depend on data historian for trending and analysis. The data collected by the devices is saved to and retrieved from a database. For performing operations on the database a language SQL (Structured Query Language) is widely used by industry. These databases return a response (a record / set of records) for a query issued as a command. The SQL based databases are prone to SQL injection attacks, if they are not maintained properly [82].

المنارة للاستشارات
www.manaraa.com

HMI is an interactive interface for administrator and other personnel who monitor the SCADA system. Highly complex systems like Nuclear reactors, Air traffic control systems rely on a HMI to visualize the state of entire system at any given instance. For ease and accessibility, many vendors of this software provide remote control applications to monitor the system via an app on a mobile. While the feature provides easy access, it stands out as huge vulnerability. An adversary only requires the login information to remotely access the HMI application and issue false commands to destabilize the system.

The access points like workstation's peripheral devices like USB ports, CD-ROMs, other device ports are highly vulnerable to malware injections, further gaining control over the system. Very strict access control mechanisms are required to avoid such attacks.

2.2.4 Attacks on SCADA Systems

With high levels of security and well architectured networks, SCADA systems stood one of the primary targets of cyber attackers. Attackers are able to enter the core of the system through various entry points of the system. Most of the attacks surfaced recently are successfully performed on critical infrastructure like nuclear plants, water management systems etc.

In 2010, a virus known as Stuxnet - specifically targeted towards nuclear plants to shutdown the centrifuges inside the plant and overwrite the setpoints of pressure and other values was undetected for more than a year duration [40]. Stuxnet was able to use about twenty zero day vulnerabilities [75] to gain control over the plant. Stuxnet has the potential to turn off pumps, control actuators, and still report that everything is normal. Now, the

المنارة للاستشارات
www.manaraa.com

popularity of this worm made it available on many blogs and websites. Availability of such powerful worms could lead to more drastic attacks upon re-engineering.

In November 2011, the Illinois Statewide Terrorism and Intelligence Center reported a cyber-attack on a small, rural water utility outside Springfield. The report stated that few attackers have gained access to the system and controlled the pumps remotely. " A hacker calling himself "Prof" posted screen shots from his computer showing him logged onto the control system of a water utility in the Texas town of South Houston" [77].

In May 2003, Slammer worm related to unpatched version of Microsoft SQL surfaced and resulted in loss of data. The data acquisition server was infected through the corporate network when an employee installed the software on his laptop which is the primary entry point for Slammer worm [44] . The Slammer worm along with other virus has affected a major petroleum company that resulted in remarkable financial damage [65].

Hidden malicious/accidental functionality in any SCADA system component could be exploited by an attacker to launch attacks such as the above. Such hidden functionality could exist in (the logic programmed into) programmable logic controllers (PLC) in RTUs and MTUs, in any computer used to run SCADA software for programming PLCs, or in any peripheral of the computer running the HMI software or the SCADA data logger, in the operating system of such computers, in the HMI software, or even, ironically, in a computer that runs the intrusion detection system (IDS) intended for protecting the SCADA system.

It is indeed for very good reasons that such threats have been recognized as "Advanced Persistent Threats" [7, 23, 28, 55, 78]. Due to the high value of targets, the possibility of sophisticated state sponsored attacks have to be considered. Sophisticated malicious func-

للاستشارات المنارة
www.manaraa.com

tionality may be introduced even during the manufacturing process of various components that could ultimately end up in SCADA systems. In addition, we can not afford to ignore the possibility that an attacker may have actually participated in the deployment of the SCADA system, or testing of the deployed system, and taken advantage of such an opportunity to inject hidden functionality in some component.

2.2.5 Related Work

In the effort towards securing SCADA systems, many security mechanisms are proposed. The emphasis on providing the critical infrastructure requirements like reliability, timeliness, availability, etc., overshadows the importance of security. Many secure strategies developed for critical infrastructure suggest better end to end security and other schemes of similar nature. A simple attack like turning off an alarm remotely, can invite more devastating undetected attacks. The fact check that " Does the system adhere to the rules laid by its designers, If not, is it detectable ?" is ignored in most of the security schemes for critical infrastructure.

2.2.5.1 Collaborative, Trust-Based Security Mechanism

Current research towards protection of the critical infrastructure focuses more on communication security and less on the protection of entire system. In a recent work done by G. Coates and K. Hopkinson, a trust-based security architecture is proposed for securing a utility network [18]. In their methodology, a trust system is created and added in strategic locations to protect existing legacy architectures, to enhance security. The trust system is built with the following characteristics.

المنارة للاستشارات
www.manaraa.com

1. TRUST system is a communication security device, with firewall and intrusion detection capabilities, designed for use with time-critical systems.
2. Trust system, is a software agent performing active security analysis and response.
3. In a network where nodes have sufficient unused hard drive capacity, memory, and processing power, the agent would be loaded directly onto the node.
4. The agent then provides an active interface between incoming messages and the nodes code, data, and applications,The emphasis on providing the critical infrastructure requirements like reliability, timeliness, availability, etc., overshadows the importance of security. Many secure strategies developed for critical infrastructure suggest better end to end security and other schemes of similar nature. A simple attack like turning off an alarm remotely, can invite more devastating undetected attacks similar to other software firewalls.
5. The trust system intercepts status messages or commands from network nodes.
6. The trust system validates input, identifies risks and bad data, and initiates appropriate alerts.
7. Trust system thwarts the illegitimate data that does not obey the policies.
8. Trust system provides data filtering feature to guard data from unauthorized personnel.

This software agent is simply a collaboration of all the general purpose security features like Intrusion detection system, firewalls, access control matrix etc. Authors advise strategic locations like electronic security perimeter (NIST) [61] for installing this agents. While such type of trusted agent may be useful for data sanitization and improving the firewall, IDS, and access control mechanisms, it might not be useful in detecting a change of setpoint made by an attacker, or cannot detect anything about a modification of "operation rule" by an unauthorized personnel.

2.2.5.2 DNP3 Secure Authentication

In an effort to secure the communication over DNP3 protocol, "the user group of DNP3 protocol for SCADA communication has adopted an extension to DNP3 called Secure Au-

المنارة للاستشارات
www.manaraa.com

thentication that meets IEC 62351-5 requirements. This extension is now part of the DNP3 protocol and is also part of IEEE 1815, the IEEE version of DNP3" [11]. In this scheme, the master station and the RTU share a secret, called update key. For a communication session between master terminal and RTU a session key is required for mutual authentication. The update key is used to compute a session key. After establishing the session secret, master station or RTU can initiate a challenge-response mechanism to validate authenticity of sender of the message. "The standard specifies that any message can be challenged, but all critical messages must be. Critical messages are essentially those that perform some kind of "write" operation" [11].

2.2.5.3 IEEE SCM

Another ongoing effort towards security in SCADA systems is the IEEE SSPP (Serial SCADA Protection Protocol). This standard defines a security protocol for control system serial communication. "The fundamental objective of SSPP is to ensure the integrity of SCADA messages, that messages are not forged, modified, spliced, reordered or replayed. With an appropriate cipher suite SSPP also provides confidentiality" [2].

The IEEE SCM (SCADA Cryptorgraphic Module) implements a three layer Serial SCADA Protocol Protection (SSPP) for accomplishing its goals - providing link security using a BITW approach. At the top is the session layer responsible for formatting methods and data to be sent in sessions, and for distinguishing between different kinds of messages and performing session key negotiation and exchange. The transport layer provides cryptographic support for messages passed by the session layer to preserve confidentiality and

المنارة للاستشارات
www.manaraa.com

integrity. The link layer formats the output of the transport layer into octets suitable for transmission over a communication link.

Ideally, the tasks performed by the SCM should be limited to bare minimum. As the security requirements for cryptographic operations are different from the requirements for the performance of auxiliary tasks, they should not be combined in the same module.

2.3 Merkle Hash Tree

A binary Merkle hash tree is constructed using a cryptographic hash function $h()$ (like SHA-1). A tree with $N = 2^L$ leaves has a height L. Let the leaves of the tree be $l_1 \cdots l_N$. Let $v_i = h(l_i), 1 \leq i \leq N$ be represent N nodes in level-one of the tree (one corresponding to each leaf). The next level (level 2) of the tree consists of $N/2$ nodes, each obtained by hashing a pair of nodes in level 1. At level 3 are $N/4$ nodes obtained by paired hashing of the $N/2$ nodes in level 2, and so on. Level $L-1$ has two nodes, and level L has a single node - the root r of the tree. The root of the tree is a commitment to all leaves.

Corresponding to any leaf l_i is a set of L intermediate nodes $\mathbf{v}_i$ (one from each level) such that $r = h'(l_i, \mathbf{v}_i)$, where the function $h'()$ represents a sequence of hash operations. Specifically the set of hashes $\mathbf{v}_i$ are intermediate nodes that can be seen as commitments *to all leaves except* l_i. If $h()$ is pre-image resistant, it is infeasible to "cook-up" a set of values l'_i, and $\mathbf{v}'_i$ that satisfies $r = h'(l_i, \mathbf{v}_i)$. Thus, if provided with values l_i and $\mathbf{v}_i$ satisfying $r = h'(l_i, \mathbf{v}_i)$, the entity that possesses r can rest assured that l_i is indeed a legitimate leaf of the tree and $\mathbf{v}_i$ are indeed valid intermediate leaves of the tree.

المنارة للاستشارات
www.manaraa.com

Just as any leaf can be independently verified against the root, it is also possible to update each leaf independently. For example, if l_i is updated to l'_i the root can be updated to $r' = h'(l'_i, \mathbf{v}_i)$ to reflect the new leaf l'_i. After the root has been updated, the old leaf l_i can no longer be demonstrated to be a part of the tree.

2.3.1 Merkle Tree Limitations

Two well known limitations of the merkle hash tree are i) the power of two requirements for the total number of leaves; and ii) the ability to readily recognize non existence of a record. The implication of the first limitation is inefficiency in scenarios where the total number of leaves N is not a power of 2. Specifically, if 513 records need to be stored, then a tree of length 1024 need to be maintained.

The second limitation is however more serious. For the module (which maintains only the root) to be convinced that no record regarding some index a exists, the module should verify the integrity of every leaf and in this process, deduce that no leaf for a exists. Obviously, this is far from practical. That the module can *not* verify non existence can be abused to perform replay attacks.

As a specific example, consider a scenario where some new information is available regarding a record for some index a (and thus needs to be updated). However, the untrusted server (which stores all records) can incorrectly claim that no information exists currently for a, and request the new information for a to be added as a new record. After this, as both the old and new records are part of the tree, the server has the ability to advertise either record.

www.manaraa.com

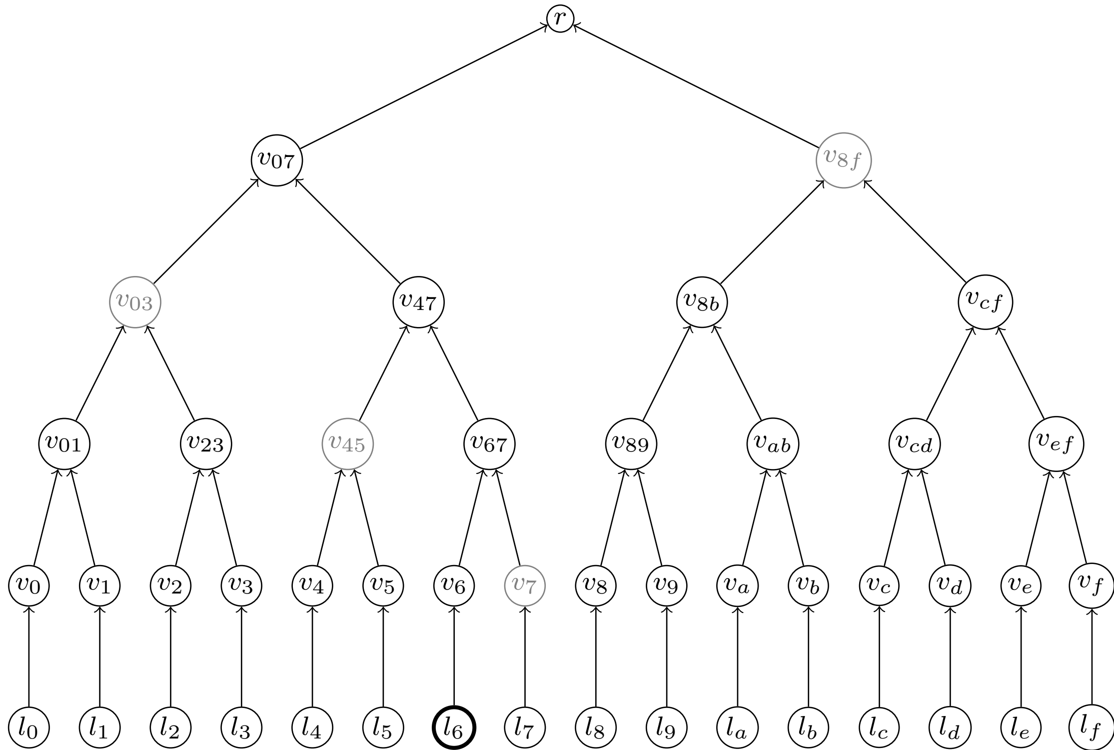

Figure 2.3

A Binary Merkle tree -16 leaves.

للاستشارات المنارة
www.manaraa.com

2.3.2 Index Ordered Merkle Tree

The index ordered merkle tree is a simple modification to the merkle tree which addresses both limitations of the merkle tree. In an IOMT a leaf $\mathbf{L}_i$ (the i^{th}leaf in the IOMT) associated with data index a is of the form $\mathbf{L}_i = (a, v_a, a')$ where the middle value v_a is the data associated with index a, and a' is the *next* data-index.

That a leaf $\mathbf{L}_i = (a, v_a, a')$ can be verified by the module to be consistent with the root implies that i) a leaf exists for index a, and ii) no leaf exists for any index that falls between a and a'. The set of uniquely indexed current leaves is an ordered list where every index points to the next higher index; the exception is the highest index (address) which wraps around and points to the least index.

A value x is *covered* by (a, a') if $(a < x < a')$, or (for the wrapped around pair) if $(x < a' \leq a)$ or $(a' \leq a < x)$. If $a = a'$ *all* values except a are covered, and implies that a is the *only* index.

The issues associated with the power of 2 requirement is addressed by simple modifications to hash functions $H_L()$ used for deriving a leaf node from an IOMT leaf and $H_V()$ used for combining two nodes to obtain a parent node. In an IOMT empty leaf is of the form $\mathbf{L}_i = (0, 0, 0)$. The function $v_i = H_L(\mathbf{L}_i)$ which maps a leaf to a leaf node is defined as

$$v_i = H_L(i, v_i, i) = \begin{cases} h(i, v_i, i') & \text{if } i \neq 0 \\ 0 & \text{if } i = 0 \end{cases} \tag{2.1}$$

المنارة للاستشارات
www.manaraa.com

The function $H_V(u, v)$ which maps two internal nodes to a common parent is defined as

$$p = H_V(u, v) = \begin{cases} u & \text{if } v = 0 \\ v & \text{if } u = 0 \\ h(u, v) & \text{if } u \neq 0, v \neq 0 \end{cases} \tag{2.2}$$

Consequently, an IOMT with a root 0 can be seen as a tree with *any* number of zero leaves of the form $(0, 0, 0)$.

By performing some simple checks the IOMT ensures that only one leaf can exist for an index. Specifically, a leaf with an index c can be inserted only if can be demonstrated that no leaf with index c exists currently by providing a leaf that covers c. Specifically, to insert a leaf for an index c two leaves need to be provided: i) an empty leaf $(0, 0, 0)$ and ii) a leaf for some other index a - say (a, v_a, a') such that (a, a') covers c. After insertion the two leaves will be modified to (c, v_c, a') and (a, v_a, c) respectively. To insert a leaf with index a when the root is zero, the root is simply set to $H_L(a, v_a, a)$. Similarly, when a leaf (x, v_x, x') needs to be deleted a leaf (b, v_b, b') should be provided such that $x' = b$. After deletion the first leaf becomes $(0, 0, 0)$ and the second becomes (b, v_b, x'). To delete a sole leaf (a, v_a, a) the current root $H_L(a, v_a, a)$ is set to 0.

Except for the case of insertion of the first leaf or deletion of a sole leaf, to insert or delete a leaf two leaves will need to be modified simultaneously. Two leaf hashes v_l and v_r can be simultaneously mapped to the root r by mapping the leaf hashes to the common parent, and then mapping the common parent to the root. Let v_p be lowest common parent of two leaf nodes v_l and v_r. More specifically, let $v_p = H_V(v_p^l, v_p^r)$ where v_p^l and v_p^r are the

المنارة للاستشارات
www.manaraa.com

left and right child of v_p. Let $\mathbf{v}_l$, $\mathbf{v}_r$ and $\mathbf{v_p}$ be a set of hashes satisfying $v_p^l = f(v_l, \mathbf{v}_l)$, $v_p^r = f(v_r, \mathbf{v}_r)$, and $r = f(v_p, \mathbf{v}_p)$. Thus,

$$r = f(H_V(f(v_l, \mathbf{v}_l), f(v_r, \mathbf{v}_r)), \mathbf{v}_p) \tag{2.3}$$

المنارة للاستشارات

www.manaraa.com

CHAPTER 3

TCB FOR UNTRUSTED MIDDLEMAN

The digital information age is characterized by the unprecedented ability of entities to disseminate and receive digital information using a wide variety of computing devices. Depending on the specific nature of the application, the information could take various forms like stock quotes, audio/video, blog, tweet, post, HTML document, file, domain name system (DNS) record, etc, and exchanged over a variety of communication channels. Irrespective of the nature of the information any such application can be seen as a set of information sources $\mathcal{P}$ and a set of consumers $\mathcal{C}$. Most often, as it is impractical for such sources and sinks to interact directly with each other, they rely on a middle-man - or more generally a set of middle-men $\mathcal{M}$.

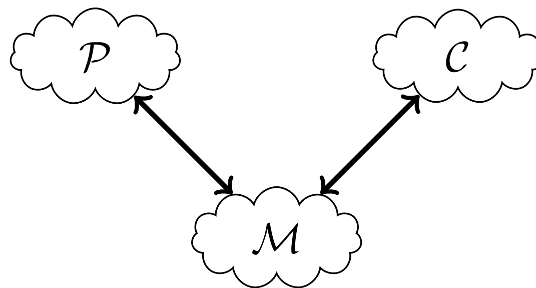

Figure 3.1

PMC Model

المنارة للاستشارات
www.manaraa.com

For example, In the process of disseminating DNS records [43] created by zone authorities to any Internet client, DNS servers are middle-men. In a SCADA system, the sensors are providers, monitor is a middleman, and consumers are actuators. Numerous network based applications that exist today, and perhaps several application paradigms that may evolve in the future can be characterized broadly under this provider-middleman-consumer (PMC) model.

While the issue of securing interactions between providers and consumers have received substantial attention, most research merely address the problem of securing the channels between i) providers and the middle-men, and ii) middle-men and consumers [12] , [74]. In doing so it is implicitly assumed that the middle-men are trusted. Most often a middle-man is a server - an untrusted software running on an untrusted platform, under the control of untrusted entities. The broad focus of the proposed research are security strategies to prevent potential abuses by identifying minimal TCB for middle-men.

3.1 TCB

A trusted computing base (TCB) [35] is a resource constraint set of software and hardware that serves as a base for securing a system amidst of untrusted components within the system.

It is not feasible to verify and certify a large complex application's software and hardware for its integrity. The verification complexity of a software/hardware increases with its size and complexity. In order to trust a TCB, the hardware and software that constitutes the TCB should be trusted. Any trusted system should be verifiable. As the verification

المنارة للاستشارات
www.manaraa.com

complexity increases with the size and complexity , it is important for a TCB to be as simple as possible with minimal software and hardware. The question is what is the minimal functionality to be offered by a TCB to amplify the trust and be verifiable.

3.2 Minimal TCB

Any security solution is essentially a strategy to realize some assurances relying on the integrity of a minimal trusted computing base (TCB). More specifically, the trust in a few components/entities that constitute the TCB is amplified using security protocols to achieve the desired assurances.

Typically, TCB functionality is offered by tamper-responsive trustworthy computing modules. Examples of trustworthy computing modules include

1. cryptographic co-processors entrusted with the task of protecting and performing computations using sensitive secrets; and
2. trustworthy platform modules (TPM), which offer the TCBs functionality in the TPM-TP model.

3.2.1 Read-proofing and Write-proofing

Just as the guaranteed TCB functionality is leveraged to provide some assurances regarding the security of the system, for realizing the guaranteed TCB functionality, two fundamental assurances are leveraged.

The first is read-proofing of secrets protected by the module. This is necessary to ensure that the modules cannot be impersonated.

المنارة للاستشارات
www.manaraa.com

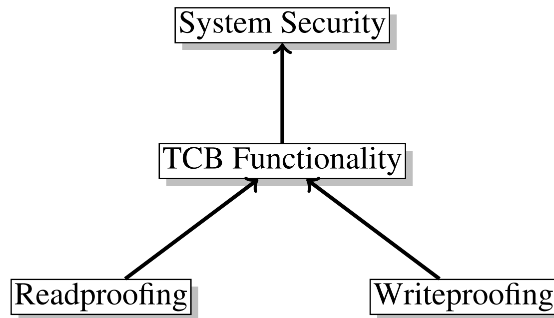

Figure 3.2

Simplified Trusted Computing Model

The second is write-proofing of software executed by the module. The software dictates the TCB functions. Write-proofing software is necessary to ensure that the TCB functions cannot be modified.

The two requirements are however not independent. With the ability to modify software at will, an attacker can force the module to reveal its secrets (for example, by inserting a set of commands to write the secret bytes out to the serial port). On the other hand, secrets that are protected can be used to authenticate software that will be executed by a computer in the module. Without the knowledge of the secret the attacker cannot modify the software (more specifically, such modifications will be detected due to failure of authentication).

In practice read-proofing is seen as a stepping stone to the more elusive goal of tamper-proofing software. Attacks aimed at modifying software to reveal secrets can be prevented by ensuring that software does not have access to at least some of the secrets that are protected. Some secrets may be *generated, stored and used* by dedicated hardware [54], [37]. However, authenticating software with the secrets provides a boot-strapping problem [67].

المنارة للاستشارات
www.manaraa.com

After all, some software should be loaded (typically the BIOS) which includes instructions to load the secret and carry out the steps required to perform authentication.

3.3 TCB Models

In the trusted computing group (TCG) [58] specification for trusted platforms, trusted platform modules (TPM) [59] are used to attest measures of loaded software, thereby permitting third parties to verify that only authorized software has gained control of a platform. To leverage TPMs to realize a trusted platform it is (unfortunately) required to trust several other components of a general purpose computer like CPU, BIOS, RAM, CPU-RAM bridge, and possibly other peripherals that have direct access to the RAM. Practical deployments of TCG trusted platforms have been hindered by several factors like i) several attacks that have been identified to violate the envisioned security goals of the TCG architecture [57]; and that ii) thorough review necessary to pre-certify software (BIOS, boot-loader, operating system and application software) is far from practical.

The "late launch" feature in the current version of TPMs (version 1.2) in conjunction with new instructions supported by Intel and AMD processors (Intel-TXT and AMD-SKINIT) [41] together provide an assurance that a "small piece of application logic (PAL)" [41] can be run unmolested on a general purpose platform - by trusting only the CPU, RAM, a CPU-RAM bridge. More specifically, the BIOS and other peripherals do not have to be trusted, and all other software that can take control of the platform need not be verified/measured. Unfortunately, some attacks have been found against the late launch architecture recently [79].

المنارة للاستشارات
www.manaraa.com

Another approach is the use of IBM 4758 [54] trustworthy computing module which sports a general purpose processor inside a protected boundary, runs a specialized operating system, and can execute special application code unmolested inside the trusted boundary. In such an approach the minimal TCB for a SCADA monitor can be executed inside such modules.

In yet another approach, Chavez et al [17] realize a trust anchor through the use of secure obfuscation techniques to run protected code. Specifically [17] suggests the use of such a trust anchor for a SCADA monitor. However, in the preliminary work reported in [17] does not describe the precise nature of the tasks performed by the trust anchor.

3.3.1 Security Model

In the envisioned security model an untrusted middle man $\mathbf{U}$ is associated with a trusted module $\mathbf{T}$. The trusted module is assumed to be severely resource limited, and performs a set of simple tasks $\mathcal{T}$. By performing tasks $\mathcal{T}$ module $\mathbf{T}$ provides some important assurances regarding the tasks performed by $\mathbf{U}$.

Data provided by providers to the middle-man $\mathbf{U}$ are assumed to be public, and are authenticated by providers for verification by $\mathbf{T}$. Data sent by middle-men to consumers are authenticated by $\mathbf{T}$ to vouch for the integrity of the data. However, the module $\mathbf{T}$ will vouch for the data only if untrusted $\mathbf{U}$ can prove to $\mathbf{T}$ some properties about the data. Thus, the purpose of the tasks $\mathcal{T}$ performed by the module is simply to *verify* the proof submitted by $\mathbf{U}$.

المنارة للاستشارات
www.manaraa.com

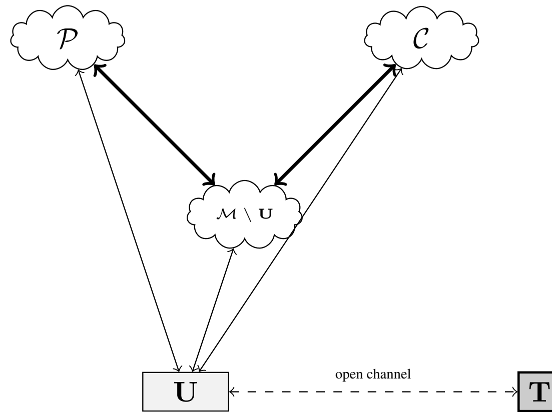

Figure 3.3

Security Model

3.4 Critical Infrastructures from the perspective of module T

Numerous CI applications that fall under the provier-middleman-consumer (PMC) model can be classified into several categories depending on the nature of data, the set of data identifiers, and the specific tasks performed by middle-men.

3.4.1 Static vs Dynamic Data

Data pertaining to an identifier may be *static* or *dynamic*. Static data has a well defined expiry period (which can be infinite) and cannot be revoked prematurely; dynamic data can be revoked prematurely. An example of static data is an address-type DNS record (with a pre-specified life-time). In scenarios where it is impractical to foresee the validity duration of data, it is necessary to treat the data as dynamic. An example of dynamic data is routing information; the route to some destination may change prematurely due to unforeseen changes in the network topology.

المنارة للاستشارات
www.manaraa.com

3.4.2 Static vs Dynamic Data Identifiers

The data identifiers may themselves be static or dynamic. For example, at the top level of the domain name system, the same set of DNS records (corresponding to top-level domains and country-code domains) are held by root servers. However at lower levels of DNS, as new zones could be introduced at any time, the identifiers are dynamic. Thus, the data at top-most level of DNS is an example of static data with static identifiers. Data at lower levels of DNS are examples of static data with dynamic data identifiers.

An example of a system with static identifiers and dynamic data is a SCADA system. In such a system a fixed number of sensors are information sources that provide dynamic measurements. A central monitor (middle-man) processes the data to check if the monitored system is in an acceptable state. If not, an alarm should be triggered. In this case the alarm module can be considered as the consumer of data.

3.4.3 TCB Tasks

Depending on the nature of application and functionality of middleman, the tasks of a TCB bound to middleman can be classified as a. Stateless, b. Stateful.

3.4.3.1 Stateless TCB

In several applications middle-men simply store and relay data. In such applications, on receipt of a query for a data with a specific identifier, the middle-men is expected to provide the requested data.

المنارة للاستشارات
www.manaraa.com

The modules associated with such middlemen do not have to be aware of the exact nature or format of the data. The modules do not store any data that is specific to the application and hence stateless. Examples of such applications include well know services like WWW, DNS, Email, and so on.

The stateless TCB approach does not suit the applications with dynamic data. For example, if a zone authority is allowed to add records, or a web service that provides updates of stock quotes etc. The Stateless TCB does not maintain any state of the system. In other words, a Stateless TCB does not store any data pertaining to the application.

Our approach towards securing DNS through a Stateless TCB by performing a simple atomic relay is provided in Chapter 4.

3.4.3.2 Stateful TCB

In some applications, when queried for a data corresponding to some identifier, middlemen may need to process data associated with multiple data identifiers to respond to the query. In such cases, in order to assure the operations performed by the middle-man, the module T may require to perform some additional application-domain specific functions. Examples of such services include web-brokering (matching a clients request with many possible service offerings), and SCADA systems (where the monitor has to check data from all sensors to decide if the system is in an acceptable state), and a dynamic lookup service (unlike static DNS, the data in the records is dynamic / expire prematurely).

Our approach towards securing SCADA systems through a TCB performing trusted process check is provided in Chapters 6, 7. A generic solution by using Stateful TCB

المنارة للاستشارات
www.manaraa.com

is provided for data dissemination systems with dynamic data, dynamic identifiers, and dynamic consumers is given in Chapter 5

www.manaraa.com

CHAPTER 4

ATOMIC RELAY FOR DNS

DNSSEC has seen poor levels of adoption as upgrading a "plain-old" DNS server to support DNSSEC will often necessitate a hardware upgrade due to an order of magnitude increase in the size of DNSSEC records (compared to plain DNS records), and substantial increase in the size of DNS responses [5]- [22]. In many cases DNSSEC may require more expensive TCP instead of UDP as the transport protocol for carrying large DNS responses. DNS resolvers and clients will also need to endure substantial computational burden due to the need to verify multiple digital signatures. Furthermore, the feasibility of zone enumeration also encourages attackers to perform supercilious queries, thus exacerbating the issue of high DNSSEC overhead.

4.1 Extending Link-Security Approaches

Cryptographic mechanisms for individually securing each link traversed by DNS records, viz., the links i) between the off-line zone authority and ANSs of the zone (for securely conveying master files); ii) between PNSs and ANSs; and iii) between clients and their PNSs, demand substantially lower overhead compared to the hierarchical PKI-like approach employed by DNSSEC. Unfortunately, link-security approaches implicitly assume that the

المنارة للاستشارات
www.manaraa.com

middle-men are trustworthy: while RRs are protected in transit, there is no protection for RRs while they reside in the DNS servers.

Specifically, symmetric key DNSSEC [56] and DNSCurve [12] simply *assume* that

1) the keys employed by DNS servers (which are used to compute the link secrets) are well protected from untrustworthy entities (else, any entity with access to the secrets of a DNS server can impersonate the DNS server to send fake RRs); and that

2) the intermediary DNS servers i) will not modify RRs, and ii) will not deny RRs that do exist.

The proposed TCB-DNS, where every DNS server houses a low complexity trustworthy module (TM), is also a link-security approach. However, TCB-DNS does not make such unjustifiable assumptions regarding the trustworthiness of DNS servers. Instead, TCB-DNS assumes that

1) secrets protected by the TM (which are used to compute link-secrets) cannot be exposed; and

2) the trivial functionality of the TM cannot be modified.

To warrant trust, an obvious requirement is that the TM functionality is as simple as possible. To motivation for TCB-DNS stems from the intuition that TMs that perform trivial functions are sufficient to provide the desired assurances **A1**, **A2** and **A3**.

4.2 Principle of Operation

In TCB-DNS every DNS server is equipped with a low-complexity TM. From the perspective of DNS servers, the TMs are black boxes that accept a formatted stream of bits

www.manaraa.com

as input, and output a message authentication code (MAC). Such MACs accompany plain DNS responses sent by DNS servers. The operations performed inside the TM (to map the input bits to a MAC) are a fixed sequence of logical and cryptographic hash operations. This simple TM functionality is the TCB of a DNS server, which is leveraged to realize all three assurances **A1** - **A3** with negligible overhead.

4.2.1 Atomic Relay

In the path of a RRSet originating from zone authority Z to the client (a stub-resolver C), are an ANS for the zone Z and the PNS used by the host C. An *atomic relay*, as the name suggests, relays a value from one entity to another, in one atomic step. A TM A in the ANS performs an atomic relay of a value V from the zone authority Z to a PNS TM P, thus eliminating the need to trust the ANS in which the TM A is housed. Similarly the TM P in the PNS performs an atomic relay of the value V from the ANS TM A to a stub-resolver C, eliminating the need to trust the PNS.

From the perspective of the TM A, it receives some input bits which specify the identity of the source Z, a value V to be relayed, a message authentication code (MAC) $M_{V,ZA}$, and the identity of the entity P to which the value V needs to be relayed. The TM A uses its secrets to compute pair-wise secrets K_{ZA} and K_{AP} (the precise mechanism for doing this is explained later Section 4.3.1). Using the pair-wise secret K_{ZA}, TM A verifies the MAC $M_{V,ZA} = h(V, K_{ZA})$ appended by Z. Following this, the TM A computes a MAC $M_{V,AP} = h(V, K_{AP})$ using the secret K_{AP}.

المنارة للاستشارات
www.manaraa.com

The values relayed by TMs are hashes of RRSets. The hashes of RRSets are computed by zone authorities and individually authenticated to each ANS TM using MACs. ANS TMs can atomically relay the hashes to any PNS TM which can then atomically relay the hash to any stub-resolver. The TMs thus provide a parallel channel for securely conveying hashes of RRSets by leveraging link-secrets (which are computed using secrets protected by the TM).

Note that in both DNSSEC and TCB-DNS end-to-end integrity of an RRSet is realized by securely conveying a pre-image resistant hash of the RRSet. In DNSSEC this is achieved by signing the hash. In TCB-DNS the integrity of the hash is assured to the extent we can trust the TMs involved in relaying the hashes.

DNSSEC provides assurance **A1** by signing hashes of regular DNS RRs, and provides assurance **A2** by signing hashes of NSEC/NSEC3 RRs. Obviously, by atomically relaying the hashes of regular RRSets and NSEC/NSEC3 records, TCB-DNS can also provide both assurances provided by DNSSEC. However, a simple addition to the capability of the TMs can provide TCB-DNS with yet another useful feature - the ability to provide assurance **A3**, and thereby eliminate the possibility of DNS-walk.

4.2.2 "Intelligent" Atomic Relay

If we merely relay hashes of NSEC/NSEC3 records, then TCB-DNS will also be susceptible to DNS-walk. Fortunately, to realize assurance **A3**, the only additional intelligent feature required of TMs is the ability to recognize that "a value V falls inside an enclosure (V_l, V_h)."

المنارة للاستشارات
www.manaraa.com

The atomic relay function performed by a TM with identity X, takes the form

$$M_{XD} = \mathcal{A}R_X(S, D, V, V_l, V_h, M_{SX}). \tag{4.1}$$

In executing this TCB function the TM X accepts some fixed-length inputs like i) S and D: the identities of a source and destination; ii) cryptographic hashes V, V_l, and V_h; and iii) a MAC M_{SX} provided by the source S. The TM outputs a MAC for the value V under two conditions:

1. the MAC M_{SX} is consistent with V; or
2. the MAC M_{SX} is consistent with values $V_l \parallel V_h$, *and* V is enclosed by (V_l, V_h).

In the latter case, the TM interprets a pair of values (V_l, V_h) authenticated by the zone authority as proof that no value enclosed by (V_l, V_h) exists in the master file. If V is enclosed, the TM outputs a MAC for V to inform D that an "enclosure for V was found."

In DNSSEC that a value V is enclosed by (V_l, V_h) is checked by the querier. The need to reveal the enclosures to the querier is the reason that assurance **A3** cannot be provided. In TCB-DNS the enclosure is checked by the ANS TM (**not** provided to the querier). To the extent that the TM can be trusted, the querier trusts that an enclosure exists for the value V (and consequently, is convinced that an RR with the name corresponding to V does not exist).

More specifically, in TCB-DNS,

1) if the queried name and type exists the response includes the desired RRSet in the `ANSWER` section; a MAC for a value V (where V is hash of the RRSet) is also included in the response.

المنارة للاستشارات
www.manaraa.com

2) to deny a name and type $n_i \parallel t_i$ the ANSWER section indicates the name and type $n_i \parallel t_i$; the MAC for the value $V = h(n_i \parallel t_i)$ is included in the response to imply that the indicated name and type does not exist.

Typically, to provide authenticated denial for a queried name-and-type, a plurality name-and-types will have to be explicitly denied (as will be explained later in Section 4.6.1).

4.3 Computing Link Secrets

For performing the atomic relay, a TM needs to compute two pairwise secrets - one shared with the sender (the previous hop), which is used to verify the hash V (or the encloser (V_l, V_h) for authenticated denial), and one shared with the destination (the next hop). Specifically,

1. ANS TMs require the ability to establish a pairwise secret with i) the zone authority for receiving hashes of RRSets, and ii) all PNS TMs for securely conveying hashes of RRSets.
2. PNS TMs require the ability to establish a pairwise secret with i) all ANS TMs for receiving hashes, and with ii) all clients who employ the PNS for conveying the hashes.

For reducing TM complexity it is essential to identify a low complexity mechanism for computing pairwise secrets. While many efficient strategies exist, the modified Leighton-Micali scheme (MLS) proposed in [49] is particularly well suited for establishing pairwise secrets between TMs in DNS servers (between a large number of ANS TMs and a large number of PNS TMs). We also extend MLS to facilitate link-secrets between i) zone authorities and ANS TMs and ii) between PNS TMs and clients.

المنارة للاستشارات
www.manaraa.com

4.3.1 MLS

Let K_X represent a secret privy only to an entity with identity X and a key distribution center (KDC). Similarly, let K_Y represent a secret known only to an entity Y and the KDC. In MLS, the secret shared between two entities X and Y is $K_{XY} = h(K_X, Y)$ *or* $K_{YX} = h(K_Y, X)$.

If the pairwise secret is $K_{XY} = h(K_X, Y)$, then entity X computes the pairwise secret by directly hashing its secret K_X; entity Y employs a public (non-secret) value $P_{YX} = h(K_X, Y) \oplus h(K_Y, X)$ to compute $K_{XY} = h(K_{XY})$ as

$$\begin{aligned} K_{XY} &= h(K_Y, X) \oplus P_{YX} \\ &= h(K_Y, X) \oplus h(K_X, Y) \oplus h(K_Y, X) \\ &= h(K_X, Y). \end{aligned} \tag{4.2}$$

On the other hand, if the pairwise secret is $K_{YX} = h(K_Y, X)$, entity Y computes the pairwise secret directly, and entity X employs the public value $P_{YX} = h(K_X, Y) \oplus h(K_Y, X)$.

MLS is an identity-based scheme, where the identity assigned to an entity can be chosen to reflect the credentials of the entity. For example, the identity of a zone authority can simply be the hash of the name of the zone. In MLS some bits of the identities (say of two entities X and Y) are also used to determine which of the two entities should employ the pair-wise public value.

المنارة للاستشارات
www.manaraa.com

4.3.2 Key Distribution for TCB-DNS

In TCB-DNS the KDC can be entity under the control of a regulatory authority (for example, ICANN). The *core* TCB-DNS entities are TMs associated with DNS servers. The *fringe* TCB-DNS entities include zone authorities (who need to securely convey RRs to ANSs) and clients (stub-resolvers) who query DNSs. Pair-wise secrets for TCB-DNS can be i) between two core entities (between two TMs), or ii) between a core entity and a fringe entity.

For the former case, a sequence number included in the TM identity specifies which of the two core entities should use the public value to compute the pairwise secret. For the latter (pairwise secret between a core entity and a fringe entity) the fringe entity employs the public value - the core entity does not.

The identity X of a TM (associated with an ANS or a PNS) is of the form $X = X_t \parallel q_x$ where X_t is a succinct code describing the nature of X and duration of validity; the value q_x is a unique number assigned sequentially to every DNS server TM. To establish a secret between TMs $X = X_t \parallel q_x$ and $Y = Y_t \parallel q_y$ where (say) $q_x < q_y$, Y is required to use the value P_{XY} to compute the pairwise secret K_{XY}; X can compute K_{XY} directly using its secret K_X.

The TCB-DNS identity Z of a zone authority is of the form $Z = Z_t \parallel Z_{name}$ where Z_{name} is a one-way function of the domain name of the zone. To enable Z to compute a pairwise secret K_{ZA} with an ANS TM A the zone authority is issued

1. a secret $K_{AZ} = h(K_A, Z)$ by the KDC, or
2. a secret K_Z, along with a public value P_{ZA}; or

المنارة للاستشارات
www.manaraa.com

3. a TM with identity Z (with secret K_Z stored inside the TM), along with a public value P_{ZA}.

In the TCB-DNS identity of a client $C = C_t \parallel C'$, C' can be a unique random value. If P is the identity of a TM in a PNS used by the client C the client C is issued i) a secret $K_{PC} = h(K_P, C)$ or ii) a secret K_C and a public value P_{CP}.

Thus, once keys have been distributed to TCB-DNS entities, computing any link-secret will require the TM to only perform a single hash computation (or a hash computation and an XOR operation). Periodically, the KDC disseminates signed revocation lists indicating identities of entities revoked.

Note that unlike the "basic" key distribution scheme for a static network of size N (where each node is issued $N - 1$ secrets) MLS can cater for a dynamic network - as new core entities (DNS server TMs) can be added at any time. In the basic scheme, to add a new node every old node should be provided a new secret, which is impractical. In MLS the new node is provided with one public value corresponding to every "old" node - old nodes do not need a public value to establish a secret with newly added nodes.

4.3.3 Multiple KDCs

At the top of the hierarchy of DNSSEC is a single root CA - which is the authority for the root zone. Though the root zone is expected to sign only public keys for gTLD and ccTLD zones, the all powerful root zone authority has the ability to misrepresent public keys for *any* zone. While ideally we would desire that this power be distributed amongst multiple independent entities, such an approach can further increase the overhead for DNSSEC.

المنارة للاستشارات
www.manaraa.com

However, MLS can be easily extended to support multiple KDCs with minimal overhead. If we use m (for example, $m = 4$) independent KDCs, an entity X receives m secrets (one from each KDC), $K_{X_i}, 1 \leq i \leq m$. Two entities X and Y can compute m independent pairwise secrets of the form $K^i_{XY}, 1 \leq i \leq m$ (one in each of the m parallel systems). All these secrets are simply XORed together to compute K_{XY} as $K_{XY} = K^1_{XY} \oplus K^2_{XY} \cdots \oplus K^m_{XY}$.

The secret K^i_{XY} can be computed only by X, Y, and the i^{th}KDC. The secret $K_{XY} = K^1_{XY} \oplus K^2_{XY} \cdots \oplus K^m_{XY}$ can be computed only by TMs X and Y (and together, by all m KDCs). While there are m public values associated with each secret, the m values can be XORed together and stored as one value; for example, $P_{XY} = P^1_{XY} \oplus \cdots \oplus P^m_{XY}$, where $P^i_{XY} = h(K_{X_i} \parallel Y) \oplus h(K_{Y_i} \parallel X)$.

Thus, a TM with identity X stores m secrets $K_{X_i}, 1 \leq i \leq m$ inside its protected boundary. To enable the TM to compute K_{XY}, the entity (DNS server) using the TM X provides two inputs: (Y, P_{XY}). The TM performs m hash operations and m XOR operations to compute

$$\begin{aligned} K_{XY} &= h(K_{X_1}, Y) \oplus \cdots \oplus h(K_{X_m}, Y) \oplus P_{XY} \\ &= K^1_{XY} \oplus \cdots \oplus K^m_{XY} \end{aligned} \tag{4.3}$$

In the rest of chapter we shall use the notation $K_{XY} = F(Y, P_{XY})$ to represent the process of computing the pairwise secret K_{XY} by entity (or TM) X.

www.manaraa.com

4.3.4 Renewal

For renewal of secrets of a TM $X = X_t \parallel q_x$ the regulatory authority simply issues a new TM with TCB-DNS identity $X' = X'_t \parallel q'_x$, with secrets $K_{X'_1} \cdots K_{X'_m}$. If at the time of renewal, the last issued sequence number was q, the new TM is issued a sequence number $q'_x = q + 1$. The owner of the TM is issued q public values (where each public value is the XOR of m public values). If the secrets of an ANS TM A is renewed, only the zone authorities using the ANS need to be issued new public values for A. If the TM P of a PNS is renewed, only the clients who use the PNS are issued with new public values corresponding to P.

More specifically, a node with sequence number q is the q^{th}node to join the network, and is issued one secret and $q - 1$ public values (or m secrets and $q - 1$ public values if we use multiple KDCs). For renewal we simply add a new node. The public values are the same size as the pair-wise keys (say 160-bits). A DNS server with a TM sequence number 10 million will need access to at most 200 MB of storage for public values (which can easily be stored in the hard-disk of the DNS server). There is no practical limit on the number of fringe entities (zone authorities and clients). Each fringe entity requires access only to a small number of public values (as they need to establish a pairwise secret only with a small number of core entities - zone authorities with TMs of all ANSs for the zone, and clients with all its PNSs).

المنارة للاستشارات
www.manaraa.com

4.4 The TCB-DNS Protocol

In this section we begin with a detailed specification of the atomic relay algorithm. We then we outline the operation of TCB-DNS by outlining the steps for creating TCB-DNS master files (in Section 4.4.2) and illustrating the sequence of events in typical a query-response process (in Section 4.4.3).

4.4.1 The Atomic Relay Algorithm

A TM with identity X stores a secret K_X inside its protected boundary - which is known only to TM X and the KDC. To relay a value from S to D the TM requires to compute secrets K_{XS} and K_{XD}. For this purpose the TM needs two additional inputs - public values P_{XS} and P_{XD}. Thus, the atomic relay function of a TM X takes the form

$$M_{XD} = \mathcal{A}R_X((S, P_{XS}), (D, P_{XD}), V, V_l, V_h, M_{SX})$$

In a scenario where X does *not* require to use a public value to compute K_{XS}, the input $P_{XS} = 0$ is provided to the TM (as XOR-ing by 0 leaves a value unchanged). It is the responsibility of the (untrusted) DNS server to store and provide appropriate public-values to its TM; if a DNS server provides incorrect public values to its TM the MAC will be rejected by the next-hop[1] which verifies the MAC.

The TM X accepts a formatted stream of bits $\mathbf{b}_i = (S \parallel P_{XS}) \parallel (D \parallel P_{XD}) \parallel V \parallel V_l \parallel V_h \parallel M_{SX}$ as input from the DNS server which houses the TM; the TM performs a simple sequence of logical and cryptographic hash operations, and outputs a MAC M_{XD}

[1]If the next-hop is a PNS, when an invalid response is received, the PNS will send the query again or query another ANS. Similarly if the next-hop is a stub-resolver C, then C will resend the query or query another PNS.

المنارة للاستشارات
www.manaraa.com

or a fixed constant ERROR.. An algorithmic description of the sequence of operations is depicted in Figure 4.1.

```
AR_X(S, P_XS, D, P_XD, V, V_l, V_h, M_SX) {
K_XD = F(P_XD, D);
IF (S == X);
   RETURN h(V || K_XD);
K_XS = F(P_XS, S);
IF (V_l == 0)
   V_i = V;
ELSE IF (((V_l < V) ∧ (V < V_h)) ∨ ((V > V_l) ∧ (V_l > V_h)))
   V_i = h(V_l || V_h);
ELSE RETURN ERROR;
IF (M_SX != h(V_i || K_SX));
   RETURN ERROR;
RETURN M_XD = h(h(S || V) || K_XD);
}
```

Figure 4.1

Atomic Relay Function

As shown in the algorithm in Figure 4.1, the TM computes the pairwise secret K_{XD} for authenticating TM output to destination D. If $S = X$ (source is indicated as the TM itself), the TM construes this as a request to output a MAC $h(V \parallel K_{XD})$ verifiable by D. This feature, as we shall see soon, permits zone authorities to use DNS TMs for protecting zone secrets.

المنارة للاستشارات
www.manaraa.com

In general (for $S \neq X$) the TM proceeds to compute the pairwise secret K_{XS} required for validating the inputs (V, V_l and V_h authenticated by source S using a MAC M_{SX}:

1. if V_l is zero the TM verifies that the MAC M_{SX} is consistent with V and K_{XS};
2. if the value V_l is non-zero, the TM verifies that i) the input MAC M_{SX} is consistent with the two values $(V_l \parallel V_h)$, *and* ii) that V is *enclosed* by (V_l, V_h). A value V is enclosed by (V_l, V_h) if $V_l < V < V_h$. If $V_l > V_h$ then V is enclosed by the "wrapped around" pair if $V > V_l > V_h$ or $V < V_h < V_l$.

On successful verification the TM outputs a MAC for the value $(S \parallel V)$ computed using the pairwise secret K_{XD} between X and D.

For ease of following the discussion in the rest of this section, note that

$$\begin{aligned} M_{ZA,V} &= \mathcal{A}R_Z(Z, 0, A, P_{ZA}, V, 0, 0, 0) \\ &= h(V \parallel K_{ZA}) \qquad (4.4) \end{aligned}$$

is a MAC for a value V computed by a TM Z (for verification by a TM A). We shall see soon that zone authorities can employ TMs in this fashion to authenticate hashes of RRSets for verification by ANS TMs. Also note that

$$\begin{aligned} M_{AP,V_Z} &= \mathcal{A}R_A(Z, 0, P, P_{AP}, V, 0, 0, M_{ZA,V}) \\ &= h(h(Z \parallel V) \parallel K_{AP}) \qquad (4.5) \end{aligned}$$

is a MAC computed by TM A which can be verified by an entity P. The MAC represents A's claim that "a value V was received from Z." If the MAC is verifiable, to the extent P trusts A, P can accept the claim that the value V was provided by Z.

Finally,

$$\begin{aligned} M_{AP,V'_Z} &= \mathcal{A}R_A(Z, 0, P, P_{AP}, V', V_l, V_h, M_{ZA,V'}) \\ &= h(h(Z \parallel V) \parallel K_{AP}) \qquad (4.6) \end{aligned}$$

المنارة للاستشارات
www.manaraa.com

is also a MAC verifiable by an entity P; on successful verification, P concludes that "a value V was received from Z." P does not need to differentiate between the two cases. In the former case, V was explicitly conveyed to A by Z through a MAC $M_{V,A}$. In the latter case, V is any value, not explicitly conveyed by Z, but happens to fall within an enclosure (V_l, V_h) (and the enclosure is authenticated by Z using MAC $M_{V',A}$).

4.4.2 Preparation of TCB-DNS Master File

Consider a zone `example.com`, which employs ANSs with TMs A and B for the zone. The sequence of steps performed by the zone authority to prepare a master file are as follows. Let the TCB-DNS identity of the zone be Z where $Z = Z_t \parallel Z_{name}$, where Z_{name} is the hash of the name of the zone (`example.com`). Recall that Z_t includes a succinct representation of the time of expiry of the secrets assigned to Z.

1 Prepare a regular plain DNS master file. Some of the required additions to plain DNS RRs are as follows:

1. Each RR will indicate an absolute value of time as the time of expiry. This value can be a 32-bit value like UNIX time, and can be different for each RR. In general the time of expiry of any RR should not be later than Z_t.
2. NS-type RRs (which indicate the name of an ANS) includes two additional values
   (a) the TCB-DNS identity of the ANS-TM, and
   (b) the value Z_t (note that from the name of the zone in the NS RR, one can compute the value Z_{name}; along with the value Z_t the TCB-DNS identity of the zone can be computed as $Z = Z_t \parallel Z_{name}$).

In general, a RRSet $\mathbf{R}$ has multiple RRs with the same name and type, and each RR indicates its own a time of expiry.

المنارة للاستشارات
www.manaraa.com

2 Let r be the total number of RRSets. For an RRset $\mathbf{R}$ with name n_j and type t_j compute i) the hash of the RRSet $v_j = h(RRSet)$; and ii) $u'_j = h(n_j \parallel t_j \parallel \tau)$, where τ is the time at which the authentication for all enclosures expire. Repeat for all r RRSets.

3 Sort the hashes $u'_1 \cdots u'_r$ in an ascending order; Let the sorted hashes be $u_1 \cdots u_r$. Now, compute values $d_1 \cdots d_r$ as

$$d_j = \begin{cases} h(u_j \parallel u_{j+1}) & j < r \\ h(u_r \parallel u_1) & j = r \end{cases} \tag{4.7}$$

Note that for the last "wrapped around" enclosure (u_r, u_1) the first value u_r is greater than the second (u_1).

4 For each of the $2r + 1$ values in $\{v_1 \cdots v_r, d_1 \cdots d_r, \tau\}$ compute MACs $M_{ZA,i} = h(v_i \parallel K_{ZA}), 1 \leq i \leq r$, $M'_{ZA,j} = h(d_j \parallel K_{ZA}), 1 \leq j \leq r$, and $M_{ZA,\tau} = h(\tau \parallel K_{ZA})$. If the zone authority Z employs a TM Z then a MAC like $M_{ZA,i} = h(v_i \parallel K_{ZA})$ is computed by using the atomic relay function of the TM as

$$\begin{aligned} M_{ZA,V} &= \mathcal{A}R_Z(Z, 0, A, P_{ZA}, v_i, 0, 0, 0) \\ &= h(v_i \parallel K_{ZA}). \end{aligned} \tag{4.8}$$

Prepare a supplementary master file with

1. the values $(\tau, M_{ZA,\tau})$;
2. r rows of the form $(i, M_{ZA,i}), 1 \leq i \leq r$, and
3. r rows of the form $((u_j, u_{j+1}), M'_{ZA,j}), 1 \leq j \leq r$,

5 Provide the supplementary master file to ANS with TM A along with the regular master file. The zone authority repeats step 4 for ANS B to create a supplementary master file with values $(\tau, M_{ZB,\tau})$; r rows $(i, M_{ZB,i})$, and r rows $((u_j, u_{j+1}), M'_{ZB,j})$.

المنارة للاستشارات
www.manaraa.com

4.4.3 Verification of RRSets

We shall consider a scenario where an ANS with TM A is queried for an RRSet `cad.example.com,A` by a PNS with TM P. Let us further assume that the query was initiated by a stub-resolver C.

4.4.3.1 Events at ANS with TM A

Let the identities A and P of the TMs be $A = A_t \parallel q_a$ and $P = P_t \parallel q_p$. If $q_p < q_a$ (sequence number of P is less than that of A) the ANS fetches P_{AP} from storage (else $P_{AP} = 0$). If the queried name and type (n_i, t_i) exists, or if a suitable delegation exists, the ANS

1. extracts the RRSet from the plain DNS master file, and computes the hash of the RRSet, v_i;
2. extracts the MAC $M_{ZA,i}$ for v_i from the supplementary master file;
3. requests TM A to compute

$$\begin{aligned} M_{AP,i_1} &= \mathcal{A}R_A((Z,0),(P,P_{AP}),v_i,0,0,M_{ZA,i}) \\ &= h(h(Z \parallel v_i),K_{AP}). \end{aligned} \quad (4.9)$$

In the response sent to the PNS, the ANS includes the RRSet in the `ANSWER` section along with the value M_{AP,i_1}. If the response is a delegation, the NS RRSet can be included in the `AUTHORITY` section along with the value M_{AP,i_1}. The TM A does not know, or care, if the response is an answer or a delegation.

To deny a name-and-type (n_i, t_i),

1. ANS extracts the values $(\tau, M_{\tau,ZA})$ from the supplementary master file for zone Z.
2. ANS computes $v_i = h(n_i \parallel t_i \parallel \tau)$;

المنارة للاستشارات
www.manaraa.com

3. ANS finds encloser for v_i (say (u_j, u_{j+1})), and corresponding MAC $M_{ZA,j}$ from the supplementary master file;

4. ANS requests TM A to compute M_{AP,τ_1} and M_{AP,i_1} as

$$\begin{aligned} M_{AP,\tau_1} &= \mathcal{A}R_A((Z,0),(P,P_{AP}),\tau,0,0,M_{ZA,\tau}) \\ &= h(h(Z \parallel \tau),K_{AP}) \\ M_{AP,i_1} &= \mathcal{A}R_A((Z,0),(P,P_{AP}),v_i,u_j,u_{j+1},M_{ZA,j}) \\ &= h(h(Z \parallel v_i),K_{AP}). \end{aligned}$$

For reasons that will be explained later in Section 4.6.1, typically the ANS will need to deny multiple name-and-type values in a response. Let us assume that q name-and-type values need to be denied. For each such name-and-type (n_l, t_l) the ANS computes $v_l = h(n_l \parallel t_l \parallel \tau)$, finds an encloser for v_l and the corresponding MAC in the supplementary master file, and requests the TM to compute MACs of the form M_{AP,l_1} (each of the q requests are made independently - each request results in the use of the atomic relay function $\mathcal{A}R_A()$ by the TM A).

In the response sent to the PNS the ANS includes (in the `ANSWER` section)

1. values τ and M_{AP,τ_1},
2. q denied name-and-type values $n_i \parallel t_i$,
3. q MACs of the form M_{AP,i_1}.

4.4.3.2 Events at the PNS with TM P

Before the PNS had sent a query to the ANS for a name and type belonging to zone Z, the PNS would have queried an ANS for the parent zone of Z - say $W = W_t \parallel W_{name}$, and obtained an NS-type RRSet for the name Z_{name} (where $Z = Z_t \parallel Z_{name}$).

المنارة للاستشارات
www.manaraa.com

Let us further assume that the NS-type RRSet was authenticated by a TM $G = G_t \parallel q_g$ (housed in an ANS for the zone W). In other words, the PNS would have received a value M_{GP,j_1} to authenticate the NS-type RRSet, where

$$M_{GP,j_1} = h(h(W \parallel v_j), K_{GP}), \tag{4.10}$$

and, v_j is the hash of the NS-type RRSet.

In TCB-DNS, the PNS is expected to verify the NS RRSet before sending a query to the delegated server. In this case, where the PNS had chosen to approach the ANS A for the zone Z (based on the information included in the NS-type RRSet authenticated by G) the PNS computes $v_{j_1} = h(W \parallel v_j)$, and requests its TM P to compute

$$x = \mathcal{A}R_P((G, P_{GP}), (A, P_{PA}), v_{j_1}, 0, 0, M_{GP,j_1}) \tag{4.11}$$

As long as $x \neq ERROR$, the PNS considers the NS records to be valid.

Similarly, prior to querying G (ANS for W, the parent of Z) the TM would have received a response from an ANS for the parent of W (unless W is the root zone which has no parent - in our case W is the gTLD zone `.com`). Such a response from an ANS of W's parent zone would have also been verified as above before a query was sent to G. Thus, after the response from the parent zone W was verified, the PNS P had sent a request to A for a name and type under zone Z.

TCB-DNS does not require queries to be authenticated. Queries merely indicate the TCB-DNS identity of the querier.

Now, after the response is received from A, the PNS P has all the necessary information to send the answer to the stub-resolver C which initiated the query.

المنارة للاستشارات
www.manaraa.com

Typically, the PNS will need to include an RRSet in the `ANSWER` section (along with a MAC computed by its TM P). For responses containing authenticated denial for q name-and-types the response will include $q+1$ values authenticated individually using $q+1$ MACs. For both cases, an NS-type RRSet will be included in the `AUTHORITY` section indicating ANS for the zone, along with a MAC computed by the TM P.

More specifically, the hash of the RRSet in the `ANSWER` section is relayed atomically from A to C, by P. Similarly, for authenticated denial, the q hashes corresponding to multiple non-existing names, and the value τ, are relayed atomically from A to C by P. The hash of the NS-type RRSet is however relayed atomically by P from G to C.

For example, to relay the RRSet with hash v_i received from A the PNS first hashes the RRSet to obtain v_i and requests its TM to compute

$$\begin{aligned} M_{PC,i_2} &= \mathcal{A}R_P((A, P_{PA}), (C, 0), v_{i_1}, 0, 0, M_{AP,i_1}) \\ &= h(h(A \parallel v_{i_1}) \parallel K_{PC}) \\ &= h(h(A \parallel h(Z \parallel v_i)) \parallel K_{PC}) \end{aligned} \tag{4.12}$$

If the hash of RRSet v_i computed by the PNS is not the same as the one authenticated by the zone authority Z, the MAC M_{AP,i_1} will be found inconsistent by the TM P, which will return $ERROR$.

Similarly, to relay the NS-type RRSet received from G along with a value M_{GP,j_1}, the PNS hashes the RRSet to obtain v_j, and uses its TM P to compute[2]

$$M_{PC,j_2} = \mathcal{A}R_P((G, P_{PG}), (C, 0), v_{j_1}, 0, 0, M_{GP,j_1})$$

[2]The value W is obtained from the NS type RRSet for the parent zone W, which was obtained by querying W's parent - the root.

المنارة للاستشارات
www.manaraa.com

$$= h(h(G \parallel v_{j_1}) \parallel K_{PC})$$

$$= h(h(G \parallel h(W \parallel v_j)) \parallel K_{PC}) \tag{4.13}$$

The response from the PNS to C thus includes

1. the NS-type RRSet (with hash v_j) for Z along with the values W, G and M_{PC,j_2} in the AUTHORITY section, AND

2. the queried RRSet with hash v_i, along with a MAC M_{PC,i_2}, and the identity A of the ANS, OR

3. authenticated denial of q name-and-type values (by including $q+1$ values and $q+1$ MACs), and the identity A of the ANS.

If the parent zone W does not support TCB-DNS (the ANS for W is not equipped with a TM) then the NS RRSet is relayed without any TCB-DNS authentication.

4.4.3.3 At the Stub-Resolver C

The stub resolver performs the following steps:

1 Extracts name of zone from the AUTHORITY section; hashes name to compute Z_{name} and hence $Z = Z_t \parallel Z_{name}$;

2 If the NS RRSet in the AUTHORITY section has TCB-DNS authentication

1. Client C computes the hash v_j of the NS-type RRSet in the AUTHORITY section and verifies that $M_{PC,j_2} = h(h(G \parallel h(W \parallel v_j)) \parallel K_{PC})$.
2. Parses W as $W = W_t \parallel W_{name}$ and verifies that W_{name} is a legitimate parent of Z_{name}.

3 C verifies that Z_{name} is a legitimate parent of the queried name.

المنارة للاستشارات
www.manaraa.com

4 C verifies that name of the zone is a parent of the queried name[3] in the ANSWER section;

5 If the ANSWER is the desired response, hash the RRSet to compute v_i; compute $v_{i_1} = h(Z \parallel v_i)$, $v_{i_2} = h(A \parallel v_{i_1})$, and using key K_{CP} verify that $M_{PC,i_2} = h(v_{i_2} \parallel K_{PC})$.

6 If the ANSWER is an authenticated denial indicating q values of the form $n_i \parallel t_i$, for each of the q values compute $v_i = h(n_i \parallel t_i \parallel \tau)$, $v_{i_1} = h(Z \parallel v_i)$, $v_{i_2} = h(A \parallel v_{i_1})$, and verify that $M_{PC,i_2} = h(v_{i_2} \parallel K_{PC})$.

RRs which have expired (as indicated by time-of-expiry field added to each RR in an RRSet) will be ignored. If the ANSWER section indicate authenticated denial and the value τ smaller than the current time, the response is ignored. If any of the TMs A and P and G involved in relaying the RRSets has been revoked by the KDC, the RRSet is ignored.

4.4.4 Proof of Correctness

Consider a scenario where the verifier C determines that the set of values $\{Z, A, v_i, M_{PC,i_2}\}$ satisfy

$$M_{PC,i_2} = h(h(A \parallel h(Z \parallel v_i)) \parallel K_{PC}). \tag{4.14}$$

In concluding that the RRSet (with hash v_i) in the ANSWER section was indeed created by the zone authority Z (as indicated in the AUTHORITY section), TCB-DNS assumes

1. the integrity of TMs A and P: more specifically, i) secrets assigned to TMs are not privy to other entities, and ii) the atomic relay function cannot be modified;

[3]Just as there is nothing that stops an authority of example.com from signing an RRSet for www.yahoo.com in DNSSEC, in TCB-DNS a zone authority can authenticate any value. However, resolvers will not accept RRSet as valid as $Z_{name} = h($example.com$)$ is not a parent of www.yahoo.com.

المنارة للاستشارات
www.manaraa.com

2. the keys of the zone authority Z (possibly protected by a TM Z) are not privy to anyone else; and

3. the hash function $h()$ is pre-image resistant.

With these assumptions, it is easy to see that:

1. as the hash function $h()$ is pre-image resistant, the value M_{PC,i_2} was computed by an entity with access to the secret K_{PC} (thus the verifier can conclude that the value M_{PC,i_2} was computed by TM P);

2. the TM P can compute M_{PC,i_2} *only* if it was provided values $v_{i_1} = h(Z \parallel v_i)$ and M_{AP,i_1}, satisfying $M_{AP,i_1} = h(v_{i_1} \parallel K_{AP})$;

3. only TM A could have computed the value M_{AP,i_1} provided to P;

4. TM A can compute M_{AP,i_1} only if it was provided values $\{v_i, M_{ZA,v_i}\}$ satisfying $M_{ZA,v_i} = h(v_i \parallel K_{ZA})$.

5. as only Z has access to secret K_{ZA}, the value v_i was created by Z.

Note that to conclude that "value v_i was indeed created by Z," it is *not* necessary that the parent zone W supports TCB-DNS. However, it is indeed desirable that all zones adopt TCB-DNS. If the parent zone also supports TCB-DNS, then the client can also verify the integrity of the NS RRSet for zone Z, and thereby confirm that A is indeed a TM associated with an ANS for the delegated zone Z.

4.5 Practical Considerations

TCB-DNS can be implemented with minimal modifications to current DNS servers. The specific modifications required to support TCB-DNS are as follows:

1. Every RRSet will indicate an absolute time of expiry (say, 32-bit UNIX time) specified by the zone authority; this value is unrelated to the TTL value[4] specified in each RR.

[4]The TTL value specifies how long an RR can be cached by resolvers.

www.manaraa.com

2. Each NS record will indicate the TCB-DNS identity of the ANS TM (this is similar to the requirement in DNSCurve where the elliptic-curve public key of the ANS is indicated in the NS record).

3. DNS queries will indicate an additional field - the TCB-DNS identity of the querier.

If an NS record for a zone W provided by a parent zone does not indicate the identity of a TM, the implication is that the indicated ANS for the zone W does not support TCB-DNS. It is also possible for a zone to employ as its ANSs, some TCB-DNS aware servers and some plain DNS servers. The NS records corresponding to TCB-DNS compliant ANSs will indicate the TCB-DNS identity of the ANS. NS records corresponding to plain DNS servers will not. Thus, a PNS which supports TCB-DNS may prefer to query a TCB-DNS compliant ANS for the zone W. Similarly, a plain DNS PNS may choose to direct its query to a plain DNS ANS for zone Z.

If a TCB-DNS server receives a query which does not indicate the TCB-DNS identity of the querier, the querier is assumed to be unaware of TCB-DNS. In this case a plain DNS record is sent as a response. If a TCB-DNS unaware server is queried by a TCB-DNS compliant resolver the DNS server will simply ignore the additional field.

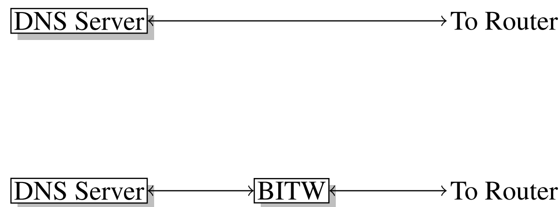

Figure 4.2

Original Configuration

المنارة للاستشارات
www.manaraa.com

TCB-DNS can easily support bump-in-the-wire implementations. Converting a plain DNS server to TCB-DNS server can be as simple as adding an additional BITW unit equipped with a DNS-TM. Only the BITW unit will need to have access to the TCB-DNS supplemental master file. The BITW unit will

1. verify TCB-DNS authentication appended to incoming DNS packets, strip authentication, and relay plain DNS packets to the DNS server; and
2. append TCB-DNS authentication to outgoing DNS packets.

4.5.1 Ideal TMs

Deployment of TCB-DNS requires an infrastructure in place for some regulatory authority (for example, ICANN or IANA) to oversee the production and verification of trustworthy DNS TMs. Mandating rigid and simple functionality can go a long way in reducing the cost for deploying such an infrastructure, reducing the cost of the TMs, and rendering them more worthy of trust.

Just as guaranteed TCB functionality can leveraged to provide some assurances regarding the security of the system, for realizing guaranteed TCB functionality two fundamental assurances provided by trustworthy computing modules are leveraged - *read-proofing* and *write-proofing*.

Read-proofing of secrets protected by the TMs is necessary to ensure that the modules cannot be impersonated. Write-proofing of software executed by the TMs is necessary to ensure that the TCB functions (usually dictated by the software) cannot be modified. The two requirements are however not independent. With the ability to modify software at will, an attacker can force the TM to reveal its secrets (for example, by inserting a set of

المنارة للاستشارات
www.manaraa.com

commands to write the secret bytes out to the serial port). On the other hand, secrets that are protected can be used to authenticate software. Without the knowledge of the secret the attacker cannot modify the software (more specifically, such modifications will be detected due to failure of authentication).

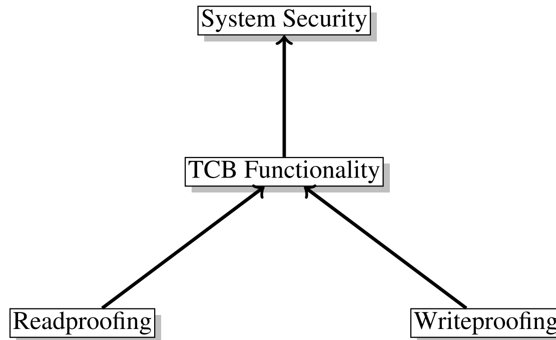

Figure 4.3

Bump-in-the-Wire (BITW) Implementation

In practice read-proofing is easier to realize [50], and is often a stepping stone to the more elusive goal of write-proofing [24]. Attacks aimed at modifying software to reveal secrets can be prevented by ensuring that software does not have access to at least some of the secrets that are protected. Some secrets may be *generated, stored and used* by dedicated hardware [54]. However, authenticating software with the secrets provides a boot-strapping problem [67]. After all, some software should be loaded which includes instructions to load a secret and carry out the steps required to perform authentication.

Realizing TMs that truly *deserve* trust calls for some simple common-sense restrictions to be imposed on the TMs. If the entire functionality of the TM is trivial enough

المنارة للاستشارات
www.manaraa.com

to be *hard-wired* (and thus eliminate the need for mutable code), we can side-step issues associated with guaranteeing the functionality of the TM. If the TM functionality is simple and immutable, it is less expensive to *verify* such functionality as the testing infrastructure can be easily automated. If the TMs do not draw significant electrical power to perform its tasks (and consequently disseminate very little heat) we can then afford to physically shield the TMs extremely well from external intrusions aimed at exposing secrets.

It is for these reasons, that in identifying good TCB functionality for TCB-DNS we enforce these restrictions. The low-complexity, low-power, hard-wired TMs employed by TCB-DNS will merely require i) protected registers for storing a few secrets, ii) a hash function (for example SHA-1), and iii) hard-wired logic which drives a fixed sequence of logical and hash computations.

4.5.2 Leveraging TPMs

While creating a dedicated infrastructure for DNS-TMs is the preferred approach to realize highly trustworthy TMs, a practical alternative to lower the cost of the infrastructural requirements is to take advantage of an existing infrastructure for *trusted platform modules* (TPM).

The trusted computing group (TCG) approach to realize a trusted platform includes a thorough specification of trusted platform modules (TPM) [4], and recommendations on how such modules can be leveraged to provide some assurances regarding the integrity of the software stack running on a platform (a general purpose computer) equipped with a TPM chip. Several manufacturers of TPM chips exist today. Many desktop/laptop com-

www.manaraa.com

puters already posses a TPM chip, or have the capability (a slot in the motherboard) to accept a TPM chip.

The TPM exposes several interfaces which can be used by the platform to submit values for secure storage inside the TPM, and conveying such values to other parties, attested using secrets protected inside the TPM. Specifically, about 120 TPM interfaces have been specified in the current version of the TPM [29]. These interfaces are intended for a wide variety of purposes like i) taking ownership of the TPM; ii) generation/regeneration of keys; iii) submitting "measurements" of loaded software (in the form of hashes of loaded software) and extending such hashes into platform configuration registers (PCR) inside the TPM; iv) attestation of PCR values by the TPM for reporting the state of the platform; v) binding secrets to specific platform states (PCR values), etc.

The TCG model relies on three *roots* of trust: the root of trust for measurement (RTM); root of trust for storage (RTS); and root of trust for reporting (RTR). RTS and RTR are housed inside the TPM chip. The RTM is however constituted by components *outside* the TPM. More specifically, trusting the RTR and RTS amounts to trusting the integrity of the TPM chip. However, trusting the RTM implies trusting numerous components of a general purpose computer like the BIOS, CPU, RAM, CPU-RAM bridge, and possibly even some peripherals which have direct access to the RAM. In addition, to an infrastructure for verifying code is trusted to verify software and disseminate measurements (hashes) of authentic verified software.

Almost every attack on the integrity of a TCG trusted platform [57] - [15] is a result of the fact that the RTM is constituted by components outside the TPM, which obviously

المنارة للاستشارات
www.manaraa.com

do not merit the same extent of trust as components inside the TPM chip. Furthermore, the problem of verifying functionality of software is becoming increasingly intangible due to the ever increasing size of software and frequency of updates. For these reasons, some researchers have focused on strategies for securing applications running on untrustworthy platforms by relying *only* on the TPM [52]. The recent TPM 1.2 version added some additional TPM interfaces which expands the scope of applications that can be secured by relying only on the TPM as the TCB. Some desirable additions to TPM interfaces have been suggested in [66].

The atomic relay interface can be simply be an additional interface offered by the next version of TPMs. More specifically, TPMs will require two additional interfaces to serve as the TCB for a DNS server: i) an interface for securely receiving MLS keys (which can then be stored in some TPM PCRs reserved for this purpose), and ii) the atomic relay interface.

4.6 TCB-DNS vs. DNSSEC

The main reasons for the poor adoption of DNSSEC are i) the significant increase in the size of zone files (over that of plain-old DNS) due to the addition of RRSIG, DS, DNSKEY, and NSEC/NSEC3 records; ii) increased bandwidth overhead for DNSSEC responses; and iii) the susceptibility of DNSSEC to DNS-walk. Due to the substantial overhead, it is especially expensive for large zones (for example, `.com`) to adopt DNSSEC.

DNSSEC and TCB-DNSSEC have many significant similarities:

1) Both do not require DNS servers and their operators to be trusted. Both protocols achieve this requirement by their ability to securely convey a pre-image resistant one-way

المنارة للاستشارات
www.manaraa.com

function of RRSets created by zone authorities to end-points (clients), without the need to trust the intermediary servers.

2) In both protocols lifetimes are imposed on the validity of zone keys. Both specify validity periods for the authentication appended for RRs (which can at most be till the expiry of the keys used for validation).

3) Both use a strategy for ordering names (or a one-way function of names) to provide authenticated denial of enclosed names (or hashes of names).

4) Both protocols do *not* possess a mechanism for revoking authentication. Consequently both protocols are susceptible to replay attacks - under some conditions. If the authentication appended for an RRSet is indicated as valid till some time t, and if for some reason, there arises a requirement to modify the RRSet before time t, then an attacker may be able to replay the old RRSet (with a signature valid till time t) until time t.

The primary differences between DNSSEC and TCB-DNS include

1) the cryptographic mechanisms employed - DNSSEC relies on digital signatures, while TCB-DNS relies on TMs to atomically relay MACs;

2) unlike TCB-DNS, DNSSEC does not provide assurance **A3**. Some of specific differences in the mechanism for authenticated denial, and the rationale for the choices made in TCB-DNS are outlined in Section 4.6.1.

3) DNSSEC demands substantially higher overhead compared to TCB-DNS; Section 4.6.2 provides a comparison of the storage bandwidth overheads of DNSSEC and TCB-DNS.

المنارة للاستشارات
www.manaraa.com

4) DNSSEC is more susceptible to replay attacks compared to TCB-DNS; Section 4.6.3 outlines the reasons for this phenomenon.

Since the discovery of the Kaminsky attack [33] the need to secure DNS has attracted renewed attention. Some modifications have been proposed to DNSSEC to address the main reasons for its poor adoption. However, while such efforts reduce some of the overhead for DNSSEC (and thereby reduce the resistance to adoption of DNSSEC), they are at the expense of watering-down some of the originally intended assurances of DNSSEC.

In Section 4.6.4 and we discuss such a mechanism, TSIG [73], which can reduce overhead for clients, but has the unfortunate side-effect of requiring to trust the PNSs. In Section 4.6.5 we discuss another modification (NSEC3 opt-out) [19] which is intended to facilitate easier adoption of DNSSEC by large zones like `.com`. This feature has an unfortunate side effect of interfering with the ability to provide authenticated denial. More recently, some attacks that exploit the NSEC3 opt-out feature have also been demonstrated [10].

4.6.1 Authenticated Denial

Consider a scenario where the ANS for the zone `example.com` is queried for a non-existent record "`a.b.example.com, A`." A negative response indicates that i) the queried name does not exist; *and* ii) no wild-card name like `*.b.example.com` exists; iii) no delegation exists for a zone `b.example.com`; and iv) no alias (type CNAME) record exists for the name `a.b.example.com`.

In NSEC the enclosures are textual strings indicating names. A single NSEC record [`example.com` [A,MX,NS], NSEC, `cat.example.com`] is adequate for the resolver to

المنارة للاستشارات
www.manaraa.com

verify that all three conditions are true (all three names that have to be denied fall within the single NSEC enclosure). In NSEC3 the enclosures are hashes of names. Each NSEC3 enclosure can only be used to deny a *specific* name (which hashes to a value inside the encloser). Thus, proof of enclosure of three different name-hashes have to be provided to the resolver.

If a record of a type different from NS does exist for `b.example.com` or if a record with name `a.b.example.com` does exist (but not the solicited type A), then the NSEC3 record has to indicate the list of types that *do* exist[5].

Though intended as an improvement over NSEC, in some ways NSEC3 is actually inferior to NSEC. In response to a query for a non existent record, NSEC revealed two unsolicited names; NSEC3 typically reveals six hashes corresponding to six unsolicited name-hashes (which are subject to brute-force attacks). Furthermore, three RRSIG(NSEC3) signatures have to be verified (instead of one RRSIG(NSEC)).

The mechanism in TCB-DNS for authenticated denial is closer to NSEC3 than NSEC. The difference is that in TCB-DNS the name and type are hashed together (in NSEC3 only the name is hashed). As with DNSSEC-NSEC3, multiple name-and-type hashes will have to be denied by the ANS by using different enclosures. At first sight, it may seem that an NSEC-like approach may be preferable for TCB-DNS. After all, if only the TM is privy to the enclosures - viz., textual strings (names) in NSEC and hashes (of names) in NSEC3, there is no need for hashing names. However, checking NSEC enclosures will require TMs to compare variable length text-strings, possibly of different formats (for example, ASCII,

[5]Thus, there are two ways in which NSEC3 fails to realize assurance **A3**: i) by being susceptible to simple dictionary attacks; and ii) by disclosing unsolicited types for a name.

للاستشارات www.manaraa.com

Unicode), which can substantially increase the complexity of TMs. With the NSEC3-like approach only fixed-length hashes need to be compared. Thus,

1. in DNSSEC with NSEC3 the purpose of hashing is to "hide" names (albeit ineffectively);
2. in TCB-DNS the purpose of hashing the names is *not* to hide the names - it is to lower the TM complexity.

In TCB-DNS, the reason for hashing name-and-type together is to ensure that (unlike NSEC3) names do not have to be disclosed if queried for a non-existent type. In TCB-DNS the number of encloser pairs equals the number of unique name-and-type values (which is the same as the number of RRSets). In NSEC3 the number of hashes correspond to the number of unique names.

The disadvantage of TCB-DNS is a small increase the number of encloser pairs. However, this is not a problem in practice. The increase in the number of encloser pairs would only be an issue for zones which have a very large number of names, *and* many types corresponding to each name. However, such large zones (like gTLD `.com`) which have large number of names, have only a single type (type NS) corresponding to most names!

Another difference between NSEC3 and TCB-DNS is the mechanism used for hashing names. In TCB-DNS the hash is computed as a function of name, type and a value τ. The value τ is the time of expiry of the authentication. In DNSSEC-NSEC3 the time of expiry is indicated in the RRSIG record; the name-hash is computed after including a salt to the name. Furthermore repeated hashing is employed to derive the name-hash. The reason for using the salt is to prevent *precomputed* dictionary attacks. The purpose of repeated hashing is to increase the computational complexity for dictionary attacks. As dictionary

المنارة للاستشارات
www.manaraa.com

attacks are not possible in TCB-DNS (as the enclosers are never sent), TCB-DNS does not need to deliberately increase the computational overhead for generating name-hashes.

4.6.2 Overhead

Table 4.1

Comparison of TCB-DNS and DNSSEC

	Overhead in Bytes			
	Bandwidth	Cache per RRSet	Cache per Name	Assurances
DNSSEC	2000	200	300	**A1**, **A2**
TCB-DNS	100	24 + 60		**A1**, **A2**, **A3**

Table 4.1 provides a quick comparison of TCB-DNS and DNSSEC. The large size of DNSSEC records is due to the fact that public keys and signatures are large (1000 to 2000 bits). This increases the cache memory requirements for name servers. The longer RR sizes, and that multiple RRSIGs, DS and DNSKEY records need to be fetched and verified, results in substantial bandwidth overhead for responses.

For a typical query, the DNSSEC specific bits that accompany the response (over and above the plain DNS records) can easily be of the order of 2000 bytes. As described in Section 2.1.4, the additional DNSSEC specific records required to verify a RRSet include (typically) one RRSIG(RR), 3 DNSKEY records, 3 DS records, and 3 RRSIG(DS) records. Additionally, verification of an RRset requires the computational overhead for verification of 4 signatures.

www.manaraa.com

For TCB-DNS the additional TCB-DNS specific bits that accompany the response (to a typical query) include i) the identity of the ANS and ii) one MAC in the `ANSWER` section, and two identities (TCB-DNS identity of the parent zone, and identity of an ANS TM of the parent zone) and one MAC in the `AUTHORITY` section. If the identities of TMs are 10 bytes long and identities of zones are 20 bytes long, and all MACs are 20 bytes long, the additional bandwidth overhead is of the order of 70 bytes. If we consider the additional values inserted in NS records the overhead may be close to 100 bytes (compared to 2000 bytes for DNSSEC).

For DNSSEC the increase in cache memory size for any RRSet is due to the addition of one RRSIG for every RRSet (about 200 bytes for every RRSet if 1600-bit RSA modulus is used). Additionally (for authenticated denial), corresponding to every unique name in the master file, one NSEC/NSEC3 record along with an RRSIG(NSEC/NSEC3) record are required: amounting to an overhead of roughly 300 bytes for every unique name in the master file.

In TCB-DNS, corresponding to every RRSet two additional values are required for regular responses - an index (of the RRSet within the master file) and a MAC. For authenticated denial, corresponding to every unique name and type (the total number of which is the same as the number of RRSets) three values are required: two hashes (enclosers), and a MAC for the enclosure. Assuming 20 byte hashes and MACs, the overhead is about 60 bytes for every unique name and type.

www.manaraa.com

4.6.3 Replay Attacks

The fact that anybody can obtain verify a digital signature is a powerful feature of digital signatures. This power can also be abused more easily when no mechanism exists for revocation. A signed packet with prematurely invalidated contents can be more easily abused, compared to a packet authenticated using a MAC.

A DNS RRSet signed by the zone authority can be sent by anyone, from any place, to any place. However, in TCB-DNS, a MAC appended by a zone authority is intended only for the TM of a specific ANS. This implies that only the entity with control of the specific ANS (who has access to the TM) can replay such packets. This substantially reduces the *scope* of possible replay attacks.

For both protocols, reducing the scope of replay attacks requires choice of short life-times for signatures (MACs for TCB-DNS). Unfortunately shorter life-times imply more frequent re-computation of authentication. Due to the substantially lower computational overhead required for TCB-DNS we can can actually afford to recompute MACs more frequently.

4.6.4 DNSSEC with TSIG

As originally intended, DNSSEC provides the end-points with the ability to verify the integrity of RRs. For most clients this is a substantial computational burden. This is especially true for an ever increasing number of battery operated mobile devices. Furthermore, to receive the large number of DNSSEC specific records from the PNS the clients may have to employ more expensive TCP instead of UDP as the transport layer. Note that for

المنارة للاستشارات
www.manaraa.com

PNSs this is less of an issue as it receives the multiple records required for verification as multiple packets from different ANSs.

It is for this reason that in most standard installations of DNSSEC the verification of RRs is performed only by the PNS. Stub-resolvers are expected to establish a secure channel with PNSs using some light-weight mechanism like TSIG [73], and obtain verified RRSets over the secure channel. TSIG is a protocol which leverages shared symmetric keys (established by other means - outside the scope of TSIG) for establishing secure channels. In DNS, TSIG is used by zone authorities to securely send master files to ANSs. This same strategy can also be used for establishing a secure channel between clients and PNSs.

The implication of using such an approach to lower overhead for clients is that *DNSSEC can no longer claim that end-points do not have to trust the middle-men.* With this approach, clients are required to trust the PNSs (and consequently their operators).

More specifically,

1. If a DNSSEC enabled PNS X is compromised, or if the TSIG secret of X is privy to an attacker, then X (or the attacker) can disseminate fake RRSets for *any* zone; such RRSets will be blindly accepted by *all* stub-resolvers which query PNS X.

2. Similarly, if a DNSCurve enabled PNS X is compromised, or if the DNSCurve secret of X is privy to an attacker, then X (or the attacker) can disseminate fake RRSets to all stub-resolvers that query PNS X.

3. On the other hand, in the case of TCB-DNS, if a PNS X (with DNS-TM P) is compromised, the attacker cannot disseminate fake RRSets. It is only if the secrets of the TM P become privy to the attacker (*and* if the TM P has not been revoked) can the attacker disseminate fake RRSets to the stub-resolvers that query PNS X.

المنارة للاستشارات
www.manaraa.com

4.6.5 NSEC3 Opt-out

For consummate realization of DNSSEC assurances even top level domains should adopt DNSSEC. While authenticated denial is an especially important feature for gTLDs, the overhead for this purpose can be substantial for large zones, and especially for zones where new names are frequently added. More specifically, zones with frequent addition / deletion of names become more susceptible to replay attacks.

Consider a scenario where an RRSet corresponding to a new name (or name-hash) x needs to be added. Before the name is added, a signed encloser (x_l, x_h) will exist for x. However, after inserting x the encloser (x_l, x_h) needs to be revoked. Two new enclosers should be added instead - (x_l, x) and (x, x_h). Similarly, consider a scenario when an existing name y needs to be removed, and two signed enclosers (y_l, y) and (y, y_h) currently exist. In this case, both enclosers (y_l, y) and (y, y_h) need to be revoked and replaced with a new encloser (y_l, y_h).

Due to the fact that it is not possible to foresee which of the currently valid records will need to be revoked due to the addition of an (as yet unknown) name in the future, it is necessary to choose small enough life-times for all NSEC/NSEC3 enclosers. Obviously, for gTLDs like `.com` with several tens of millions of names, this is far from practical.

In TCB-DNS, due to the low overhead for computing MACs even gTLDs can afford to recompute enclosers more frequently. However, frequent re-authentication of NSEC3 records in DNSSEC is expensive for two reasons. The obvious reason is that the computational overhead for digital signatures is high. The other reason is that NSEC3 *deliberately* increases the complexity of hashing names to render dictionary attacks more time-

www.manaraa.com

consuming. Due to the substantial overhead involved in re-generation of signed NSEC3 records, DNSSEC is forced to employ larger life-times for NSEC3 signatures and consequently become more susceptible to replay attacks.

Recently, NSEC3 with an opt-out specification [19] has been proposed to make it more practical for gTLDs to adopt DNSSEC. Using opt-out NSEC3 can reduce the instances leading to revocation of RRSIG(NSEC3) RRs, thereby permitting longer lifetimes for NSEC3 RRSIGs. An NSEC3 record indicating an encloser (x_l, x_h) with an unset opt-out bit is proof that no enclosing records exist. However, if the opt-out bit is set, the implication is that zero or more unsigned delegations *may* exist - thereby diluting assurance **A2**. Furthermore, some serious security exploits resulting from using the opt-out approach have been identified recently [10].

Adoption of DNSSEC has been marred by the substantial overhead required, and the issue of DNS-walk. The large increase in the zone file size is especially severe for gTLD DNS servers and DNS servers employed by specialized DNS service providers who run DNS services for a large number of zones. This is further exacerbated by supercilious queries from "DNS-walkers." While the need to address DNS-walk is especially crucial for large DNS operators, ironically, using NSEC/NSEC3 feature of DNSSEC for this purpose is risky for such DNS servers due to the possibility of DNS-walk. Recent attempts to reduce resistance to adoption of DNSSEC have unfortunately come at the expense of security.

The primary insight for TCB-DNS approach stems from the fact that cryptographic techniques for independently securing each link in a query response process (like the approaches in symmetric key DNSSEC and DNSCurve) demand substantially lower over-

المنارة للاستشارات
www.manaraa.com

head. As the light-weight link-security approaches require the intermediaries (DNS servers) to be trusted, a natural question then is “what is the minimal TCB for a DNS server?” As long as this TCB is trustworthy we will not be required to trust other components of the intermediary servers.

A TCB which simply relays hashes can provide assurances **A1** and **A2**. By adding some intelligence to the simple relay function (to verify that “one input lies between two other inputs”) we can realize assurance **A3**, and thus eliminate the problem of DNS-walk. Due to the negligible overhead (a few tens of bytes of bandwidth overhead, and computation overhead amounting to a few hashes) even gTLDs can easily switch to TCB-DNS.

To summarize, the main advantages of TCB-DNS over DNSSEC are

1. TCB-DNS demands substantially lower overhead;
2. TCB-DNS eliminates the issue of DNS-walk;
3. TCB-DNS is less susceptible to replay attacks;
4. due to the low overhead for verification, TCB-DNS will not require clients to trust PNSs (as in DNSSEC with TSIG);
5. due to the low overhead for re-authentication of enclosers, TCB-DNS does not need to employ potentially dangerous practices like “NSEC3 opt-out.”

Deployment of TCB-DNS ideally requires a dedicated infrastructure in place for some regulatory authority to oversee the production and verification of trustworthy DNS TMs. Alternately, two additional interfaces (one for for accepting MLS keys, and the second for performing the atomic relay) can be added to the next version of the trusted computing group (TCG) specification for trusted platform modules (TPM).

المنارة للاستشارات
www.manaraa.com

CHAPTER 5

MINIMAL TCB FOR DATA DISSEMINATION SYSTEM

A common characteristic of any data dissemination system (DDS) is that the providers and consumers of data may not be able to (or desire to) interact directly with each other, and consequently, rely on middle-men.

A mechanism for securing the end-to-end link between the client and a server platform can eliminate the need to trust numerous other infrastructural components in the path between the client and the server. However, users are still required to trust the server. Trust in a server implies confidence in the integrity of the server software, the platform on which the server software is executed, personnel who may have control over the platform, and that secrets employed by the platform (for authenticating served data to consumers and acknowledging receipt of data to providers) are well protected.

This chapter proposes a comprehensive security solution for a generic DDS consisting of a dynamic set of providers disseminating dynamic data through an untrusted middle-man, to a dynamic set of consumers. The paper identifies a simple TCB for a generic DDS that can be leveraged to realize all desired assurances regarding the operation of the DDS. Specifically, the TCB is a set of simple functions $F_1() \cdots F_n()$ executed inside the boundary of a trusted module $\mathbf{T}$. Our desire to simplify the TCB translates to a desire

www.manaraa.com

to limit the computational and storage burden required for module $\mathbf{T}$ to execute functions $F_1() \cdots F_n()$.

Central to the proposed solution is the capability of even severely resource limited modules to maintain an index ordered merkle tree (IOMT). An IOMT is a simple extension of the well known merkle hash tree [42] to provide the ability to verify non-existence. Our specific contribution is a precise characterization of the TCB functionality as four functions: i) $F_{uk}()$, used to issue symmetric secrets to the users of the DDS (viz., providers and consumers of data); ii) $F_{idl}()$, used to insert or delete leaves of an IOMT; iii) $F_{upd}()$, used to update a record from a provider; and iv) $F_{qry}()$ to respond to a query by any user.

5.1 Organization

In Section 5.2 we discuss the model for a generic DDS and enumerate the desired assurances $\mathcal{A}$. In Section 5.2.1 we discuss some of the current efforts to provide some assurances regarding the operation of an untrusted middle-man, which rely on the use of authenticated data structures (ADS), and some of the limitations of such approaches. In Section 5.2.2 we provide a broad overview of the proposed approach where a trusted module $\mathbf{T}$ serves as the TCB for the DDS. In Section 2.3.2 we provide an algorithmic overview of the index ordered merkle tree. Section 5.2.3 provides an algorithmic description of the TCB functions executed by the trusted module $\mathbf{T}$.

5.2 A Generic Data Dissemination System

We model a data dissemination system as being composed of a dynamic set of users $\mathcal{U}$ and a look-up server L. Users could be providers or consumers, or both. Any user may

المنارة للاستشارات
www.manaraa.com

provide a succinct record $\mathbf{R} = [o, l, v, \tau]$ regarding an object to a look-up server. The object is uniquely identified by the owner o and a label l assigned by the owner, and is associated with a value v and a duration of validity τ of the record.

The owner o of a record is permitted to remove or modify the record at any time. Specifically, the owner may modify the record even while the current record has not expired. Any record may be queried by any user by specifying the owner and label. The querier expects the response to contain the most recent version of the queried record. More specifically, the querier expects the same response as she would *if* she had directly queried the owner o.

If the object itself is succinct (for example, an email address, a public key, etc.), the object may also be stored by the look-up server, and may be returned along with the response (in this case the value v may be a cryptographic hash of the email address or public key). More generally, the value v provides some information regarding the object, and could take several forms like the location of the object (a URL U), the cryptographic hash of the object (which can permit the client to verify the integrity of the object after it is obtained from location U), information necessary to establish a private channel with a data server at location U, etc.

The look-up servers simply do not care about the specific nature of the object or the purpose of a specific query. One query may be for the location $v = U$ of a file $l = F$ provided by a provider $o = P$. To establish a secret with the server U another query may be made for a record provided by $o = U$. The object F fetched from U may include components authenticated by another entity X. To verify the integrity of such components

المنارة للاستشارات
www.manaraa.com

the client may desire the public key of X. This data may be disseminated through the same DDS by another entity - for example, a certificate authority $o = C$. If the CA C desires to revoke the public key of entity X, the CA may simply instruct the look-up server to remove the record indexed by $o = C$ and $l = X$.

Some of the basic desired assurances regarding the operation of any DDS are as follows:

1) Records can be modified only by owners; specifically, modifications to records by any entity other than the owner will be detectable by consumers;

2) Servers should only respond with records corresponding to the most recent update, and should not be able to replay prematurely invalidated records;

3) Servers will not be able to hide the presence of records that exist;

4) Servers will only provide a record if explicitly queried; more specifically, servers should *not* need to reveal the existence of records that were not explicitly queried.

5.2.1 Related Work

Several researchers have addressed issues in reliably querying an untrusted server using *authenticated data structures* (ADS) [8, 16, 20, 21, 25, 26, 39]. In such scenarios, clients who query a server trust only the originator/provider of the data (and not the server). Specifically, even while the originator of the data is not online, from a security perspective, ADS based schemes strive to provide the same assurances possible in scenarios where the queriers directly query the originator.

المنارة للاستشارات
www.manaraa.com

Broadly, an ADS can be defined by a construction algorithm $f_c()$ and a verification algorithm $f_v()$. The provider A of a set of records $\mathcal{D}_A$ computes a static summary $d = f_c(\mathcal{D}_A)$. The records $\mathcal{D}_A$ are hosted by an untrusted repository/server. Along with a response R to a query by a client, the server is expected to send a *verification object* (VO) ν satisfying $d = f_v(\nu, R)$, and the signature of the provider for the summary d. The client is now convinced that the response R would be the same *if* the client had directly queried A.

In ADS based approaches the owner A of the set of records $\mathcal{D}_A$ constructs a *hash tree* like data-structure with the records as leaves and the succinct summary d as the root of the tree. From this perspective, the ADS construction algorithm $f_c()$ can be seen as the algorithm to insert a leaf into the tree, and the VO can be seen as a set of hashes required to verify the integrity of any leaf against the root d using the verification algorithm $f_v()$. Most commonly used hash tree data structures for ADS applications include skip-lists, red-black trees and B-trees, all of which provide the capability to *order* records in a set (based on some index).

The purpose of ordering records is to permit succinct responses to i) queries for non existing indexes, ii) maximum/minimum value queries and iii) range queries. For example, if a querier seeks a record for an index X that does not exist, a two adjacent records in the tree can be sent (along with VOs to verify the two records against the root d signed by the originator) - one for an index x and one for the next index y such that $x < X < y$. As the tree is constructed by the originator, to the extent the querier trusts the originator of the data, the querier is assured that the queried index does *not* exist. When queried for all records in a range X to Y the server provides all records that fall in the range (each

المنارة للاستشارات
www.manaraa.com

accompanied by an independent VO), and in addition, to prove completeness (to assure the querier that the server has *not* omitted any existing record) the server provides two additional records - a record for index $x < X$ indicating X as the next record and a record indicating that $y > Y$ is the index that follows Y.

5.2.1.1 Limitations of Existing Approaches

Some limitations of the ADS based approach render it unsuitable for several practical services with any of the following characteristics:

a) *Multiple Independent Providers:* In scenarios with multiple independent providers a record for an index X may be provided by and entity A and the record with the next higher index Y may be provided by an independent entity B. Clearly, neither A nor B can construct the hash tree.

b) *Truly dynamic data:* In most ADS based applications it is assumed that whenever any record is modified, the new root is signed by the originator and issued to the server. In scenarios where future modifications are unforeseen, the originator needs to sign the roots with short enough validity durations to ensure that the old root cannot be replayed by the server. Thus, the originator needs to send fresh signatures for the current root periodically, even if no updates were performed. In scenarios where the originator desires to be involved *only* for purposes of providing updates, existing ADS schemes are unsuitable.

c) *Revealing unsolicited information:* In many application scenarios it is desirable that servers should not be required to provide unsolicited information to queriers. As an example, in the case of the domain name system (DNS) [43], [9], to prove that no record with

المنارة للاستشارات
www.manaraa.com

the queried DNS name exists, a DNS server is required to provide two name names that cover the queried name, thus revealing unsolicited information. This is the cause of the well known “DNS walk” or “zone enumeration” issue associated DNSSEC [76], [36].

d) *Low bandwidth overhead is desired:* In some scenarios, the overhead for the verification object (usually a sequence of hashes) required by clients for verifying any record may be unacceptable.

e) *Entrusting secrets to servers:* In some application scenarios the data to be conveyed to the client may be a secret. While ADSs can ensure integrity of data stored at untrusted servers, they do not address issues related to privacy of the data. Thus, in scenarios where middle-men need to be entrusted with secrets, conventional ADS schemes cannot be used.

5.2.2 Salient Features of the Proposed Approach

All the above inadequacies can be overcome if ADSs are *constructed and verified* by a *trusted third party* (TTP). Specifically, as typical ADS construction and verification algorithms involve only simple sequences of cryptographic hashing and logical operations, the TTP can be a low complexity trustworthy module $\mathbf{T}$. As the intent of the TTP is to ensure that middle-man cannot violate rules, module $\mathbf{T}$ functionality for constructing/verifying ADSs is the TCB for a middle-man.

In the proposed security model, the look-up servers are untrusted. However, a look up server has access to a trusted module $\mathbf{T}$ which performs some trivial functions $f_1() \cdots f_n()$ that constitute the TCB for the system. While a look-up server may be required to maintain and serve a dynamic number (say n) of records where n could be millions or even billions,

www.manaraa.com

the module $\mathbf{T}$ is assumed to possess only modest computational ability and small constant $\mathbb{O}(1)$ storage capability.

Records submitted by providers are stored as leaves of an index ordered merkle tree (IOMT), uniquely indexed as a function of the owner o and a label l (more specifically, $h(o, l)$, where $h()$ is a standard cryptographic hash function like SHA-1). The module stores only the root of the IOMT (a single hash). By performing simple sequences of hash operations, the module can verify the integrity of any record against the root of the tree.

In the proposed approach, a query from any user (a consumer) specifies the owner o and label l. The querier expects a response that is cryptographically authenticated by module $\mathbf{T}$. Similarly, requests for updates by users (providers) are authenticated by providers for verification by the module. On submitting an update request the providers expected an authenticated acknowledgement from the module. On receipt of a response authenticated by the module, to the extent that the users trust the module, they are assured that all four assurances are met.

5.2.2.1 Query-Response Authentication

Modules have to verify authentication appended by providers for every update, and sign every response for verification by the querier. Obviously, for servers that may have to handle large volumes of queries and updates, the cost of authentication will be a significant bottle-neck - especially if asymmetric primitives are employed.

To amortize the overhead for authentication, asymmetric primitives are used only for setting up symmetric keys between the module and the users (providers and consumers).

المنارة للاستشارات
www.manaraa.com

Message authentication codes (MAC) based on such symmetric keys are then used for authentication of exchanges between users and the module.

5.2.2.2 Opportunistic Shared Secrets Between Users

Often, a query is to locate some service/entity S with whom the querier Q expects to interacts soon after the query, and would thus desire a mechanism to secure the interaction. It is beneficial to use the trusted module to also opportunistically provide additional information required for the Q and S and to establish a secret K_{QS}. To cater for resource limited portable devices, it is desirable that mechanisms for establishing the shared secret be limited to symmetric primitives.

In the proposed approach, the shared secret established between the module and the users are also leveraged to opportunistically establish shared secrets *between* users. Any user (say provider S) can submit a record to the look-up server to request the trusted module to serve as a mediator. In response to a query from some user Q for such a record from S, the querier will receive a non secret value p_{QS} that can be used to compute a symmetric secret K_{QS}. Both Q and S will need to perform only a single hash operation to compute the common secret K_{QS}.

5.2.2.3 Potential Applications

While almost any Internet/Intranet based service can be seen as falling under the category of a DDS, some specific examples are as follows:

1) Dissemination of dynamic DNS records by zone authorities. Response to negative queries will *not* reveal records that exist, and thus overcomes the DNS-walk issue that

المنارة للاستشارات
www.manaraa.com

plagues the current DNSSEC [9] approach to secure DNS. In addition, unlike DNSSEC, the proposed approach can also be used to provide assurances for *dynamic* DNS (where DNS records may expire prematurely). Furthermore, server platforms may also convey records to facilitate any client (who performs a DNS look-up for the server) to opportunistically establish a shared secret with the server, which could be used as an IPSec security association.

2) Mobile servers with highly dynamic addresses could disseminate their reach-ability information (and enable clients to establish a secure channel with such servers);

3) Dissemination of dynamic revocation lists by certificate authorities, without the bandwidth overhead associated with certificate revocation lists (CRL).

4) Publication of dynamic quotes by any "exchange";

5) Dissemination of encryption secrets for Email messages; a user desiring to send an encrypted Email to some address will merely make a query to a look-up server to obtain a symmetric secret for encrypting the message.

5.2.3 TCB Functions

The module $\mathbf{T}$ exposes four functions $F_{uk}()$, $F_{idl}()$, $F_{upd}()$, and F_{qry} to the look-up server housing the module. In general inputs to the functions include values sent by a user to the look up server, and values stored by the server that are demonstrably consistent with the IOMT root stored inside the module. The outputs of the module include values like current time (according to the module), a message authentication code (MAC) for verification by a user encrypted secrets that can be decrypted by the user.

المنارة للاستشارات
www.manaraa.com

5.2.3.1 Conveying User Secret

Interface $F_{uk}()$ is employed to securely convey a secret K_o to a user with public key U_o (who is assigned an identity $o = h(U_o)$). The secret K_o may be used by the user to send authenticated update requests regarding objects owned by the user or send authenticated queries for objects provided by any user.

User :	Generate Key pair (R_o, U_o)
User :	Choose random challenge c, Compute $C = F_{enc}(U, c)$
User → LU :	$U_o, C, (\tau),$
LU → $\mathbf{T}$:	$F_{uk}(U_o, C, \tau)$
$\mathbf{T}$:	$o := h(U_o); t_{ek} = t + \tau; K_o = h(S, o, t_{ek});$
$\mathbf{T}$:	$K_o^c := f_{dec}(C) \oplus K_o;$
$\mathbf{T}$ → LU :	$t, C' = f_{enc}(U_o, K_o^c), \mu = h(t, C', \tau, K_o)$
LU → User :	$t, C', \mu, (\tau)$
User :	$K_o = f_{dec}(R_o, C') \oplus c;$ Verify $\mu = h(t, C', \tau, K_o)$
User :	$t_{ek} = t + \tau$

Any user can generate a key pair (R_o, U_o) (using the specific asymmetric scheme supported by the module) and send a challenge to the module $\mathbf{T}$, encrypted using the module's public key, along with the user's public key U_o. The user or the server can specify the validity duration τ for the MAC key K_o issued to the user. The key K_o can be used by the user for computing MACs to authenticate update requests or queries sent by user till time $t_{ek} = t + \tau$ (time according to the module).

المنارة للاستشارات
www.manaraa.com

5.2.3.2 Inserting and Deleting IOMT leaves

The module maintains the root of an IOMT with any number of leaves of the form (a, v_a, a'). It is the responsibility of the server to store the leaves and intermediate hashes. The module considers a leaf as valid only if provided a set of hashes $\mathbf{v}_a$ satisfying $f(l_a, \mathbf{v}_a) = \xi$ where the leaf hash is computed as $l_a = H_L(a, v_a, a')$.

If $v_a \neq 0$, the leaf is interpreted as a record provided by owner o with label l such that $a = h(o, l)$, and $v_a = h(v, t_e, t_o)$, where v is a value associated with the record, t_e is the expiry time of the record, and $t_o \in \{0, t_{ek}\}$ (t_o can be zero, or the expiry time t_{ek} of the key K_o of the owner o). If $t_o \neq 0$ the module interprets this as a request to enable a private channel between any querier of the record and the owner o.

If $v_a = 0$ (if the middle value in the leaf is zero) the leaf is a "place-holder" and implies that no information is available regarding index a. The server can request insertion of any place holder (if no leaf or place holder currently exists for the index to be inserted) or delete any place holder. TMM function $F_{idl}()$ verifies the simple conditions that need to be satisfied for a place-holder to be deleted, and can be used to delete or insert a place-holder can be described algorithmically as shown below.

المنارة للاستشارات
www.manaraa.com

LU $\rightarrow$ $\mathbf{T}$: $(l, v_l, l'), (r, v_r, r'), i, \mathbf{v}_l, \mathbf{v}_r, \mathbf{v}_p$

$\mathbf{T}$ $\rightarrow$ LU : $\xi = F_{idl}((l, v_l, l'), (r, v_r, r'), i, \mathbf{v}_l, \mathbf{v}_r, \mathbf{v}_p)\{$

IF $(l = 0) \wedge (r = 0)$ RETURN; //At least one leaf should be non zero

IF $(l = 0) \vee (r = 0)$//If one leaf is zero it is the only leaf

$\xi_1 := H_L(i, 0, i); \xi_2 := 0;$ //i is the sole index

ELSE

IF $(l = i)//(l, v_l = 0, l') \rightarrow (0, 0, 0)$

IF $(v_l \neq 0) \vee (r' \neq i)$ RETURN; //Prereqs not satisfied

$l'_l := 0; l'_r = H_L(r, v_r, l');$

ELSE IF $(r = i)//(r, v_r = 0, r') \rightarrow (0, 0, 0)$

IF $(v_r \neq 0) \vee (l' \neq i)$ RETURN; //Prereqs not satisfied

$l'_r = 0; l'_l := H_L(l, v_l, r');$

ELSE RETURN;

$l_l := H_L(l, v_l, l'); l_r := H_L(r, v_r, v');$

$\xi_1 = f(H_V(f(l_l, \mathbf{v}_l), f(l_r, \mathbf{v}_r)), \mathbf{v}_p);$ //Root before deletion

$\xi_2 = f(H_V(f(l'_l, \mathbf{v}_l), f(l'_r, \mathbf{v}_r)), \mathbf{v}_p);;$ //Root after deletion

IF $(\xi = \xi_1)$ $\xi := \xi_2;$ // delete index i

IF $(\xi = \xi_2)$ $\xi := \xi_1;$ //insert index i

RETURN ξ;

}

To delete a place holder the LU server provides two current leaves - a place-holder $(i, 0, i)$ corresponding to the index i to be deleted, and a leaf $(j, v_j, j' = i)$ which points to the place-holder to be deleted. An exception is for deletion of a sole leaf $(i, 0, i' = i))$ in the tree (in which case the root $\xi = H_L(i, 0, i)$ should be set to 0).

المنارة للاستشارات
www.manaraa.com

To compute the root from two leaves three sets of complementary hashes are provided to the module. The module computes the roots ξ_1 and ξ_2 - the roots before and after deletion respectively. If the current root ξ is either ξ_1 or ξ_2 it is reset to ξ_2 or ξ_1 respectively. Specifically, if the current root is ξ_1, and if the leaves provided satisfy the condition for deleting index i, by setting the root to ξ_2 a leaf with index i is deleted. On the other hand, for the same inputs, if the current root is ξ_2 then by setting the root to ξ_1 a leaf corresponding to index i is *inserted*.

5.2.3.3 Updating Records

Typically, to provide a record or update a record the owner needs to send the values corresponding to the new record authenticated using a MAC μ. An exception for updating a stored record is when the stored record has expired, in which case the server can request the module to convert the record to a place holder (which can then be deleted if required using $F_{idl}()$).

A request for update from user o for a record with label l, includes a value v', a period of validity τ, and a flag f, a nonce n, and a MAC computed over values o, l, v, τ, f, n and secret K_o. After completion of the update the user expects an acknowledgement authenticated by the module.

To enable the module to compute $K_o = h(S, o, t_{ek})$ the inputs include the time of expiry t_{ek} of the key K_o. If no leaf exists for index $a = h(o, l)$ a place holder is inserted by the server using $F_{idl}()$. If the current leaf is a place holder (as will be the case when the record is provided for the first time) this fact is indicated to the module by setting $t_e = 0$.

www.manaraa.com

In the updated record the value v is set as requested to v'. The expiry time t_e of the record is set as $t_e = t + \tau$. If the flag f is set in the request the value t_o is set to be the same as the time of expiry t_{ek} of the key K_o of the owner. If $f = 0$ the value t_o is set to 0 instead.

User $\rightarrow$ LU : $o, l, v', n, \tau, f, t_{ek}, \mu = h(v', l, \tau, f, n, K_o)$

LU : If no leaf with index $a = h(o, l)$ insert index a using $F_{idl}()$

LU : If leaf for index a is a place holder, set $t_e = 0$;

LU : If no request from user $\mu = 0; t > t_e$

LU $\rightarrow$ $\mathbf{T}$: $(o, l, v, t_e, t_o, a', \mathbf{v}, \mu, t_{ek}, n, v', \tau, f)$

$\mathbf{T}$ $\rightarrow$ LU : $F_{upd}(o, l, v, t_e, t_o, a', \mathbf{v}, \mu, t_{ek}, n, v', \tau, f)\{$

$a := h(o, l); v_a := (t_e = 0) \, ? \, 0 \; : \; h(v, t_e, t_o);$

IF $(\xi \neq f(H_l(a, v_a, a'), \mathbf{v}))$ RETURN;

IF $(\mu = 0) \wedge (t > t_e)$//Expired record

RETURN $\xi := f(H_l(a, 0, a'), \mathbf{v});$

IF $(t > t_{ek})$ RETURN; //Expired user key

$K_o := h(S, o, t_{ek}); t'_e = t + \tau;$

IF $(\mu \neq h(v', l, \tau, f', n, K_o))$ RETURN;

$t'_o := (f = 1)? \, t_{ek} \; : \; 0;$

$v'_a := h(v', t'_e, t'_o); \xi := f(H_l(a, v'_a, a'), \mathbf{v});$

RETURN $\xi, t, \mu' := h(a, v', t'_e, f, n, K_o);$

$\}$

LU $\rightarrow$ User (only on successful execution of $F_{upd}()$): μ', t

User : $t'_e = t + \tau; a = h(o, l);$ Check $\mu = h(a, v', t'_e, f, n, K_o);$

Only if $F_{upd}()$ executes successfully will the server receive a MAC μ' that can be conveyed to the user.

المنارة للاستشارات
www.manaraa.com

5.2.3.4 Querying Records

A user q may send a query for an object by specifying the owner o and label l and a nonce. No information may exist regarding the queried index a due to one of the following reasons

1. no leaf with index a exists; or
2. the leaf with index a is a mere place holder; or
3. the record has expired;

As any such reason can be verified by the module, the module can send an acknowledgement to the effect that no data is available. On the other hand, if the queried index exists, the module prepares an authenticated response which conveys the value v, and the remaining duration of validity (which is $t_e - t$). In addition, if the value t_o is not zero the module includes an additional value p_{qo} in the response which will enable q to compute a shared secret with the owner o of the queried record.

Specifically, if $t_{ek} > t$ and $t_o > t$ (both are current) the module computes $K_q = h(S, q, t_{ek})$, $K_o = h(S, o, t_o)$, and

$$p_{qo} = h(K_q, o, t_o) \oplus h(K_o, q, t_{ek}). \tag{5.1}$$

Using secret K_q the user q can compute $K_{qo} = h(K_q, o, t_o) \oplus p_{qo} = h(K_o, q, t_{ek})$ which can be readily computed by o using its secret K_o. Thus, both q and o can compute a secret by performing a single hash. The query response process can be algorithmically described as follows:

المنارة للاستشارات
www.manaraa.com

User $q \rightarrow$ LU : $a_q, n, \mu = h(a_q, n, K_q)$;

LU : either $a = a_q$ or a covers a_q

LU : If leaf for index a is a place holder, set $t_e = 0$;

LU $\rightarrow$ **T** : $(a, o, l, v, t_e, t_o, a', \mathbf{v}, q, \mu, t_{ek}, n)$

T $\rightarrow$ LU : $F_{qry}(a, o, l, v, t_e, t_o, a', \mathbf{v}, q, \mu, t_{ek}, n)\{$

IF $(t > t_{ek})$RETURN ; //Expired user key

$K_q := h(S, o_q, t_{ek}); a_q := h(o, l); p_{qo} := 0;$

IF $(\mu \neq h(a_q, n, K_q)$ RETURN;

$v_a := (v = 0)\,?\,0\,:\,h(v, t_e, t_o);$

IF $(\xi \neq f(H_l(a, v_a, a'), \mathbf{v}))$ RETURN;

IF $(t_e < t)$ RETURN;

IF $(a = a_q) \wedge (v_a \neq 0) \wedge (t < t_e)$//Unexpired Queried Record

  IF $(t_o > t)$//Compute pairwise public value

    $K_o := h(S, o, t_o); p_{qo} = h(K_o, h(q, t_{ek})) \oplus h(K_q, h(o, t_o));$

  $\mu' = h(a, v, t_e - t, p_{qo}, t_o, n, K_q);$

ELSE IF $(a = a_q) \vee ((a < a_q < a') \vee (a_q < a' < a) \vee (a' < a < a_q))$

  $\mu' = h(a_q, 0, 0, 0, 0, n, c)$; //Expired Record or Record NA

ELSE RETURN; //incorrect proof of non existence by server

RETURN ξ, t, μ', p_{qo};

}

LU $\rightarrow$ User (only on successful execution of $F_{qry}()$): $t, \mu', p_{qo}, v, t_e, t_o$;

User : if $v = 0$ Verify $\mu' = h(a_q, 0, 0, 0, 0, n, K_q)$

User : if $v \neq 0$ Verify $\mu' = h(a_q, v, t_e - t, p_{qo}, t_o, K_q)$

User : if $t_o > 0$ compute $K_{qo} = h(K_q, h(o, t_o)) \oplus p_{qo}$

المنارة للاستشارات
www.manaraa.com

A simple trusted module with fixed functionality defined by functions $F_{uk}()$, $F_{idl}()$, $F_{upd}()$ and $F_{qry}()$ can be utilized to assure the operation of any look-up server, and thereby secure a wide range of applications under the DDS model. Specifically, the server maintains all records and internal nodes of the IOMT, and is forced to ensure that the values stored by the server remains consistent at all times with the root stored inside the module. Any record that cannot be demonstrated to be consistent cannot be updated, or conveyed to users. Specifically, only if the updates are applied in a consistent manner can the server send an authenticated acknowledgement to the user requesting the update; only if a record is consistent with the root can the server send the record to the querier (along with a MAC generated by the module).

In the proposed approach asymmetric cryptographic primitives are used sparingly - only for establishing shared secrets between users and the module. Unlike conventional ADS systems where for verification of any record the client requires a verification object in the form of a set of $\log_2 n$ hashes (where n is the total number of records stored by the server), in the proposed approach the VO is provided by the middle-men to the module, and only a single MAC is sent to the user to attest the accompanying record.

No component of the server, the user in control of the server, or the numerous components necessary for the functioning of any wide area network, need to be relied upon to realize the desired assurances. As long as the cryptographic hash function $h()$ is pre-image resistant, and the functions executed by the module cannot be modified, and the secrets protected by module cannot be exposed, all desired assurances are guaranteed.

المنارة للاستشارات

www.manaraa.com

Our motivation to reduce the complexity of operation performed by the module $\mathbf{T}$ is to improve the trustworthiness of the module. The simpler the functionality of the module, the better is the ability to verify the integrity of such functionality. Furthermore, due to generic nature (fixed functionality irrespective of the type of the look-up service) such modules can be easily mass produced; the process for verifying and certifying fixed functionality modules can also be easily automated.

المنارة للاستشارات
www.manaraa.com

CHAPTER 6

MINIMAL TCB FOR SCADA SYSTEM MONITOR

An essential prerequisite for the ability to monitor a SCADA system is an accurate picture of the current states of all sensors and actuators of the system. Misrepresentations of the state can be perpetrated either by sending misleading information (for example, by impersonating a sensor) or by preventing sensor measurements from reaching the monitor (for example, jamming). We identify a minimal trusted computing base (TCB) for a SCADA monitor, and a strategy to leverage the TCB efficiently to realize the assurance that "any misrepresentation of the SCADA system state will be identified." In the proposed approach the TCB is a severely resource limited trusted module. Periodic reports from sensors/actuators are stored by the untrusted SCADA monitor as leaves of an ordered Merkle hash tree; only the root of the tree is stored inside the module. The untrusted monitor is required to periodically offer proof to the trusted module regarding the integrity and freshness of reports from all sensors.

At any instant of time, the *state* of a SCADA system is defined by the values sensed by various sensors, and the states of different actuators (for example, on/off, open/closed) in the system. An important prerequisite for the ability to monitor a SCADA system is

www.manaraa.com

a mechanism ensure that the monitor is provided a complete and accurate picture of the SCADA system state.

Attacks on SCADA systems [60] aimed at misrepresenting the state of the SCADA system can take two broad forms:

1. impersonation of sensors/actuators to provide incorrect state information, and
2. preventing a sensor/actuator report from reaching the monitor.

The former category of attacks can be addressed by mandating cryptographic authentication of sensor data. The emerging standard for SCADA cryptographic modules [63] can be used for this purpose. The second category of attacks could be carried out by an attacker in a variety of ways like i) jamming the channel, ii) cutting a wire, iii) destroying a sensor, etc.

6.1 Problem Statement

An obvious strategy to detect such attacks would be to mandate periodic reports from every sensor (which is a common requirement in any case in most SCADA systems). In practice, every (authenticated) sensor report sent to the monitor can indicate the time t_e before which the next update from the sensor is due (or t_e is the time of expiry of the current report). For a system with N sensors, at any instant of time the monitor should possess N fresh reports. More specifically, at a time t, no record should indicate an expiry time $t_e < t$ (if any such stale record exists, the monitor will trigger an alarm).

To meet the required goals - ensuring that sensor records cannot be impersonated, or illegally modified, or even hidden[1] from the monitor - we trust the monitor to

[1] The existence of a stale record - or the absence of a fresh record - should not be hidden from the monitor.

المنارة للاستشارات
www.manaraa.com

1. verify the cryptographic integrity of sensor records and store the records,
2. periodically verify the expiry time of all records to ensure that no record is stale, and
3. raise an alarm if a stale record is found.

In practice the monitor is often a general purpose computer. Consequently, it is ill-advised to simply trust the monitor.

6.2 Minimal TCB

In this chapter we investigate a minimal TCB for a SCADA system monitor. In our approach we assume that the monitor is an untrusted computer $\mathbf{U}$. The monitor $\mathbf{U}$ is required to periodically offer proof to a trusted module $\mathbf{T}$ that "no sensor record is stale." The trusted module $\mathbf{T}$ may be housed inside $\mathbf{U}$, or plugged into $\mathbf{U}$, or may merely be accessible by $\mathbf{U}$ over an open network.

The module exposes some interfaces to $\mathbf{U}$ through which $\mathbf{U}$ submits authenticated records from sensors, and few other values as proof that "no record is stale." The functions performed by the module to verify the proof are the TCB functions. As long as i) the functionality of $\mathbf{T}$ cannot be modified, and ii) secrets protected by $\mathbf{T}$ cannot be revealed, this TCB can be leveraged to realize the desired assurance regarding the SCADA system monitor - that the presence of any stale record cannot be hidden from the module $\mathbf{T}$.

Our specific contribution is the enumeration of the TCB as a set of (three) low complexity functions performed inside the module $\mathbf{T}$: i) Update() for updating received sensor records, ii) FProof() to submit proof of freshness of all N records; and iii) Init() for initializing the module.

المنارة للاستشارات
www.manaraa.com

6.2.1 Principle of Operation

Every sensor shares a secret with the module; reports from sensors are authenticated using message authentication codes (MAC), and indicate the time of expiry of the record. When a report from any sensor is submitted to the module $\mathbf{T}$, the module performs a fixed sequence of simple logical and cryptographic hash operations.

The dynamic set of N current records form the leaves of a Merkle tree, and are stored by the untrusted monitor $\mathbf{U}$. Only the root of the tree is stored inside the module $\mathbf{T}$. Employing a Merkle tree permits the module to verify the integrity of any leaf (sensor record) by performing $\log_2 N$ hash operations (and comparing the result with the root stored inside).

In general, different sensors may indicate different validity durations, depending on the dynamics and the criticality of the process sensed. Sensors that are required to send frequent updates can indicate smaller validity times (seconds or even tenths of seconds) while less critical sensors, or sensors sampling slowly varying processes, may indicate larger intervals (several seconds or even minutes). The update interval for a particular sensor may also change dynamically as some processes may need to be sensed more frequently at certain times.

The untrusted monitor $\mathbf{U}$ is required to periodically prove to the module that no record is stale (till some time $t_e > t$, where t is the current time), and thereby convince the module to not trigger the alarm till time t_e. More specifically, the time t_e is the earliest expiry time among all N sensors. Thereafter, before time t_e the monitor should be able to convince the module to not trigger the alarm till some time $t'_e > t_e$, and so on. It is easy to see that

المنارة للاستشارات
www.manaraa.com

time difference τ between submission of two such proofs is at most equal to the validity duration of the sensor with most frequent updates (or least validity duration).

For a system with a large number (say N) of sensors, it is obviously impractical for the resource limited module to verify each of the N records individually in every interval of duration τ to satisfy itself that no record is stale. A significant novelty in the proposed approach lies in a strategy for indexing records added to the Merkle hash tree to ensure that the module requires to verify only one record to conclude that *no* record is stale. This indexing strategy is motivated by NSEC [76] - a strategy used in the domain name system (DNS) security protocol, DNSSEC [9], to provide authenticated denial of existence of DNS records.

In Section 6.4 we outline the proposed strategy for securing a SCADA monitor using a minimal TCB.

6.3 Background

The ability to reliably monitor SCADA systems controlling critical infrastructure is an important requirement for the security of any nation. As general purpose computers cannot be trusted to perform this task it is necessary to identify a minimal amount of trusted hardware to enforce this requirement.

Several approaches mentioned in Section 2.2.5 have been proposed to improve the trustworthiness of software running on untrustworthy general purpose computers.

المنارة للاستشارات
www.manaraa.com

The primary utility of a Merkle tree stems from a single cumulative verification point for all its data records, which makes it easier to store that one value in a secure location (instead of storing all the data records). This useful feature has been taken advantage of in various application scenarios. The secure co-processor AEGIS [62] utilizes this feature to expand trust in the root hash of the tree which is stored inside its secure memory to provide a trusted boundary for the values stored outside. In [52] a Merkle tree is used to leverage one trusted monotonic counter to realize a large number of virtual monotonic counters.

We assume that a resource limited trustworthy module $\mathbf{T}$ is the verifier which stores the root of a Merkle tree. The prover is an untrusted monitor **U** which stores all N leaves and the $2N - 1$ internal nodes. The N leaves are records specifying the states of N sensors - each record indicating the latest sensor measurement and the time of expiry of the record. The untrusted **U** is expected to maintain the current state of all sensors. However, only the trustworthy module $\mathbf{T}$ is trusted to *enforce* this requirement.

6.4 Trustworthy SCADA Monitor

We consider a SCADA system with i) N sensors/actuators (which send authenticated sensor reports at dynamically varying frequencies (in the rest of this chapter we refer to both sensors and actuators as sensors); ii) an untrusted monitor **U**, iii) a trusted module $\mathbf{T}$; and an iv) alarm module $\mathbf{A}$. Each sensor has a unique identity, represented as $S_1, S_2, \ldots S_N$. The monitor is a software running on an untrusted general purpose computer **U**, and has access to a trustworthy module $\mathbf{T}$.

www.manaraa.com

6.4.1 Overview of Proposed Approach

The module $\mathbf{T}$ has the following limited abilities:

1. $\mathbf{T}$ stores i) the dynamic root r of a Merkle tree; and iv) a secret K provided by a trusted authority; it is assumed that no entity can illegally modify the values r, or expose the secret K;
2. $\mathbf{T}$ can perform a fixed sequence of operations involving cryptographic hashing and logical operations; and
3. $\mathbf{T}$ possesses a clock.

It is assumed that the clocks of all sensors (or cryptographic modules associated with sensors) and the alarm module are synchronized "reasonably well" with the clock[2] of the module $\mathbf{T}$. A sensor S_i (or a module associated with the sensor S_i) is assumed to have access to a secret

$$K_i = h(K \parallel S_i). \tag{6.1}$$

(which can be readily computed by $\mathbf{T}$ using its secret K). The alarm module $\mathbf{A}$ possesses a secret $K_A = h(K \parallel A)$ which can also be readily computed by $\mathbf{T}$. If the alarm module is provided values

$$\mathbf{V} = [t_e \parallel \mu] \text{ where } \mu = h(t_e \parallel K_A), \tag{6.2}$$

the alarm will be suppressed till time t_e. To ensure that alarm is not triggered the untrusted monitor $\mathbf{U}$ is required to periodically obtain such values from the module $\mathbf{T}$, and provide them to the alarm unit. In order to obtain the value $\mu = h(t_e \parallel K_A)$ from $\mathbf{T}$, the monitor $\mathbf{U}$ is required to prove to $\mathbf{T}$ that "no sensor record expires before time t_e."

[2]The clock drift δ should be substantially smaller than τ where τ is the smallest duration of validity of sensor records.

المنارة للاستشارات
www.manaraa.com

The sensors send sensed data in the form of authenticated sensor records. A sensor record sent by a sensor S_i is of the form

$$\mathbf{R}_i = [S_i \parallel \xi_i \parallel t_i \parallel \mu_i] \text{ where}$$
$$\mu_i = h(S_i \parallel \xi_i \parallel t_i \parallel K_i), \quad (6.3)$$

where ξ_i is the measured value, t_i is the time of expiry (or time before which the next report from the sensor S_i is due), and μ_i is a MAC for verification by module $\mathbf{T}$.

Sensor records are stored by $\mathbf{U}$. Upon arrival they are submitted to the module $\mathbf{T}$ (along with some other values, as we shall see soon). In response, $\mathbf{T}$ performs a sequence of hash and logical operations and updates its root r. At any time t, $\mathbf{U}$ is required to store N records - one for each sensor. Each record is a leaf of a Merkle tree, and only the root of the tree is stored inside $\mathbf{T}$.

6.4.2 Example of the Utility of the Proposed Approach

Consider a system with (say) $N = 10000$ sensors where different sensors have different periods of updates (and the period of updates themselves are dynamic). Assume that on an average, 100 sensors update values once every second or so, 500 sensors update values once every ten seconds, 2000 sensors update values once every 100 seconds, and the rest (7400) update values once every 400 seconds. Thus (on an average), during every 400 second interval the monitor receives about 75400 reports (or about 190 reports per second). For this example, the minimum duration of validity τ is one second. Thus, at least once every second the monitor $\mathbf{U}$ should prove to the module $\mathbf{T}$ that no sensor record is stale, and obtain a MAC verifiable by $\mathbf{A}$.

المنارة للاستشارات
www.manaraa.com

At a time t, to ensure that no record is stale the module will need to check *every* record - to determine the record with the earliest expiry time t_e, and that $t_e > t$. Obviously, all 10000 values cannot be stored inside storage limited $\mathbf{T}$. As the records are stored outside (as leaves of a merkle hash tree), in order to accept the record as authentic, module $\mathbf{T}$ expects instructions $\mathbf{v}_i$ to map any record (say leaf l_i) to the root r as $f(h(l_i), \mathbf{v}_i)$. Unfortunately, it is far from practical for the module to verify each of the $N = 10000$ records every second. In other words, such an approach does not scale well (for large N).

Central to the proposed approach is a strategy for maintaining an ordered list of sensor records - ordered by expiry time - to ensure that only one record need to be verified by the module $\mathbf{T}$ to determine the earliest expiry time. More specifically, for the example above with $10,000$ sensors the module will *not* need to process $190 + 10000$ sensor records every second (190 record updates and 10000 records to be checked to identify the sensor with the least expiry time). Instead, the module needs to process only 190 updates and 1 (instead of 100000) stored sensor record every second.

6.4.3 Auxiliary Sensor Data

At any time t, the set of N records stored by $\mathbf{U}$ is of the form

$$\begin{array}{cccc} S_1 & \xi_i & t_1 & a_1 \\ S_2 & \xi_2 & t_2 & a_2 \\ & \vdots & & \\ S_N & \xi_N & t_N & a_N \end{array} \tag{6.4}$$

المنارة للاستشارات
www.manaraa.com

where ξ_i is the reported value for sensor S_i, and t_i is the time till which the sensor record is valid. The last field - the "auxiliary" value - a_i in a record for sensor S_i is obtained by sorting the values $t_1, t_2, \ldots, t_N$ of all N records in an ascending order. More specifically, the value that follows t_i is a_i. If t_i happens to be the highest value (last value in the sorted order) then the value that follows is the first value - or a_i is the first (smallest) value in the sorted list.

Shown below is an example depicting t_i and a_i values for a set of $N = 8$ sensor records.

	ξ	t_e	a
S_1	5	1002	1008
S_2	6.78	845	848
S_3	0	850	1002
S_4	5	840	842
S_5	4.44	848	850
S_6	0	1008	835
S_7	0.76	835	840
S_8	0	842	845

(6.5)

The least t_i value (835 for S_7) is accompanied by $a_7 = 840 = t_4$ - the next higher value. Similarly, $a_4 = 842 = t_8$, and so on. Corresponding to the record S_6 with the highest $t_6 = 1008$ is $a_6 = 835 = t_7$, the least t value.

The reason that the sensor data is stored with an auxiliary field is to be able to easily offer proof to the module $\mathbf{T}$ that "no record is stale." Now, with the auxiliary value a_i, $\mathbf{U}$ has to provide only one sensor record - the record which includes a wrapped-around

المنارة للاستشارات
www.manaraa.com

value (t_l, a_l) with $a_l < t_l$ (in the specific example above, the record for S_6 with values $(1008, 835)$). This record indicates that $a_l = 835$ is the earliest time of expiry. Thus, as long as $a_l = 835$ is greater than the current time t, the module is convinced that *all* records are fresh.

That a leaf l_i indicates a value (t_i, a_i) is proof that "no leaf exists in the tree with a expiry time t *covered* by (t_i, a_i). Such an approach is used in NSEC [76] records in DNSSEC (DNS security) [9] for providing authenticated denial of queried records. An NSEC record of the form (`abc.example.com`, `add.example.com`) proves that no record pertaining to a name `ac.example.com` exists.

6.4.3.1 Initializing Sensor Data

A trusted authority constructs initial sensor records of the form $[S_i \parallel \xi_i \parallel d_i \parallel a_i]$ where d_i is the initial expiry time of S_i, and the value a_i is obtained by sorting all d_i values in an ascending order, and choosing the value following d_i from the sorted list as a_i.

Let r_0 be the root of the tree with the initial N records as root. The monitor $\mathbf{U}$ is initialized by providing all leaves to $\mathbf{U}$; the module $\mathbf{T}$ is initialized by providing the root r_0, and setting its clock to 0. Simultaneously, the clocks of all sensors and the alarm unit are set to zero.

The interface Init() to initialize $\mathbf{T}$ can be called only by the trusted authority to provide the value r_0 (the initial root). In response, the module $\mathbf{T}$ i) sets its clock counter to zero; ii) stores r_0 in an internal register. From this point onwards, every received sensor record is submitted by $\mathbf{U}$ to $\mathbf{T}$ to update the root.

المنارة للاستشارات
www.manaraa.com

6.4.4 Updating a Sensor Record

As stored sensor records are updated (with every received sensor record), it is necessary to ensure the consistency of the t_i, a_i values in all N records at all times. To achieve this, updating a sensor typically requires *three* records to be modified. More specifically, apart from the record for which an update is received, the a_i values in two *other* records may need to be modified. In some (non-typical) cases, the a_i value in one other record will need to be modified.

For example, consider an an update for S_5 with a new validity time 851 (to replace the old validity time 848). As a result, the auxiliary value in records S_2 and S_3 need to be modified. Specifically,

1. the value a_3 needs to be changed from 1002 to 851
2. the value a_2 needs to be changed from 848 to 850

As another example (of a non typical case), if an update for S_5 causes the validity time to be modified from 848 to 849, then a_2 should be modified from 848 to 849.

More generally, to update a record for an index u with current values (t_u, a_u) to a new expiry time t'_u, the algorithm is as follows:

1. determine the record index p such that $a_p = t_u$ (this record index p "points" to the record indexed u to be updated).
2. determine the record index c such that (t_c, a_c) *covers* t'_u. More specifically,
   (a) if $t_c < a_c$ then (t_c, a_c) covers t'_u if $t_c \leq t'_u \leq a_c$;
   (b) if $t_c > a_c$, then (t_c, a_c) covers t'_u if $t'_u > t_c$, *or* if $t'_u < a_c$;
   (c) if $t_c = a_c$, then (t_c, a_c) covers t'_u if $t'_u = t_c = a_c$.
3. If $u == c$, or in other words, if (t_u, a_u) itself covers the new value t'_u, then set $a_p = t'_u$; the value a_u remains unchanged in the updated record;

المنارة للاستشارات
www.manaraa.com

4. If $u \neq c$, then set i) $a_p = a_u$; ii) $a_u = a_c$; and iii) $a_c = t'_u$;

For the example where S_5 was updated to a new validity time 851, the indices u, p and c are respectively $u = 5$, $p = 2$ and $c = 3$. For the second example where S_5 was updated to a new validity time 849, the indices are $u = c == 5$, and $p = 2$.

6.4.5 Interface Update()

An interface Update() is used by the untrusted monitor $\mathbf{U}$ to submit new authenticated reports from sensors to the module $\mathbf{T}$ to cause the module to update the root of the tree.

To update the record for a sensor S_u the monitor $\mathbf{U}$ provides the following inputs:

1. the current leaf l_u for S_u (which indicates values ξ_u, t_u and a_u);
2. a new record for sensor S_u (authenticated by S_u through MAC μ_u), indicating time of expiry t'_u. This record will be verified by the module before it replaces the old leaf l_u.
3. a current leaf l_p (for sensor S_p) with $a_p = t_u$,
4. a current leaf l_c, with the pair (t_c, a_c) which covers t'_u.
5. a set of "complementary" hashes $\mathbf{v}_{u,p,c}$ to permit $\mathbf{T}$ verify the validity of the three current records against the root.

The module $\mathbf{T}$ processes the inputs as follows:

1. Verify MAC μ_u;
2. Verify $f_3(v_u, v_p, v_c, \mathbf{v}_{u,p,c}) = r$;
3. If $v_u == v_c$, modify l_p and l_u (to l'_p and l'_u) as follows
   (a) replace value a_p in leaf l_p with τ'_u; and
   (b) replace values ξ_u and τ_u in l_u with ξ'_u and τ'_u.
4. If $v_u \neq v_c$, modify l_p, l_u and l_c (to l'_p, l'_u and l'_c) as follows
   (a) replace (in l_u) ξ_u with ξ'_u, t_u with t'_u, and a_u with a_c;

المنارة للاستشارات
www.manaraa.com

(b) replace (in l_p) a_p with t'_u; and

(c) replace (in l_c) a_c with t'_u.

5. Compute $v'_u = h(l'_u), v'_p = h(l'_p), v'_c = h(l'_c)$;

6. Compute the new root as $r' = f_3(v'_u, v'_p, v'_c, \mathbf{v}_{u,p,c})$.

6.4.6 Interface FProof()

The interface FProof() is employed periodically by the monitor $\mathbf{U}$ to obtain a MAC to convince the alarm module to not trigger the alarm.

The monitor $\mathbf{U}$ uses this interface to submit proof that no record is stale, Specifically, the values provided as input are i) a leaf l_i with (t_i, a_i) where $a_i < t_i$, and ii) a set of complementary values $\mathbf{v}_i$ such that $f(v_i, \mathbf{v}_i) = r$ where $v_i = h(l_i)$.

The module $\mathbf{T}$ first verifies the integrity of the leaf l_i against r using $f()$; verifies that $t < a_i < t_i$ where t is the current time, and outputs a MAC $\mu = h(t_i \parallel K_A)$ where K_A is computed as $h(K \parallel A)$.

As illustrated in the example in Section 6.4.2, without the proposed strategy the module will have to perform $190 + 10000$ merkle tree operations every second - 190 leaf updates and 10000 leaf verifications per second. With the proposed strategy updating a sensor record using interface Update() may call for updating 2 or 3 leaves simultaneously. Even if we assume[3] that updating three leaves simultaneously is equivalent to updating 3 leaves independently the complexity for the proposed approach is $3 \times 190 + 1$ merkle tree operations per second.

[3] In general the number of hash operations required for updating three leaves simultaneously is lower than updating three leaves independently - especially if the common parent of the three leaves is well below the root.

المنارة للاستشارات
www.manaraa.com

Ensuring detection of misrepresentations of sensor states is however just one more step towards securing SCADA systems. Once it is ensured that the untrusted monitor cannot misrepresent (modify or hide) states $\{\xi_1 \cdots \xi_N\}$ of any sensor/actuator, mechanisms are required to verify that the state $\{\xi_1 \cdots \xi_N\}$ is a "valid" one.

المنارة للاستشارات

www.manaraa.com

CHAPTER 7

EVAULATING SCADA SYSTEM STATE

The module approach provided in Chapter 6 assures that all the sensor records are fresh. Only the presence of fresh records is not sufficient to determine the state of the system. A mechanism to build an evaluation function that denotes the entire state of a system from the available records and existing rules is addressed in this chapter.

For a SCADA system with N sensors/actuators, let $v_1 \cdots v_N$ represent the states of the sensors/actuators at a time t. The system is deemed to be in an acceptable state at time t if some function $f(v_1 \ldots v_n)$ evaluates to TRUE, where the function $f()$ is specified by the designer of the system. Else, an alarm should be triggered.

7.1 Principle of Operation

If sensor reports cannot be impersonated, and the module $\mathbf{T}$ can compute the function $f(v_1, v_2, \ldots, v_n)$ specified by the designer, the module $\mathbf{T}$ can then be convinced that the system is in an acceptable state if $f()$ evaluates to TRUE. However, that the non programmable module functionality should be usable for *any* SCADA system implies that we cannot make any assumptions regarding the type of function $f()$ or the number N of sensors/actuators in the system. Thus, for example, the module $\mathbf{T}$ may not be capable of storing all N current sensor reports.

المنارة للاستشارات
www.manaraa.com

In the proposed approach, the modules employ a Merkle hash tree [42] to *virtually* store the N current records as leaves of the Merkle tree. Only the root of the tree (a single hash) is stored inside the module $\mathbf{T}$. By performing simple sequences of hash operations the module $\mathbf{T}$ can verify the integrity of any leaf against the root stored inside. In addition, a novel concept of using "synthetic" records to specify relationships between leaves is used to evaluate the function $f()$.

For our purposes the leaves are sensor records, which can be inserted/updated only under certain conditions. A resource limited module $\mathbf{T}$ capable of i) storing the root, ii) performing simple sequences of hash functions to evaluate $h'()$, and iii) verifying simple pre-conditions necessary for updating a leaf, can provide integrity assurance to a large database of sensor records stored in an insecure location.

7.2 Sensor leaves and Synthetic Leaves

The leaves (sensor records) are stored by an untrusted computer $\mathbf{U}$. The module $\mathbf{T}$ stores only the root of the tree. More specifically, the Merkle tree employs two types of leaves, sensor leaves and synthetic leaves. Sensor leaves are of the form $l_i = i \parallel v_i \parallel t_i$ where i is a unique label for a sensor/actuator, v_i is a value reported by the sensor/actuator (the state of the sensor), and t_i is the time of expiry of the record. For a system with N sensors/actuators the tree possesses N sensor leaves.

Synthetic leaves do not correspond to sensor records. Instead, they are some function of two sensor records. A synthetic leaf is of the form $l_m^s = m \parallel v_m \parallel t_m \parallel OP \parallel i \parallel j$, where i and j refer to two leaf indexes, and OP refers to one of a small number of binary

www.manaraa.com

operations. The fields $OP \parallel i \parallel j$ is interpreted to imply that v_m is a function of leaves i and j. More specifically, $v_m = OP(v_i, v_j)$. For example, if OP is a code for $+$ then v_m is computed as $v_i + v_j$; if OP is AND, then $v_m = v_i \wedge v_j$, etc. While the time of expiry of sensor records are indicated in received sensor records, the time of expiry of synthetic records are set as the minimum of the time of expiry of its constituents. Or $t_m = \min(t_i, t_j)$.

A synthetic record may be a function of any two leaves (even leaves that contain synthetic records). It is reasonable to expect that a system with N sensors will require roughly N synthetic records to effectively specify the function $f()$. As a synthetic record includes a value which is a function of two other leaves, for a SCADA system with N real sensors we can expect (as a rough estimate) $N/2$ synthetic sensors (say $\mathbf{S}_1$) each specifying functions of two real sensors, $N/4$ synthetic sensors $\mathbf{S}_2$ each specifying functions of $N/2$ synthetic sensors $\mathbf{S}_1$, and so on till one synthetic record in a leaf l_n^s is a function of *all* sensors. The value v_n in l_n^s determines if the SCADA system is in an acceptable state. For example, only if $v_n = 1$, and $t_n > t$ (where t is the current time) will the alarm be suppressed. Note that the expiry time of t_n of l_s^n will be the minimum of expiry times of all sensor records.

When any sensor is updated, about $\log_2(N)$ synthetic sensor records will also need to be updated to ensure that l_n^s reflects the overall state. It is important to note that the "hook" which ensures that the untrusted $\mathbf{U}$ will have to submit all sensor records *and* update affected synthetic sensor records, is the expiry time. Unless all sensors are updated the expiry time will not be current (as l_s^n will have the least expiry time of all records) and hence the alarm cannot be suppressed.

المنارة للاستشارات
www.manaraa.com

7.3 Operation of module

The designer of a SCADA system initializes the Merkle tree which includes all leaves corresponding to every real and virtual sensor. In all real-sensor leaves the expiry time is set as a period in number of clock-tick counts for which the sensor can remain silent (without providing an update). For example, if the clock-tick frequency is 1 MHz, and sensor corresponding to l_5 is expected send a fresh measurement at least once in every 10 second interval, the expiry time t_5 will be set to 10 million.

The rules that govern acceptable states are specified as synthetic leaves. The leaf l_n^s which ultimately indicates if an alarm is necessary is given a special label 0. The specific choice of initial values $v_1 \cdot v_n$ is not important, except that they should be chosen to ensure that they do not reflect an alarm condition. The designer supplies the entire tree to the untrusted computer $\mathbf{U}$.

Before a module $\mathbf{T}$ can be used in a deployment, the module $\mathbf{T}$ requires to establish shared secrets with the designer of the system, the SCM associated with the MTU (which supplies all sensor records), and the alarm module. It is also assumed that some mechanism exists for the module $\mathbf{T}$ to indicate its current clock-tick value to the alarm module and SCM.

The module $\mathbf{T}$ expose a function INITIALIZE() which accepts the root r_0, duly authenticated by the designer. When this interface is called the module $\mathbf{T}$ sets the root to r_0, and its clock-tick counter to zero, and instructs the alarm module and SCM to set their counters to zero. From this point onwards, the SCM will modify sensor records to indicate time of expiry in terms of the clock tick count of the module $\mathbf{T}$, and outputs such sensor

المنارة للاستشارات
www.manaraa.com

records authenticated for verification by module $\mathbf{T}$. The sensor records are made available to $\mathbf{U}$.

7.3.1 Module Interfaces

To update a real sensor record the module $\mathbf{T}$ exposes an interface UPDATE() which expects as inputs i) a record l_i' authenticated by the SCM along with ii) the current record l_i, and iii) a set of hashes $\mathbf{v}_i$ such that $r = h'(l_i, \mathbf{v}_i)$. The module $\mathbf{T}$ incorporates the new record by modifying the root to reflect the new record as $r = h'(l_i', \mathbf{v}_i)$.

To update a synthetic record the module $\mathbf{T}$ exposes an interface UPDATESYN() which expects the following inputs: i) a current current synthetic leaf l_m^s along with ii) hashes to prove l_m^s against the root; iii) leaves l_i and l_j corresponding to indexes specified in the synthetic leaf l_m^s along with ii) hashes to prove the integrity of leaves l_j and l_i against the root of the tree. In response the module $\mathbf{T}$ a) computes $v_m = OP(v_i, v_j)$; b) sets the expiry time t_m to $t + m = \min(t_i, t_j)$; and c) updates the root to reflect the modified l_m^s.

At a time t (when the clock-tick counter value of the module $\mathbf{T}$ is t), to convince the module $\mathbf{T}$ that the system is in an acceptable state;
$\mathbf{T}$ exposes an interface SILENCEALARM() which expects as inputs i) the special synthetic leaf l_n^s, and ii) hashes to map the leaf to the root. If $v_n^s = 1$ and $t_n > t$, the module $\mathbf{T}$ outputs a MAC for the value t_n verifiable by the alarm unit to indicate that the alarm should be suppressed till time t_n.

It is the responsibility of $\mathbf{U}$ to submit duly authenticated sensor records using the interface UPDATE() and update synthetic sensor records by using interface UPDATESYN().

المنارة للاستشارات
www.manaraa.com

Unless $\mathbf{U}$ manges to keep the synthetic records updated, it cannot use the interface SILENCEALARM() to silence the alarm. It is also the responsibility of $\mathbf{U}$ to maintain the leaves of the Merkle tree and the intermediate nodes, and modify the intermediate nodes whenever any leaf is updated. However, at the risk of sounding repetitious, we still do not need to trust $\mathbf{U}$ as failure to do its job will result in an alarm. Similarly, jamming attacks which could result in preventing sensor records from reaching the MTU will not be able to suppress the alarm. Furthermore, as the rules specified by the designer (as synthetic records) cannot be modified, the fact that the system is in an unacceptable state *cannot* be hidden from the module $\mathbf{T}$.

The possibility of hidden malicious functionality in complex components mandates strategies to secure critical systems that relies only on components in which we can rule out HMF. We have outlined an approach to secure SCADA systems by relying on a trustworthy module of trivial complexity, which can be realized in a trusted environment.

In the proposed model it is the responsibility of the untrusted computer $\mathbf{U}$ to ensure that all sensor records are submitted to module $\mathbf{T}$ using interface UPDATE() and that all synthetic records are updated (using interface UPDATESYN()), and that they remain updated at all times. If $\mathbf{U}$ fails to do so, the time of expiry of the leaf l_n^s cannot be updated, and thus the alarm cannot be silenced.

Updating every sensor record will require updates to about $\log_2 N$ synthetic sensor records. As updating any record requires $\mathbb{O}(\log_2 N)$ hash operations, the complexity for updating a sensor record is $\mathbb{O}(\log_2^2 N)$. For example, for a complex SCADA system with

المنارة للاستشارات
www.manaraa.com

million sensors (or $log_2 N \approx 20$), updating any sensor will require of the order of $20 \times 20 = 400$ hash operations.

The module $\mathbf{T}$ does not care about the specifics of *how* the deployment operates. The task entrusted to module $\mathbf{T}$ is clear and simple - to verify that the physics of the process is consistent with the intent of the designer. The personnel who translate the design into a deployment (for example by writing software logic in PLCs) may choose to impose rules that are stricter than the rules specified by the designer - as long as they do not violate the rules specified by the designer. Irrespective of logic used to control the process, an alarm will ensue if the state of the process is in an unacceptable state as deemed by the designer.

Another beneficial side effect of the proposed strategy stems from it's loose relationship with deployment details. Deployed systems can be easily upgraded as demanded by ever changing technologies. As mot SCADA system components are not trusted, the security mechanism is not affected by changes to such components. The security mechanisms also does not interfere with, or is unaffected by, any other security strategy that may be used in addition. cannot use the interface SILENCEALARM() to silence the alarm. It is also the responsibility of $\mathbf{U}$ to maintain the leaves of the Merkle tree and the intermediate nodes, and modify the intermediate nodes whenever any leaf is updated. However, at the risk of sounding repetitious, we still do not need to trust $\mathbf{U}$ as failure to do its job will result in an alarm. Similarly, jamming attacks which could result in preventing sensor records from reaching the MTU will not be able to suppress the alarm. Furthermore, as the rules specified by the designer (as synthetic records) cannot be modified, the fact that the system is in an unacceptable state *cannot* be hidden from the module $\mathbf{T}$.

المنارة للاستشارات
www.manaraa.com

CHAPTER 8

A SECURITY ARCHITECTURE FOR SCADA SYSTEMS

Our research led to identify the roles of untrusted middleman in distinct PMC systems. The importance of identifying a minimal TCB for critical Infrastructure systems is discussed in Chapter 3. As mentioned, depending upon the nature of parameters like data, identifiers, application domain, two types of TCB modules 1. Stateful, 2. Stateless are identified.

Atomic relay functionality provided in Chapter 4 serves the purpose of a stateless TCB. A stateful TCB solution for generic data dissemination system (which can also handle dynamic DNS) is provided in Chapter 5.

The strategies for designing a TCB for SCADA systems discussed in Chapters 6, 7 do not take most of the practical issue into account, and merely include some components of the TCB functionality required to secure SCADA systems.

This chapter provides a complete specification of an architecture for securing SCADA systems. This architecture, based on a trusted hardware module - which we refer to as a SCADA TCB (STCB) module - is intended to be usable for any SCADA system - irrespective of the nature and size of the system. The STCB based security architecture will include specifications for a) the functionality of STCB modules; b) processes to be adopted

www.manaraa.com

by the *designer* and the *deployer* of the system; and c) an STCB protocol, for updating the state of STCB modules, and obtaining SCADA state reports.

8.1 Overview of STCB Approach

The STCB security model emerges from a broad perspective of a SCADA system, where a *stake holder* desires to have an accurate picture of the state of a CI system. A SCADA system is the *agent* employed by the stake holder to control and report the state of the system.

The state reports from a SCADA system can be seen as a function of the current states of all sensors associated with the system. For a SCADA system characterized by n sensors, let $v_1 \cdots v_n$ represent the states of the n sensors, and let

$$[o_1 \cdots o_s] = \mathcal{F}(v_1 \cdots v_n) \tag{8.1}$$

represent a function that captures the "physics" of the controlled system, and reports values $[o_1 \cdots o_s]$ to the stake-holder as the "state of the system." In practical SCADA systems evaluation of $\mathcal{F}()$ is performed jointly by numerous untrusted SCADA system components that may include PLCs in multiple RTUs and MTUs, the HMI, and even actions by human operators. Consequently, the integrity of the state reports are far from assured.

The goal of the STCB security model is to *guarantee the integrity of state reports provided by the agent*. To achieve it's goals, the STCB security model relies only on a) the integrity of STCB modules, and b) the integrity of clearly defined processes to be adopted by entities identified as the *designer* and the *deployer* of the SCADA system.

www.manaraa.com

The designer is an entity with good domain knowledge (regarding the CI system); the deployer is a security professional who is not required to possess any knowledge of the CI system. To the extent the stake-holder trusts the integrity of the STCB modules, and the verifiable processes adopted by the designer and the deployer, the stake-holder is assured of the integrity of the state report — even if malicious functionality may exist in SCADA system components.

Current approaches to secure SCADA systems often possess features like a) cryptographic protection of links between RTUs and MTUs [31, 38, 53, 64, 74, 80] to prevent message injection attacks by attackers and b) intrusion detection systems to facilitate early detection of attacks [13, 47, 83]. Such efforts do not however address attacks that exploit hidden functionality in SCADA system components or the IDS.

STCB based security will not interfere with any security mechanisms that may already exist. It is a passive approach which merely monitors and evaluates the state of a SCADA system. More specifically, the STCB approach can *not* prevent hidden functionality from being exploited. Good security practices are still necessary to a) reduce the possibility of hidden functionality, and to b) make it difficult for attacker to be able to actually exploit such functionality. What the STCB approach guarantees is that *if* some malicious functionality is exploited to drive a CI system into unacceptable states (the definition of which is specified by the designer of the system), the fact that the system is in an unacceptable state can *not* be hidden from the stake-holder.

The STCB approach does *not* address attacks on the physical sensors themselves. It is reasonable to assume that it is impractical for attackers to introduce hidden malicious

المنارة للاستشارات
www.manaraa.com

functionality in extremely simple components like sensors that can be remotely exploited. Thus, traditional physical security measures are assumed to be adequate for protection of sensors.

8.1.1 STCB System Components

The additional components introduced into a STCB-secured SCADA system include

1. an *untrusted* "STCB system manager" **U**,
2. STCB modules $\{M_1 \cdots M_k\}$, and M_0.

All STCB modules are identical, and are capable of executing a set of simple TCB functions. Modules $M_1 \cdots M_k$ are "closely bound" to SCADA system sensors.

In the rest of this chapter we shall use the term sensor module (SM) for STCB modules $M_1 \cdots M_k$, and the term central module (CM), for STCB module M_0. The untrusted STCB manager **U** periodically receives *sensor reports* from SMs $M_1 \cdots M_k$ and makes them available to CM M_0. CM M_0 evaluates $\mathcal{F}()$, and outputs *state reports*.

From a broad perspective, the authenticity of the *inputs* to $\mathcal{F}()$ are assured by SMs $M_1 \cdots M_k$; the integrity of the function $\mathcal{F}()$ is assured by the CM.

The exact make up of the manager **U** is irrelevant for our purposes of guaranteeing the integrity of $\mathcal{F}()$, as **U** is not trusted. Unless **U** performs it tasks faithfully, valid state reports can *not* be sent to the stake holders.

The state reports are relayed by the STCB manager **U** to an STCB module M_r associated with a stake-holder. Any number of stake holder modules like M_r may exist. More generally, a stake-holder module may be the CM for another STCB deployment.

المنارة للاستشارات
www.manaraa.com

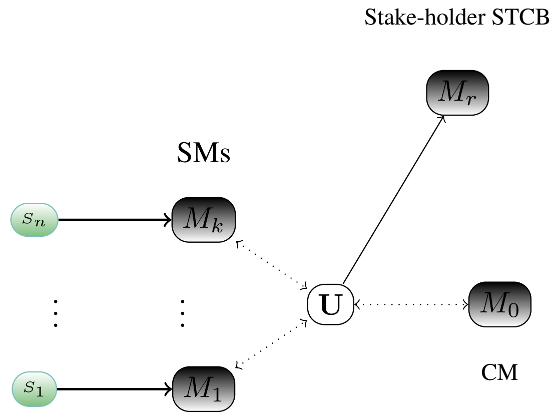

Figure 8.1

STCB components

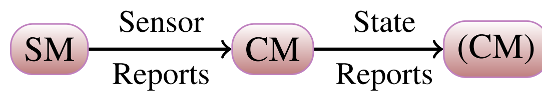

Figure 8.2

Information flow in the STCB model

المنارة للاستشارات
www.manaraa.com

For example, the state reports from different SCADA systems may be provided as "sensor reports" to a system at a higher level of hierarchy. In such a scenario, the stake-holder module M_r can be seen as the CM of an STCB deployment at a higher level of hierarchy. Module M_r considers the state reports from the CMs of systems at the lower level as "sensor reports" from *foreign* STCB deployments.

Any number of hierarchical levels may exist. For example, state reports from multiple SCADA systems in a town may be inputs to a single SCADA system that monitors the health of all such systems in the town. The reports from such SCADA systems in different towns may be inputs to another SCADA system at an even higher level of hierarchy, that monitors the health of all systems in a state, and so on.

8.1.2 Evaluating $\mathcal{F}()$

The main challenge lies in the choice of a strategy for evaluating *any* $\mathcal{F}()$ (which can be substantially different for different SCADA systems) inside the trusted confines of severely resource challenged STCB modules. Recall that we desire to *deliberately* constrain STCB modules to possess only modest memory and computational abilities. Consequently, we constrain STCB modules to perform only logical and cryptographic hash operations. By performing simple logical operations the STCB modules support a simple instruction set $\mathcal{A}$.

As no restrictions are placed on the nature and scale of the SCADA system, some of the specific challenges are that

1. the number of sensors n (also the size of inputs to $\mathcal{F}()$) can be unlimited;

المنارة للاستشارات
www.manaraa.com

2. evaluation of $\mathcal{F}()$ may require evaluation of complex functions, and thus challenging to represent using merely the instruction set $\mathcal{A}$.
3. reports corresponding to different sensors may arrive asynchronously (necessitating frequent re-computation of $\mathcal{F}(v_1 \cdots v_n)$).

All such challenges are addressed either directly or indirectly through the use of Merkle trees [42].

8.1.2.1 Merkle Trees in the STCB Approach

In the STCB approach resource challenged STCB modules store only the root of the tree, and have the ability to perform $f_v()$ operations. This capability is leveraged to assure the integrity of

1. a dynamic database of n sensor measurements;
2. any number of simple static "algorithms" to evaluate $\mathcal{F}()$; the algorithms are small number of instructions (belonging to the instruction set $\mathcal{A}$) supported by STCB modules; and
3. static *look-up tables* (of any size) for evaluating complex functions in $\mathcal{F}()$.

Specifically, the STCB module M_0 for an STCB deployment stores a (static) root of a static merkle tree, and the (dynamic) root of a dynamic merkle tree. The leaves of the static tree are the specifications for a specific STCB deployment — provided by the designer and the deployer of the system. The leaves of the dynamic tree are the current states of the n sensors of the system. The leaves and all intermediate nodes of both trees, are stored by the untrusted STCB manager **U**.

www.manaraa.com

8.1.3 STCB Designer and Deployer

One of the main motivations for clearly demarcating between the roles of a designer and a deployer is that entities with good domain knowledge (for example, an entity with in-depth knowledge about the domain of specific CI system, like a nuclear plant) are often unlikely to be security experts. Likewise, security experts are unlikely to be experts in the domain of the specific CI system.

In the STCB security model, the designer is a domain expert with good knowledge of the CI system. The designer is required to be aware of the purpose of each sensor in the system, and the interpretation of their states. For example, (say) in a water-tank control system, "if $S_5 > 100$ (water level greater than 100) S_6 should be zero (the pump should be off)." The responsibility of the designer is to come up with a specification for the function $\mathcal{F}(v_1 \cdots v_n)$ that captures the physics of the system.

The deployer is a security professional who may not possess any CI system domain knowledge. The responsibility of the deployer is to procure and install STCB modules. The deployer needs to be aware of steps to be taken, for example, to

1. facilitate establishment of shared secrets between modules;
2. securely connect (for example, using tamper-evident connectors) physical sensor outputs to SMs, and record such bindings (for example, (S_5, M_8) indicating that sensor S_5 is connected to module M_8);
3. deploy the STCB manager **U** — which includes installation of all hardware/software necessary to relay SM outputs to the STCB manager **U**, setting up a channel to the CM M_0, and a channel to be used for conveying state reports to stake holders. However, **U**, and such channels, are *not* trusted.

المنارة للاستشارات
www.manaraa.com

8.2 STCB Design

The designer is entrusted with the responsibility of describing function $\mathcal{F}()$ in a special manner comprehensible by resource limited STCB modules. Specifically, due to the asynchronous nature of the sensor reports, it is inefficient to evaluate $\mathcal{F}()$ *entirely* every time a fresh report from any sensor is available. Consequently, for a system with n dynamic inputs (corresponding to sensors $S_1 \cdots S_n$) evaluation of $\mathcal{F}()$ is realized as

$$\mathcal{U}_1() \circ \mathcal{U}_2() \circ \cdots \circ \mathcal{U}_n(), \tag{8.2}$$

where $\mathcal{U}_i, 1 \leq i \leq n$ is evaluated whenever an fresh measurement from sensor S_i is available. It is the responsibility of the designer of the system to translate the function $[o_1 \cdots o_m] = \mathcal{F}(v_1 \cdots v_n)$ into an appropriate set of functions $\mathcal{U}_i$, $1 \leq i \leq n$.

8.2.1 STCB Design Tree

The designer provides a specification of $\mathcal{U}_i, 1 \leq i \leq n$ by constructing a static merkle tree — the design tree — with root ξ_s. The tree includes $n + 1$ leaves

$$\xi_0, \mathbf{G}_1 \cdots \mathbf{G}_n \tag{8.3}$$

where ξ_0 is itself a root of a merkle tree with n leaves that specify the *initial* state of n sensors as records $\mathbf{S}_i, 1 \leq i \leq n$. The other n leaves correspond to the n design records $\mathbf{G}_i, 1 \leq i \leq n$. The sensor records and design records are of the form

$$\mathbf{S}_i = [S_i, t_i, o_{i_1} \cdots o_{i_w}, \tau_i]$$

$$\mathbf{G}_i = [S_i, S_{i,1} \cdots S_{i,q}, \alpha_i, \lambda_i, S_i'] \tag{8.4}$$

المنارة للاستشارات
www.manaraa.com

Each sensor is associated with a set of $w + 3$ *dynamic* values. Specifically, the value v_i is the latest measurement of sensor S_i, and t_i is the time of the measurement. $o_{i_1} \cdots o_{i_w}$ are the w outputs of function $\mathcal{U}_i$ that is evaluated whenever a fresh report is available from sensor S_i. The value τ_i is a measure of time associated with the outputs $o_{i_1} \cdots o_{i_w}$.

In the design record $\mathbf{G}_i$, the sensor identity S_i conveys that $\mathcal{U}_i$ is to be evaluated whenever a fresh report from S_i is available. The values $S_{i,1} \cdots S_{i,q}$ specify up to q *related* sensors. Such sensors are "related" to S_i as the states of such sensors can influence $\mathcal{U}_i$. Some or all of the q values $S_{i,1} \cdots S_{i,q}$ can be set to zero if less than q related sensors suffice.

The value $\alpha_i = h(\mathbf{A}_i)$ is the hash of a small number (say, m) of instructions chosen from the set $\mathcal{A}$. The value λ_i is a one way function of a set of (say, l) constants $\mathbf{C}$. Such constants may specify various values like set-points, permitted ranges of measurements, minimum expected frequency of reports from sensors, etc. In addition, such constant values may also be used as look up tables. The value S'_i is optional, and is the identity of a "synthetic" sensor (explained later).

8.2.2 Inputs and Outputs of $\mathcal{U}_i$

Due to limited memory inside STCB modules, there is a need for a strict upper bound on the number of inputs to, and outputs of, each $\mathcal{U}_i$. In other words, irrespective of the total number of sensors n, note that $\mathcal{U}_i$s are restricted to specifying only a) up to q related sensors, b) l constants, c) m instructions, and d) one synthetic sensor as inputs. Each $\mathcal{U}_i$ produces w outputs.

المنارة للاستشارات
www.manaraa.com

As $\mathcal{U}_i$ is re-evaluated whenever a fresh report $\tilde{v}_i, \tilde{t}_i$ is available from sensor S_i, the inputs necessary to evaluate $\mathcal{U}_i$ are stored in reserved volatile registers inside STCB modules, and include

1. values in the record $\mathbf{S}_i$ associated with sensor S_i (stored in a register $\mathbf{s}_0$ inside the module);
2. values $\tilde{v}_i, \tilde{t}_i$ in a fresh report from sensor S_i (register $\mathbf{r}$);
3. values in records $\mathbf{S}_{i,1} \cdots \mathbf{S}_{i,q}$ for related sensors $S_{i,1} \cdots S_{i,q}$ (registers $\mathbf{s}_1 \cdots \mathbf{s}_q$);
4. l constants in $\mathbf{C}_i$ (register $\mathbf{c}$); and
5. m instructions $\mathbf{A}_i$ (some of which may be set to 0 to represent "no operation" if m instructions are not required to evaluate $\mathcal{U}_i$);

The m logical operations in $\mathbf{A}_i$ provide the instructions to recompute the outputs $\tilde{o}_{i_1} \cdots \tilde{o}_{i_w}$ of $\mathcal{U}_i$ following a fresh report from S_i. On evaluation of $\mathcal{U}_i$ the record $\mathbf{S}_i$ is modified. Specifically,

1. $\tilde{v}_i, \tilde{t}_i$ replace the previous values (v_i, t_i),
2. outputs $\tilde{o}_{i_1} \cdots \tilde{o}_{i_w}$ replace outputs $o_{i_1} \cdots o_{i_w}$ of the previous execution of $\mathcal{U}_i$,
3. and τ_i is replaced with

$$\tilde{\tau}_i = \min(\tau_{i_1} \cdots \tau_{i_q}, \tilde{t}_i) \tag{8.5}$$

   to reflect the *staleness* of the w outputs.

Note that dynamic values associated with any S_i may be affected not just by values corresponding to sensors directly related to S_i, but also sensors *indirectly* related to S_i — for example sensors related to a related sensor S_j (once removed) or sensors related to a sensor related to S_j (twice removed) and so on. Computing the value τ as in Eq (8.5) ensures that the value τ_i will be the least of the sensor-report time t corresponding to every sensor that is directly or indirectly related sensor S_i.

للاستشارات www.manaraa.com

On a continuous basis, as and when new sensor reports are available, the states of the reporting sensors are modified. A subset of dynamic values corresponding to a subset of sensors may be reported to the stake-holder as values $o_1 \cdots o_m$ describing the state of the system. For example, if value o_{j_2} (second output of $\mathcal{U}_j$) is one of the values reported as the state of the system, the time associated with the state o_{j_2} is reported as τ_j.

8.2.2.1 Synthetic Sensors

The sensors $S_1 \cdots S_n$ can be of three types — *real* sensors, *state-report* sensors, and *synthetic* sensors.

Real sensors are physical sensors in the SCADA deployment. Specifically, during the STCB deployment phase, real sensors are bound to SMs.

State reports from a foreign STCB system are seen by the receiving CM as a "sensor" report; as such reports are authenticated by the CM of the foreign deployment, state-report sensors are bound to foreign CMs.

Synthetic sensors are *not* bound to CMs or SMs. In a design record $\mathbf{G}_i$, if $S'_i \neq 0$, implies that evaluation of $\mathcal{U}_i$ results in the "synthesis of a fresh report from a (synthetic) sensor $S'_i = S_j$." Just as a fresh report from a sensor S_i should be followed by evaluation of $\mathcal{U}_i$, a fresh report from synthetic sensor $S_j = S'_i$ should be followed by evaluation of $\mathcal{U}_j$.

The primary motivation for using such synthetic sensors is to cater for complex $\mathcal{U}_i$ where the fixed number of (m) instructions in $\mathbf{A}_i$ may be insufficient. By specifying a synthetic sensor $S_j = S'_i$, evaluation of $\mathcal{U}_i$ is *continued* as evaluation of $\mathcal{U}_j$. Similarly,

المنارة للاستشارات
www.manaraa.com

evaluation of $\mathcal{U}_j$, specified by the designer as $\mathbf{G}_j = [S_j, S_{j,1} \cdots S_{j,q}, \alpha_j, \lambda_j, S'_j]$ may be continued again, if necessary, by specifying $S'_j \neq 0$.

8.2.2.2 Constants and Look-Up Tables

In general, the value λ_i — which is a one way function of constants required to evaluate $\mathcal{U}_i$ — may be a function of *multiple* sets of l constants (l constants in each set). More specifically, λ_i is itself the root of a merkle tree, where each leaf specifies a set of l constants. Any number of such leaves may exist, with a minimum of one.

Permitting an unlimited number of constants facilitates the use of look-up tables (LUT) for evaluating $\mathcal{U}_i$. An LUT for evaluating a complex function $y = f(x)$ will have many sets of l constants — say $C_{j,1} \cdots C_{j,l}$ where there are no practical limits on j. In each set $c_1 = C_{j,1} \cdots c_l = C_{j,l}$ two of the l constants will specify the range of the independent variable x, and one will specify the corresponding dependent variable y. For 2 dimensional LUTs of the form $y = f(x_1, x_2)$, four of the l constants will specify the ranges for the two independent variables, and a fifth constant will specify the corresponding value of y.

Special instructions (say LUT1 and LUT2) in the instruction set $\mathcal{A}$ will specify the operands – the dependent and independent variables. As one possible design of the two instructions, instruction LUT1 interprets constants c_1 and c_2 as the range of the independent variable x and constant c_3 as the corresponding dependent variable y. Before the module executes the instruction LUT1, it expects the value of the input operand to be within the range of constants c_1 and c_2 — else the execution will not proceed. If the input operand satisfies the requirement, then the value of the output operand is set to c_3. Similarly, for

المنارة للاستشارات
www.manaraa.com

LUT2, constants c_1 and c_2 specify the range of the first input operand x_1; c_3, c_4 specify the range of the second operand x_2; c_5 is the corresponding output y.

8.2.3 Instruction Set $\mathcal{A}$

Each instruction in $\mathcal{A}$ specifies a logical operation (opcode), input operands (1, 2 or 3) depending on the type of opcode, and an output operand. The operands are restricted to be values in STCB registers $\mathbf{s}_0 \cdots \mathbf{s}_q$, $\mathbf{c}$, $\mathbf{r}$, etc. Specifically, as the instructions in each $\mathcal{U}_i$ can modify only values in register $\mathbf{s}_0$ (the current state of sensor S_i when $\mathcal{U}_i$ is computed), only such values, and a temporary register T can be specified as output operands.

Examples of simple logical operations include traditional operations like addition, logical operations, bit-wise operations, COPY, MOV, etc., and some special instructions like LUT1 and LUT2. Other potentially useful special instructions for SCADA systems is a bounds checking operation CHKB which checks a specific value is within set-points specified as constants and tolerance checking TOL where two values are verified to be close enough — within a tolerance specified by a third value.

Ultimately, a comprehensive specification for STCB modules will fix values like the number of related sensors q, number of outputs w, and the number of constants l (and hence the number of addressable values in the STCB registers). Such a specification will also include a detailed listing of all permitted opcodes and their interpretation. This chapter, however, is restricted to describing some of the salient features of STCB modules.

المنارة للاستشارات
www.manaraa.com

8.3 STCB Deployment

The deployer of the SCADA system is trusted to verify the integrity of the physical bindings between various sensors and SMs. Specifically, the deployer is required to permanently connect the outputs of every sensor to an SM, and apply tamper-evident seals to such connections. The deployer specifies *binding* records of the form

$$\mathbf{B}_i = (S_i, M_j, \epsilon_i, \theta) \tag{8.6}$$

to convey that

1. measurements corresponding to sensor S_i will be reported by a module M_j; if S_i is a real sensor, then M_j is an SM; if S_i is a state-report sensor, then M_j is the identity of the CM of a foreign system. Recall that synthetic sensors are not bound to CMs or SMs and thus have have no binding records. The deployer may be totally oblivious of the existence of such records.
2. ϵ_i is an *achievable minimum round-trip duration* between module M_j, and the CM for the deployment, and
3. $\theta = 0$ implies a record corresponding to a real sensor; $\theta \neq 0$ implies report from a foreign STCB system with STCB descriptor $\xi_{sp} = \theta$.

To indicate that a module M_0 was deployed as the CM for the STCB system, the binding records includes a record for the CM M_0 as $\mathbf{B}_0 = (S = 0, M_0, \epsilon = 0, \theta = 0)$.

Depending on the requirements specified by the stake holder, the deployer also specifies *reporting* records of the form

$$\mathbf{R}_j = (S_r, M_r, S_j, l), 1 \leq l \leq w. \tag{8.7}$$

to indicate that the value o_{j_l} (corresponding to sensor S_j) should be reported to the stakeholder module M_r, and that the report should indicate o_{j_l} as the latest (time τ_j) "measurement" from state-report sensor S_r.

للاستشارات المنارة
www.manaraa.com

All such $\mathbf{B}_i$ and $\mathbf{R}_j$ are included as leaves of a static Merkle hash tree constructed by the deployer — the STCB *deployment tree*. Let ξ_p be the root of the deployment tree with leaves

$$\{\cdots \mathbf{B}_i \cdots\}, \{\cdots \mathbf{R}_j \cdots\} \tag{8.8}$$

The end-result of the design and deployment processes are two hash trees with roots ξ_s and ξ_p. The design root ξ_s can be seen as concise representation of the physics $\mathcal{F}(v_1 \cdots v_n)$ of the system. The deployment root ξ_p is a concise representation of the bindings between real-sensors & SMs, and state-report-sensors & CMs of foreign STCB deployments. The value

$$\xi_{sp} = h(\xi_s, \xi_p) \tag{8.9}$$

can now be seen as the root of a merkle tree with two sub-trees — the design tree to the left, with root ξ_s, and the deployment tree to the right, with root ξ_p. The static value ξ_{sp} is the unique descriptor for a specific STCB deployment (deployed to secure a specific SCADA system).

Note that two different deployments of identical SCADA systems may have the same design root ξ_s, but will have different deployment root ξ_p as different STCB modules will be used in the two deployments. If $h()$ is collision resistant, no two STCB deployments will have the same descriptor ξ_{sp}.

للاستشارات المنارة
www.manaraa.com

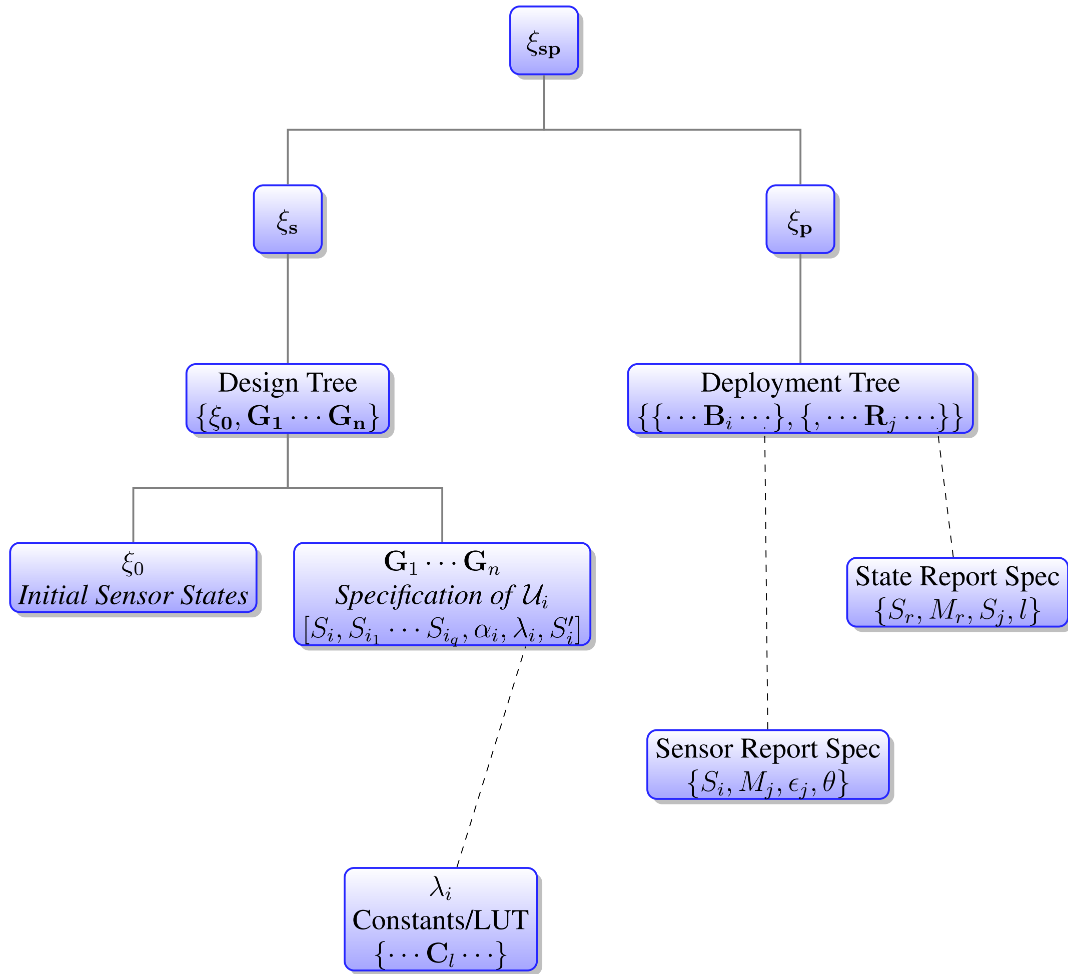

Figure 8.3

Static Descriptor ξ_{sp} – a specification for an STCB system

www.manaraa.com

8.3.1 STCB Operation

To commence operation in a SCADA deployment, the STCB module M_0 associated with the system is initialized with the STCB descriptor ξ_{sp}, and the value ξ_0 corresponding to the initial state of sensors. All leaves of the tree with root ξ_{sp} (which includes two sub-trees — the design tree and the deployment tree) are stored by $\mathbf{U}$; all n leaves of the form $\mathbf{S}_i, 1 \leq i \leq n$ corresponding to the initial states of all sensors are also stored by $\mathbf{U}$.

During regular operation of the system the STCB manager $\mathbf{U}$ receives authenticated sensor reports from SMs (and possibly CMs from foreign deployments), and submits them one at a time, to the CM M_0. Such reports take the form of a *message authentication code* (MAC) computed as

$$\mu = h(S, v, t, c, \xi_{sp}, K'), \tag{8.10}$$

where

1. values (S, v, t) indicates a report for sensor S to convey a fresh measurement v and measurement time t.
2. c is the clock-tick value of the module that created the report;
3. ξ_{sp} is STCB descriptor of the creator of the report (which was the value used to initialize the module);
4. K' is a shared secret between the creator (SM or foreign CM) and receiver (CM M_0).

Along with the report from sensor S_i (authenticated by a module M_j), $\mathbf{U}$ also submits a binding record $\mathbf{B}_i$ constructed by the deployer, consistent with the static root ξ_{sp}.

The STCB manager $\mathbf{U}$ is then required to submit other values required for the CM to evaluate $\mathcal{U}_i$. Such values include a) sensor state record $\mathbf{S}_i$, b) the state records of the (up to)

المنارة للاستشارات
www.manaraa.com

q related sensors; c) a design record $\mathbf{G}_i$ consistent with descriptor ξ_{sp}; d) l constant values $\mathbf{C}_i$; and e) m instructions $\mathbf{A}_i$.

As dynamic sensor states associated with the n sensors are maintained by $\mathbf{U}$ as leaves of dynamic a merkle tree with root ξ, sensor state records provided by $\mathbf{U}$ will be accepted as valid only if they can be verified to be consistent with a copy of the dynamic root ξ stored inside the CM M_0. The instructions $\mathbf{A}_i$ will be accepted as valid only if $\alpha_i = h(\mathbf{A}_i)$ where α_i is specified in the design record $\mathbf{G}_i$. The set of constants $\mathbf{C}_i$ will be accepted as valid only if $h(\mathbf{C}_i)$ can be demonstrated to be a node in the binary tree with root λ_i — where λ_i is specified in the design record $\mathbf{G}_i$.

The values used for evaluating $\mathcal{U}_i$ have fixed reserved locations in the internal memory of the STCB modules, and are specified as the operands for the m instructions in $\mathbf{A}$. An internal module function $f_{eval}(\mathbf{A}_i)$ executes every instruction sequentially. The end result is the modification of the values in the sensor state record $\mathbf{S}_i$ of S_i. To "remember" changes such changes to $\mathbf{S}_i$ the module modifies the dynamic root ξ.

If untrusted $\mathbf{U}$ does *not* modify $\mathbf{S}_i$ in the same manner, then $\mathbf{S}_i$ will not longer be consistent with the root ξ stored inside the CM. Thus, *if* $\mathbf{U}$ provides a fresh sensor report to the CM to invoke $\mathcal{U}_i$, it is forced to modify the state of S_i exactly in the manner specified by the designer.

At any time, a reporting record $\mathbf{R} = (S_r, M_r, S_j, l \in \{1 \cdots w\})$ consistent with ξ_{sp} can be provided as input, along with the state record $\mathbf{S}_j$ consistent with ξ, to request the STCB module M_0 to report (to module M_r) values o_{j_l} and τ_l in a state-report "sensor" S_r.

المنارة للاستشارات
www.manaraa.com

If **U** does not invoke functions to evaluate any $\mathcal{U}_i$, the time t_i associated with S_i can not be updated. Thus, in any state-report that directly or indirectly depends on sensor S_i the time τ will be stuck at t_i, and will thus be recognized as stale by the stake-holder (or CM of a foreign deployment).

8.3.1.1 STCB Interfaces

To interact with the STCB modules (SMs and CM), **U** employs various interfaces exposed by the modules. An interface $F_{peer}()$ is used to set up shared keys between modules that send and receive sensor/state reports. A function $F_{hs}()$ can be invoked to engage two modules in a hand-shake sequence withing a duration ϵ, to enable the modules to estimate their respective clock offsets with an error less than ϵ. Interface $F_{init}()$ is used to initialize a module as a CM for a deployment with identifier ξ_{sp}, or as a SM for a specific *real* sensor S_{in} (in an STCB deployment with identifier ξ_{sp}).

A function $F_{snd}()$ is invoked to request a module to send a report to another module, in which $F_{rcv}()$ is invoked to accept the report. Function $F_{ld}()$ is used to load various values necessary to evaluate some $\mathcal{U}_i$. Function $F_{upd}()$ is then invoked to evaluate $\mathcal{U}_i$ and update the root ξ stored inside the module. Functions $F_{rcv}(), F_{ld}$ and $F_{upd}()$ are utilized only in modules used as CMs.

To the extent the SMs and the deployer (who is trusted to verify and specify bindings between sensors and SMs) are trusted, we can trust the authenticity of sensor reports provided to the CM module. To the extent the CM module M_0, and the designer (who is re-

المنارة للاستشارات
www.manaraa.com

sponsible for specifying the functions $\mathcal{U}_i()$) are trusted, the stake-holder trusts the integrity of the state reports.

In practice, the SMs will need to be located as close as possible to the sensors to improve the security of the binding between sensors and SMs. The CM could be housed in any location — for example, a secure location far removed even from the SCADA control center. Components of **U** will need to be housed close to SMs, *and* close to the CM.

8.4 STCB Architecture

STCB modules have a unique identity, and a unique secret issued by a trusted key distribution center (KDC). Two modules (say) M_i and M_j can use their respective secrets (say) Q_i and Q_j issued by the KDC to compute a common pairwise secret K_{ij} [49]. Specifically, associated with a pair of modules M_i, M_j is a non-secret value P_{ij} which is also made available by the KDC (for example, in a public repository), where P_{ij} is computed as

$$P_{ij} = h(Q_i, M_j) \oplus h(Q_j, M_i). \tag{8.11}$$

Module M_i and M_j can compute a common secret $K_{ij} = h(Q_i, M_j)$,

$$K_{ij} = \begin{cases} h(Q_i, M_j) \oplus 0 & \text{Computed by Module } M_i \\ h(Q_j, M_i) \oplus P_{ij} & \text{Computed by Module } M_j \end{cases} \tag{8.12}$$

Every module possesses three values that are affected whenever a module is powered on:

1. a clock tick counter c, which is set to 0;
2. a non-volatile session counter σ, which is incremented; and
3. a random secret γ, which is spontaneously generated inside the module.

المنارة للاستشارات
www.manaraa.com

8.4.1 Module Registers

Non-volatile storage inside the module is used to store three values — secret Q issued by the KDC, session counter σ, and module identity M.

Every module has the following reserved volatile registers:

Self-secret	γ
Static root	ξ_{sp}
Dynamic root	ξ
Peer module params	$\mathbf{p} = \{M', \sigma', K_{in}, K_{out}\}$
Constants	$\mathbf{c} = [C_0 \cdots C_l]$
Sensor report register	$\mathbf{r} = \{\tilde{S}, \tilde{v}, \tilde{t}\}$
Sensor state register	$\mathbf{s}_0 = [S_0, u_0, t_0, o_{0_1} \cdots o_{0_w}, \tau_0]$
Related sensor states	$\mathbf{s}_1 \cdots \mathbf{s}_q$
where	$\mathbf{s}_j = [\hat{S}_j, \hat{u}_j, \hat{t}_j, \hat{o}_{j,1} \cdots \hat{o}_{j_w}, \hat{\tau}_j]$
Temporary register	T
SM Registers	S_{in}, v_{in}

The self-secret γ spontaneously generated inside the module remains privy only to the module. This secret is used for computing self-certificates. A self-certificate is a "memorandum to self," — memoranda issued by the module for verification by itself at a later time, during the same session σ.

The register ξ_{sp} is the (160-bit) descriptor of the STCB system. The register ξ is the dynamic root of a Merkle tree. Register $\mathbf{p}$ contains various parameters regarding a peer module from which a) sensor/state report has to be received, or b) a report has to be sent.

المنارة للاستشارات
www.manaraa.com

The register $\mathbf{c}$ is reserved for storing a set of l constants to be used to evaluate some $\mathcal{U}_i$ corresponding to a sensor S_i. Values that reflect the current state $\mathbf{S}_i$ of S_i should be stored in register $\mathbf{s}_0$. The states of q related sensors are stored in registers $\mathbf{s}_1 \cdots \mathbf{s}_q$. Register $\mathbf{r}$ is reserved for a freshly received sensor report from S_i.

For example, if at some instant of time,

1. the contents of the dynamic record $\mathbf{S}_i$ corresponding to sensor S_i is stored in location $\mathbf{s}_0$, and
2. records corresponding to sensor $S_{i,1} \cdots S_{i,q}$ are stored on location $\mathbf{s}_1 \cdots \mathbf{s}_q$ respectively, and
3. values from a fresh report from sensor S_i are stored in register $\mathbf{r}$, then

$$\begin{aligned}
S_i &= \tilde{S} \\
\hat{S}_j &= S_{i,j} \\
u_0 &= v_i \text{ (measurement of } S_i) \\
\hat{u}_j &= v_{i,j} \text{ (measurements of related sensors of } S_i) \\
\hat{o}_{j_x} &= o_{(i,j)_x}, 1 \leq x \leq w
\end{aligned} \tag{8.13}$$

The SM register S_{in} indicates the sensor to which the module is bound (if the module is used as a SM) or is set to zero (if the module is used as a CM). If the module is used as an SM, the register v_{in} always contains the (dynamic) sensor measurement.

STCB modules have a built in hash function $h()$ which is reused extensively for binary tree ($f_v()$) computations, computing shared secrets, computing message authentication codes, and self-certificates.

المنارة للاستشارات
www.manaraa.com

An in-built function $f_{exec}(\mathbf{A})$ in every module can execute a set of m instructions $\mathbf{A}$, where each instruction (chosen from the set $\mathcal{A}$) identifies a) an opcode (type of logical operation), b) one or more input operands (from the values stored in registers $\mathbf{s}_0$, $\mathbf{s}_1 \cdots \mathbf{s}_q$, $\mathbf{c}$, T), depending on the type of opcode; and c) the output operand ($o_{0_1} \cdots o_{0_w}$, or temporary register T).

8.4.2 Initializing Peer Parameters

The pairwise secret K that a module M shares with a peer module M' is used for computing message authentication codes (MAC) for outgoing messages to peer M', and for verifying incoming MACs from peer M'. Specifically, the secret used by M for computing outgoing MACs is $K_{out} = h(K, \sigma)$ where σ is the session counter of M; consequently, the secret used for *verifying* MACs received from M' is $K_{in} = h(K, \sigma')$, where σ' is the session counter of M'.

STCB modules possess reserved registers to store the identity M' of a peer module (to which it needs to send a message, or from which it needs to receive a message), the session counter σ' of the peer, and MAC secrets K_{in} and K_{out}. Function $F_{peer}()$ exposed by a module can be invoked to populate values $M', \sigma', K_{in}, K_{out}$ related to a peer module M'.

$$
\begin{array}{l}
F_{peer}(I, P, s)\{ \\
\quad M' := I; \sigma' := s; K := h(Q, M') \oplus P; \\
\quad K_{in} := h(K, \sigma'); K_{out} := h(K, \sigma); \\
\}
\end{array}
$$

To facilitate secure communications between two modules M_i and M_j, $F_{peer}(M_j, P_{ij}, \sigma_j)$ should be invoked on M_i, and $F_{peer}(M_i, 0, \sigma_i)$ should be invoked on M_j. M_i computes the

المنارة للاستشارات
www.manaraa.com

pairwise secret using the public value P_{ij}; M_j computes the same value without using P_{ij} (or $P_{ij} = 0$ as XORing with 0 results in no change).

8.4.3 Self Certificates

Two types of certificates are computed by STCB modules — binary hash tree certificates, and offset certificates.

8.4.3.1 Binary Tree Certificates

A binary hash tree certificate is computed as

$$\rho_{bt} = h(x, x', y, y', \gamma). \tag{8.14}$$

Such a self-memoranda states that "x is a node in a binary hash tree with root y," and "if $x \to x'$ then $y \to y'$."

STCB modules expose a function $F_{mt}()$ which evaluates a sequence of hash operations $f_v()$, and output a binary tree certificate.

```
  F_mt(x, x', v_x){
  y := f_v(x, v_x); y' := f_v(x', v_x);
  RETURN ρ_bt := h(x, x', y, y', γ);
}
  F_mtc(x, x', y, y', z, z', ρ1, ρ2){
  IF (ρ1 ≠ h(x, x', y, y', γ)) RETURN ERROR;
  IF (ρ2 ≠ h(y, y', z, z', γ)) RETURN ERROR;
  RETURN ρ_bt := h(x, x', z, z', γ);
}
```

A function $F_{mtc}()$ concatenates two such certificates to create another certificate. Specifically, a certificate binding node x (and x') to an ancestor y (and y') and a certificate binding a node y (and y') to an ancestor z (and z') can be combined to a certificate binding node x (and x') to an ancestor z (and z').

المنارة للاستشارات
www.manaraa.com

The primary need for the function $F_{mtc}()$ is due to restrictions on the size of inputs to module functions. Specifically, we can now place a hard limit on the size of the input $\mathbf{v}_x$ — to (say) 8 hashes. For computing relationships between a node and the root of a tree with a million leaves (20 levels) three calls to $F_{mt}()$ (to produce certificates binding i) a level zero node to a level 8 node, ii) level 8 node to a level 16 node, and iii) a level 16 node to a level 20 node) and two calls $F_{mtc}()$ (to combine the first two certificates, and combine the resulting certificate with the third certificate) can be used.

8.4.3.2 Offset Certificates

An *offset certificate* is computed as

$$\rho_{os} = h(M', \sigma, \sigma', os, \epsilon, \gamma), \tag{8.15}$$

and states that the module M (that issued the certificate) had performed a handshake within a duration ϵ with a module M', and had estimated the offset between their clocks to be os. The certificate also states that the handshake was performed when it's session counter was σ and the session counter of M' was σ'. The offset certificate is issued by a function $F_{hs}()$ exposed by modules.

The function $F_{hs}()$ can be invoked on pair of modules to perform a handshake, after which the initiator of the handshake obtains an estimate of the clock offset of the responder. Before $F_{hs}()$ is invoked, $F_{peer}()$ should be invoked on both nodes to set up respective peer identities, session counters, and secrets K_{in} and K_{out} to be used for incoming and outgoing MACs.

المنارة للاستشارات
www.manaraa.com

The function $F_{hs}()$ has three inputs — a received MAC μ' (from peer M') with time stamp c', and a time-stamp $\tilde{c}$ that was previously sent to peer M' (which had triggered the response μ' from M'). The output of $F_{hs}()$ is either a MAC intended for the peer or a self-MAC intended for itself, indicating the estimated offset for peer M'.

$F_{hs}()$ is first invoked on the initiator with all inputs $(\mu', c', \tilde{c})$ set to zero; the output of $F_{hs}()$ is $\mu_1 = h(c_i^1, 0, \sigma_r, h(K, \sigma_i))$ where c_i^1 and σ_i are the current clock-counter and the session counter of the initiator and σ_r is the session counter of the responder.

$F_{hs}()$ is then invoked in the responder module with inputs $(\mu_1, c_i^1, 0)$. If the clock tick count of the responder is c_r, the output is $\mu_2 = h(c_r, c_i^1, \sigma_i, h(K, \sigma_r))$.

$F_{hs}()$ is then invoked on the initiator for the second time, at time c_i^2, with inputs (μ_2, c_r, c_i^1). The offset between the clock of the initiator and the responder can be estimated by the initiator to within the round-trip duration $\epsilon = c_i^2 - c_i^1$. The best estimate of the initiator is that, when the clock tick count of responder was c_r, the clock tick count of the initiator was $(c_i^1 + c_i^2)/2$, and thus, the best estimate of the offset is $os = (c_i^1 + c_i^2)/2 - c_r$.

The output of $F_{hs}()$ in this case is the offset certificate. This certificate can be provide to the module at any time to convince the module that "the offset to M' was estimated as os with a tolerance of ϵ," and that "the offset to M' was estimated when the session counters of the modules were σ_i and σ_r." If any of the two session counters had changed since the certificate was issued, the certificate becomes invalid.

للاستشارات المنارة

www.manaraa.com

```
F_hs(μ', c̃, c'){
IF (μ' = 0) //Send challenge
   RETURN h(c, 0, σ', K_out); //Sent as challenge
IF (μ' ≠ h(c', c̃, σ, K_in) RETURN ERROR;
IF (c̃ = 0) //Respond to challenge
   RETURN h(c, c', σ', K_out);
ELSE //Process response to estimate offset
   ε := c − c̃; os := (c + c̃)/2 − c';
   RETURN ρ_os := h(M', σ, σ', os, ε, γ);
}
```

8.4.4 Initializing STCB Modules

Initializing an STCB module M implies initializing three internal registers reserved for values ξ_{sp}, ξ and S_{in}. Specifically, a module M can be initialized to participate in a deployment ξ_{sp} only if a binding record for module M can be demonstrated to be consistent with ξ_{sp}.

As ξ_0 is a node in a tree with root ξ_{sp}, **U** can use $F_{mt}()$ to obtain a certificate

$$\rho = h(\xi_0, \xi_0, \xi_{sp}, \xi_{sp}, \gamma). \tag{8.16}$$

Similarly, as binding record $\mathbf{B}_i = [S_i, M, \epsilon, \theta]$ that exists in deployment tree is used to initialize the register S_{in} Now **U** can use $F_{mt}()$ to obtain a certificate

$$\rho = h(v, v, \xi_{sp}, \xi_{sp}, \gamma). \tag{8.17}$$

Function $F_{init}()$ can be used to initialize a module M as

1. a CM for a deployment ξ_{sp} or
2. as an SM for a sensor S in deployment ξ_{sp}.

المنارة للاستشارات
www.manaraa.com

To initialize a module as a CM for the deployment, the inputs (ξ_1, ξ_2, ρ) to $F_{init}()$ are such that ρ is a binary tree certificate relating a node $\xi_0 = \xi_1$ and root $\xi_2 = \xi_{sp}$ (inputs S' and ϵ' are set to 0).

To initialize the module as a SM the certificate, inputs S' and ϵ are non-zero. The binary tree certificate should relate $\xi_1 = h(S', M, \epsilon', 0)$ and $\xi_2 = \xi_{sp}$ to prove to the module that "in an STCB system with descriptor ξ_{sp}, the module M (which is being initialized) is authorized to report measurements corresponding to sensor S'." Accordingly, the register S_{in} in the module M is set to S'.

For a module M_j initialized as a SM for a sensor S_k, the output of the sensor S_k is physically connected to module M_j using a tamper-evident seal by the deployer. The physical connection ensures that the sensor measurement v_k is always available in the register v_{in} of the SM. measurement Later (during regular operation) module M_j can not be initialized to act as a SM for any other sensor $S' \neq S_k$, as no record binding S' to M_j can be demonstrated to be a part of the tree with root ξ_{sp}.

```
F_init(ξ0, r, S, ε, ρ1, ρ2){
  IF (ρ1 ≠ h(ξ0, ξ0, r, r, γ)) RETURN ERROR;
  x := h(S, M, ε, 0);
  IF (ρ2 ≠ h(x, x, r, r, γ)) RETURN ERROR;
  ξsp = r; ξ := ξ0; Sin = S'; RETURN;
}
```

During regular operation, any dynamic sensor record can be loaded on to any register s_0 or $\mathrm{s}_1 \cdots \mathrm{s}_q$ using function $F_{ld}()$. A record s provided as input is simply loaded onto register s_j where j is the index specified. Specifically, the record is loaded only if the inputs ρ and $h(\mathrm{s})$ are consistent with dynamic root ξ.

المنارة للاستشارات
www.manaraa.com

8.4.5 Sensor and State Reports

In an STCB deployment with SMs $M_1 \cdots M_k$, and STCB module M_0, the state reports are made available to a module M_r associated with the stake holder. In general, M_r can be seen as an STCB module associated with a different STCB system at a higher lever of hierarchy.

The handshake sequence (which involves two calls to $F_{hs}()$ in the initiator module and one $F_{hs}()$ call in the responder module) the handshake sequence is orchestrated by **U** between

1. k responders $M_1 \cdots M_k$, with M_0 (as initiator)
2. M_r as initiator and M_0 as responder.

After the $k + 1$ hand-shake sequences have been completed, k self-certificates of type OS ρ_{os} are created by M_0 — one corresponding to each SM, and one self-certificate is created by M_r. Such certificates indicate both the estimated offset os, and the maximum error ϵ in the estimate os.

Now modules are ready to exchange authenticated messages. More specifically, SMs send authenticated and time-stamped sensor reports to M_0, and STCB module M_0 can send state reports to M_r. Such messages exchanged between modules are computed as

$$\mu = h(S, v, t, c, \xi_{sp}, K_{out}). \tag{8.18}$$

where for sensor reports (from SMs to M_0)

1. $S = S_{in}$ is the identity of a sensor that is bound to the module that created the report, and $v = v_{in}$;
2. $t = c$ is the current clock tick count of the creator of the report.

المنارة للاستشارات
www.manaraa.com

3. ξ_{sp} is the value used to initialize the module.

A report from M_0 to stake holders is made in accordance with a record $[S_r, M_r, S_i, 1 \leq l \leq w]$. Such a report can be created for the benefit of M_r only if the state of sensor S_i is loaded in register s_0 (or $S_0 = S_i$), to report the value e_0^l and time τ_0. In such a report $S = S_r$ is a label assigned to the report, $v = e_0^l$, and $t = \tau_t$ is the time associated with $v = e_0^l$ (or $t \neq c$).

$F_{snd}()$ outputs $\mu = h(S_{in}, v_{in}, c, c, \xi_{sp}, K_{out})$ when invoked in a SM (with register $S_{in} \neq 0$). When invoked on an STCB module, this function should be able to verify the existence of an appropriate reporting record **R** that authorizes the module to report one of the w values $e_o^1 \cdots e_o^w$ stored in register s_0.

```
F_snd(S', j, ρ){
IF (S ≠ 0)
   RETURN c, μ = h(S_in, v_in, c, c, ξ_sp, K_out);
y = h(S', M', S_0, j);
IF (ρ = h(y, y, ξ_sp, ξ_sp, γ)
   RETURN c, μ = h(S', o0_j, τ_0, c, ξ_sp, K_out);
ELSE RETURN ERROR;
}
```

Corresponding to a state report for a "sensor" S_r, while the STCB module M_0 that generates the report is initialized with the ξ_{sp}, the stake-holder module M_r — which in general can be seen as the STCB module associated with a foreign STCB system may be initialized with a different descriptor ξ_{sp}'. For the module M_r to accept the report from a foreign system, the deployment tree in the foreign system with root ξ_{sp}' should include a binding record

$$\mathbf{B} = [S_r, M_0, \epsilon, \theta = \xi_{sp} \neq 0]. \tag{8.19}$$

www.manaraa.com

Function $F_{rcv}()$ can be invoked in the STCB module to provide a fresh sensor report to the module. Specifically, before a sensor report from a module M_j can be provided to a module, the function $F_{peer}()$ should be invoked on the receiving module to set $M' = M_j$. Now,

1. inputs S_i, ϵ', θ to $F_{rcv}()$ are used to compute the leaf hash $x = h(S_i, M', \epsilon', \theta)$ of a binding record $\mathbf{B}_i$.
2. input ρ is used to confirm that $x = h(S_i, M', \epsilon', \theta)$ (the hash of the binding record) is a node in a tree with root ξ_{sp}.
3. inputs ϵ, os and ρ_{os} are used to verify that $\rho_{os} = h(M', \sigma, \sigma', os, \epsilon, \gamma)$.
4. and inputs v_i', t', c' and μ' are used to verify that $\mu' = h(S_i, v_i', t', c', y, K_{in})$ where $y = \xi_{sp}$ if $\theta = 0$, or $y = \theta$ if $\theta \neq 0$.

The function returns error if $\epsilon > \epsilon'$, or on failure of verification of inputs ρ_{nv} or ρ_{os} or μ'.

Ultimately, the purpose of function $F_{rcv}()$ is to receive two values v_i' and time t_i' corresponding to a sensor S_i where $t_i' = t + os$ is the offset corrected time associated with v_i'. Values S_i, v_i' and t_i' are then stored in a reserved register $\mathbf{R}$ for further processing.

```
F_rcv(S_i, θ, ε', ρ_bt, os, ε, ρ_os, v'_i, ρ, t', c', μ', ){
IF (ε > ε') RETURN ERROR;
IF (ρ_os ≠ h(M', σ, σ', os, ε, γ)) RETURN ERROR;
y = (θ = 0)?ξ_sp : θ;
x = h(S_i, M', ε', y);
IF (ρ_bt ≠ h(x, x, ξ_sp, ξ_sp, γ)) RETURN ERROR;
IF (μ' ≠ h(S_i, v'_i, t', c', y, K_in) RETURN ERROR;
r := (S̃ = S_i, ṽ = v'_i, t̃ = t' + os);
}
```

8.4.6 Sensor Updates and Incremental State Evaluations

Values $\mathbf{r}$ in a fresh sensor report from S_i are part of the inputs used to evaluate $\mathcal{U}_i$. Evaluation of $\mathcal{U}_i$ results in modifications to the state $\mathbf{S}_i$ of sensor S_i. Before $\mathcal{U}_i$ can be

المنارة للاستشارات
www.manaraa.com

evaluated it should be ensured that appropriate values are loaded on to registers $\mathbf{r}$, s_0, $\mathrm{s}_0 \cdots \mathrm{s}_q$, and $\mathbf{c}$.

Recall that register $\mathbf{r}$ is populated by function $F_{rcv}()$. Function $F_{ld}()$ can be used to load the dynamic values $\mathbf{S}$ associated with any sensor on to any of the $q+1$ registers s_0, $\mathbf{s}_0 \cdots \mathbf{s}_q$.

$F_{ld}(\mathbf{s}', j, \rho)\{$
$x := h(\mathbf{s}')$;
IF $(\rho \neq h(x, x, \xi, \xi, \gamma))$ RETURN ERROR;
//Record **is** a leaf in the dynamic tree
$\mathbf{s}_j := \mathbf{S}$;
$\}$
$F_{upd}(\mathbf{C}, \lambda, \rho_c, \mathbf{A}, \rho, \xi', \rho_{upd}, S_i')\{$
IF $(S_0 \neq \tilde{S})$ RETURN ERROR;
$tmp = h([S_0, R_1 \cdots R_q, h(\mathbf{A}), \lambda, S_i'])$; // hash of a design record
$//tmp$ should be a node in the static tree with root ξ_{sp}
IF $(\rho \neq h(tmp, tmp, \xi_{sp}, \xi_{sp}, \gamma))$ RETURN ERROR;
$tmp = h(\mathbf{C})$; // hash of constant record
$//$ should be a node in the constant tree with root λ
IF $(\rho_c \neq h(tmp, tmp, \lambda, \lambda, \gamma))$ RETURN ERROR;
$tmp := h(\mathbf{s}_0)$; // Record before evaluation of $\mathcal{U}$
$\tau_0 := \min_0(\tilde{t}, \tau_1' \cdots \tau_q')$;
$T := \tilde{v}; v_0 := \tilde{v}; t_0 := \tilde{t}$;
IF $((f_{\mathbf{A}}() = ERROR) \vee (\rho_{upd} \neq h(tmp, h(\mathbf{s}_0), \xi, \xi', \gamma)))$
CLEAR-ALL AND RETURN;
$\xi := \xi'$;
IF $(S_i' \neq 0)$ $\tilde{S} := S_i'$;
$\}$

The main purpose of $F_{upd}()$ is to evaluate $\mathbf{U}_i$ corresponding to a sensor S_i. For this purpose, $F_{upd}()$ verifies that all inputs required to evaluate $\mathbf{U}_i$ are available. If a design record corresponding to sensor S_i is

$$\mathbf{G}_i = [S_i, S_{i_1} \cdots S_{i_q}, \alpha_i, \lambda_i, S_i'] \tag{8.20}$$

www.manaraa.com

it should be ensured that the current state of sensor S_i is loaded onto register s_0, and the states of related sensors $S_{i_1} \cdots S_{i_q}$ are loaded on to registers $\mathrm{s}_1 \cdots \mathrm{s}_q$ respectively. Other values required to evaluate $\mathcal{U}_i$ are provided as inputs to $F_{upd}()$.

Specifically,

1. the constants $\mathbf{C}$ should be such that $x = h(\mathbf{C})$ is node in a tree with root λ_i. This can be demonstrated by providing a certificate $\rho_c = h(x, x, \lambda_i, \lambda_i, \gamma)$.
2. the instructions $\mathbf{A}$ should be such that $h(\mathbf{A}) = \alpha_i$.
3. λ_i and α_i should exist in the design record $[S_0, S_{i_1} \cdots S_{i_q}, \alpha_i, \lambda_i]$. More specifically, $y = h(S_0, S_{i_1} \cdots S_{i_q}, \alpha_i, \lambda_i, N_i)$ should be a node in the tree with root ξ_{sp}. This can be demonstrated by providing a certificate $\rho = h(y, y, \xi_{sp}, \xi_{sp}, \gamma)$.
4. the values $S_0 \in \mathbf{s}$ (the identity of the sensor to be updated) and $\tilde{S} \in \mathbf{r}$ (the sensor from which a fresh report has been received) should be the same.

During execution of the algorithm $\mathbf{A}$, in situations where many options exist for choosing the set of constants $\mathbf{C}$ consistent with λ it is the responsibility of $\mathbf{U}$ to choose the correct set of constants that satisfy the range of the independent variable(s). On successful evaluation of the algorithm $\mathbf{A}$ the status of sensor S_0 in register s_0 will be updated. If x is the hash of register s_0 *before* the update, and if x' is the hash of register s_0 *after* the update, then a new root ξ' and a certificate ρ should be provided as input satisfying

$$\rho = h(x, x', \xi, \xi', \gamma). \tag{8.21}$$

8.5 STCB Protocol

The STCB protocol can be seen as the actions to be performed by the untrusted STCB manager $\mathbf{U}$ to submit sensor reports from SMs and CMs of foreign deployments to the CM M_0 of the STCB deployment, obtain state-reports from M_0, and submit such reports to stake holders (or CMs of foreign deployments).

المنارة للاستشارات
www.manaraa.com

8.5.1 Generation of Offset Certificates

The first step in the operation of an STCB deployment is that of performing handshakes between various modules to obtain offset certificates. In general one offset certificate will be generated for every module specified in the binding records of the deployment.

For a STCB system with n sensors (real, state-report and synthetic) the total number of binding records is $n' + 1$, where $n - n'$ is the number of synthetic records: no binding record exists for synthetic records, and one binding record is for the CM M_0. The total number of distinct modules in general will be $n'' \leq n'$. Specifically, while there will exist one module corresponding to every real sensor, as a single CM may report multiple states, the number of state-report sensors may be greater that the number of foreign CMs that provide state reports.

A total of n'' hand shake sequences will be invoked to obtain n'' offset certificates. Recall that each such sequence begins with a challenge from the CM M_0 generated using $F_{hs}()$ to which a response is generated by invoking $F_{hs}()$ in the responder module, and finally, the response is submitted to the CM to generate the certificate. If any SM is rebooted, the offset certificate corresponding to the SM has to be regenerated. If the CM is rebooted, all offset certificates will need to be regenerated.

Before $F_{hs}()$ is invoked, $F_{peer}()$ should be invoked on both modules to set up the MAC secrets K_{in} and K_{out}.

المنارة للاستشارات
www.manaraa.com

8.5.2 Generating Static Binary Tree Certificates

The second step is for **U** to obtain binary tree certificates corresponding to all leaves of the static tree with root ξ_{sp}. Specifically, as **U** maintains the tree with root ξ_{sp}, **U** can readily provide the complementary nodes for any leaf in the static tree to function $F_{bt}()$, and obtain certificates of the form

$$\rho_s = h(x_s, x_s, \xi_{sp}, \xi_{sp}, \gamma) \tag{8.22}$$

where x_s is the cryptographic hash of the s^{th}leaf of the static tree. The total number of static leaves is $(n + 1) + (n' + 1) + m$ where

1. n is the number of design records: one for each sensor (real, state-report, or synthetic)
2. one leaf corresponds to the value ξ_0 in the design tree.
3. $n' + 1$ is the number of binding records (including one for the CM M_0), and
4. m is the number of reporting records.

8.5.3 Initialization and Regular Operation

The third step is the initialization of STCB modules to operate in deployment ξ_{sp} — by invoking $F_{init}()$ on each STCB module. For initializing the modules the two binary tree certificates are required: one linking ξ_0 to ξ_{sp}, and one linking a binding record for the module with the static root ξ_{sp}.

On completion of the three steps, the STCB manager maintains a dynamic merkle tree with leaves as sensor records. As the initial values of such records are specified by the designer, the root of the tree should be the same as the initial value $\xi = \xi_0$ stored by the STCB module.

المنارة للاستشارات
www.manaraa.com

Once all STCB modules have been initialized, sensor reports from SMs (or CMs of foreign deployments) are submitted to the CM as and when they are received. In general, not all sensors may report at the same frequency.

As all SMs send messages only to the CM, $F_{peer}()$ needs to be invoked only once on each SM (which was already performed before invoking $F_{hs}()$ to generate offset certificates).

To create a sensor report from an SM, $\mathbf{U}$ will invoke $F_{snd}()$ on an SM. Some sensor reports corresponding to state reports from foreign CMs may also be received by $\mathbf{U}$.

Once a sensor report for some sensor S_i has been received, $\mathbf{U}$ is expected to make appropriate modifications to the sensor state $\mathbf{S}_i$, and accordingly, modify the Merkle tree maintained by $\mathbf{U}$. Let $\xi \rightarrow \xi'$ be the change in the root of the dynamic tree, corresponding to the modification $\mathbf{S}_i \rightarrow \mathbf{S}'_i$ triggered by the received sensor report. If $h(\mathbf{S}_i) = x$ and $h(\mathbf{S}'_i) = x'$, $\mathbf{U}$ can readily determine complementary nodes required to obtain the certificate

$$\rho_{bt} = h(x, x', \xi, \xi', \gamma) \tag{8.23}$$

The STCB manager invokes $F_{peer}()$ followed by $F_{rcv}()$ to submit the report to the CM. Recall that inputs to $F_{rcv}()$ include a MAC received from an SM/CM, a binding record along with a certificate linking the record to ξ_{sp}, and an offset certificate corresponding to the reporting module.

Corresponding to the sensor state record for sensor S_i and q related sensors $\mathbf{U}$ invokes $F_{bt}()$ to obtain $q + 1$ certificates binding the sensor state records to dynamic root ξ. Fol-

المنارة للاستشارات
www.manaraa.com

lowing this, the STCB manager uses $F_{ld}()$ up to $q+1$ times to load the a) previous sensor state $\mathbf{S}_i$ on to register s_0 and b) the states of related sensors on to registers $\mathrm{s}_1 \cdots \mathrm{s}_q$.

Finally, the STCB manager invokes $F_{upd}()$. Recall that the inputs to $F_{upd}()$ include ρ_{bt} obtained as per Eq (8.23), a certificate binding design record $\mathbf{G}_i$ to STCB descriptor ξ_{sp}, the set of m instructions $\mathbf{A}_i$, a set of l constants, and a certificate binding the constants to a value λ in the design record. Only if the modification $\mathbf{S}_i \rightarrow \mathbf{S}'_i$ computed by the CM is exactly the same as that performed by the STCB manager $\mathbf{U}$ will the update be successful in modifying the dynamic root ξ stored inside the CM to ξ'.

At any time the STCB manager can invoke a $F_{upd}()$ to load a state record consistent with ξ on to register s_0. Now $F_{snd}()$ can be invoked to create a state-report. Note that when $F_{snd}()$ is invoked on a CM a certificate binding a reporting record to the static root should be provided as input.

The ever growing complexity of systems poses a severe threat — the possibility of hard-to-detect hidden functionality that can be exploited to take control of the system. Current strategies for securing SCADA systems are predominantly focused on development of suitable intrusion detection systems. Such security measures ignore the very real possibility of hidden functionality in the intrusion detection systems themselves.

In the proposed approach to secure SCADA systems only STCB modules are trusted to provide the assurance that "no attack will go undetected." The proposed approach involves three stages — a design process carried out by a designer with good domain knowledge, a deployment process carried out by a security professional, and regular operation of the STCB system. The designer and deployer together specify a concise description ξ_{sp} of the

المنارة للاستشارات
www.manaraa.com

system. During regular operation, an STCB module reports the state of a system identified as ξ_{sp} to stake-holders.

Some of the important features of the STCB approach that make it well suited for any SCADA system of any size include

1. the ability to support hierarchical deployments;
2. the ability to support any type of function $\mathcal{U}$- if necessary through the use of 1-D, or 2-D look up tables (which are also specified as leaves of the design tree), and
3. the ability to specify synthetic sensors.

Such features are intended to enable the use of STCB modules for securing any SCADA system.

The first pre-requisite for deployment of STCB based security solutions is the actual availability of STCB modules/chips. Towards this end, the work that has been performed is a small first step — identification of a functional specification of the such chips. In arriving at an appropriate functional specification, some our main goals have been

1. reduce computational and memory requirement inside STCB chips,
2. reduce interface complexity (size of inputs and outputs to/from the STCB chips), and
3. simplify the STCB protocol - which is a specification of a sequence of interactions with the STCB modules - to realize the desired assurances.

The proposed functional specification (for STCB modules) is merely *a* specification, and not *the* specification. Just as there are numerous ways to realize a block-cipher or a hash function, there are numerous ways to arrive at a "set of STCB functions" (which can be leveraged to realize the same assurances). The functional specification in this chapter is however the first of its kind.

www.manaraa.com

8.5.4 STCB Design Example

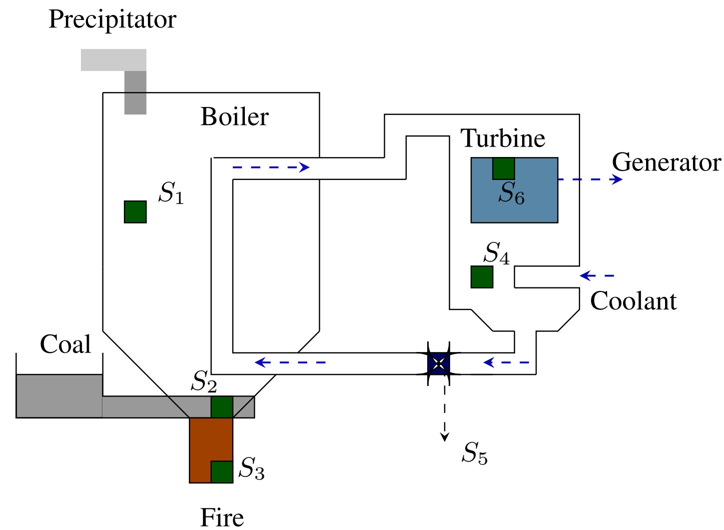

Figure 8.4

Example: Simplified version of thermal power plant

As an example of the design process for a SCADA system, let us consider a simplified version of a thermal power plant with six sensors $S_1 \cdots S_6$, (see Figure **??**).

S_1 temperature sensor inside boiler

S_2 coal weight sensor (coal fed into the boiler).

S_3 position of fire regulator.

S_4 temperature inside turbine cell.

S_5 pressure inside turbine cell.

S_6 speed of turbine.

www.manaraa.com

Let $v_1 \cdots v_6$ be the values reported by sensors $S_1 \cdots S_6$ required to determine the state of the system.

Let us assume that the state report expected by the stake-holder is a single bit value — $o_1 = 1$ if the system is in an acceptable state, and $o_1 = 0$ otherwise. According to the designer, the system is in an acceptable state if the following conditions are satisfied:

1. $v_1 \cdots v_6$ are all within thresholds $(x_l^1, x_h^1) \cdots (x_l^6, x_h^6)$ respectively, where (x_l^i, x_h^i) represents lower and higher thresholds for sensor S_i.
2. The speed of turbine should be between upper and lower limits depending on the temperature and pressure inside the turbine cell. $v_6 = f_1(v_4, v_5) \pm \delta_1$ where δ_1 is another threshold (the speed of the turbine should be a specific function of the pressure and temperature inside the turbine cell)
3. The position of the fire regulator should be between upper and lower limits depending on the current speed of turbine v_6 and the current temperature and pressure values inside the boiler cell. $v_3 = f_2(y_1, v_6) \pm \delta_2$ where $y_1 = f_3(v_1, v_2)$ is a function of the temperature and pressure of the boiler.

Let the maximum number of related sensors be $q = 3$; the number of outputs of each $\mathcal{U}$ be $w = 2$; and the number of constants $l = 8$. A possible design of functions $\mathcal{U}_i, 1 \leq i \leq 6$ is as follows:

1) $\mathcal{U}_1$ checks if v_1, v_2 are within thresholds $(x_l^1, x_h^1), (x_l^2, x_h^2)$ respectively. The inputs are v_1, v_2 and constants. The output is written in o_{0_1}. For evaluation of $\mathcal{U}_1$, S_2 is specified as a related sensor. As no other related sensors are used, S_{1_2} and S_{1_3} are set to 0;

2) $\mathcal{U}_2$ performs LUT2 operation for function $f_3()$. The inputs are v_1, v_2 and an LUT leaf. The output is written o_{0_2}. For chaining the output of $\mathcal{U}_1$ (now stored in register s_1) to entire system state, the current value at output register o_{1_1} of S_1 is copied to S_2's output register o_{0_1}.

المنارة للاستشارات
www.manaraa.com

3) $\mathcal{U}_3$ regards S_2 and S_6 as related sensors, and perform LUT2 operation $f_2()$ on o_{1_2} (an output of of related sensor S_2) and value $u_2 = v_6$ of other related sensor S_6 (now stored in register s_2. The contents of output register o_{1_1} of S_2 are copied to o_{0_1} of S_3.

4) $\mathcal{U}_4$ checks if v_4, v_5 are within thresholds $(x_l^4, x_h^4), (x_l^5, x_h^5)$ respectively. The inputs are v_4, v_5 and constants. The output is provided in register o_{0_1}. S_5 is specified as a related sensor of S_4.

5) $\mathcal{U}_5$ regards S_4 as related sensor and perform a LUT2 operation $f_1()$ on value v_4 of related sensor S_4 (value u in record s_1) and v_5 (u in register s_0), the output is stored in o_{0_2}. The content register o_{1_1} (of S_4) are copied to o_{0_1} (of S_5)

6) $\mathcal{U}_6$ regards S_3 and S_5 as related sensors. This function checks if v_3, v_6 are within thresholds $(x_l^3, x_h^3), (x_l^6, x_h^6)$ respectively and the output is stored at register o_{0_1} of S_6. In addition, $\mathcal{U}_6$ also performs the following steps

1. if v_3 available at u_1 satisfies $f_2 \pm \delta_2$ — the result of f_2 is now available at o_{1_2} of related sensor S_3; the result of the check is stored at o_{0_2} of S_6. The result of an AND operation performed on outputs in o_{0_1} and o_{0_2} is stored back in o_{0_1}.
2. if v_5 available at u_2 satisfies $f_1 \pm \delta_1$ — the output of f_1 is available at o_{2_2} of related sensor S_5; the 1/0 result is stored in register o_{0_2}.

A result of AND operation of output registers of S_6 – 0_{0_1}, 0_{0_2} placed in 0_{0_1} of S_6 represents the entire state of system (acceptable -1 , unacceptable - 0).

The constants required to evaluate $\mathcal{U}_i$ are provided as a leaf that can be proved against corresponding root λ. The designer specifies the following constant trees:

1. A tree with one leaf with 8 constants $[x_l^1, x_h^1, x_l^2, x_h^2, x_l^4, x_h^4, x_l^5, x_h^5]$ with root λ_a.
2. A tree with one leaf with 8 constants $[\delta_1, \delta_2, x_l^6, x_h^6, x_l^3, x_h^3, 0, 0]$ with root λ_b.

www.manaraa.com

3. Three trees — one for a 2D LUT for function $f_1()$ with root λ_c; one for a 2D LUT for function $f_2()$ with root λ_d; and the third for 2D LUT for function $f_3()$ with root λ_e;

With available information from $\mathcal{U}_1 \cdots \mathcal{U}_6$ the designer specifies the following design records:

$$
\begin{aligned}
\mathbf{G}_1 &= [S_1, \{S_2, 0, 0\}, \alpha_1, \lambda_a, 0] \\
\mathbf{G}_2 &= [S_2, \{S_1, 0, 0\}, \alpha_2, \lambda_e, 0] \\
\mathbf{G}_3 &= [S_3, \{S_2, S_6, 0\}, \alpha_3, \lambda_d, 0] \\
\mathbf{G}_4 &= [S_4, \{S_5, 0, 0\}, \alpha_4, \lambda_a, 0] \\
\mathbf{G}_5 &= [S_5, \{S_4, 0, 0\}, \alpha_5, \lambda_c, 0] \\
\mathbf{G}_6 &= [S_6, \{S_3, S_5, 0\}, \alpha_6, \lambda_b, 0]
\end{aligned} \tag{8.24}
$$

where $\alpha_1 \cdots \alpha_2$ are hashes of instructions outlined in Table 8.1.

المنارة للاستشارات
www.manaraa.com

Table 8.1

Instruction Set for Thermal Plant

Algorithm	OPCODE	Input registers	Output registers
	CHKB	$\mathbf{c}:C_1, \mathbf{s_0}:u_0$	T
α_1	CHKB	$\mathbf{c}:C_3, \mathbf{s_1}:u_1$	$\mathbf{s_0}:o_{0_2}$
	AND	$T, \mathbf{s_0}:o_{0_2}$	$\mathbf{s_0}:0_{0_1}$
α_2	LUT2	$\mathbf{s_1}:u_1, \mathbf{s_0}:u_0$	$\mathbf{s_0}:o_{0_2}$
	COPY	$\mathbf{s_1}:o_{1_1}$	$\mathbf{s_0}:o_{0_1}$
α_3	COPY	$\mathbf{s_1}:o_{1_1}$	$\mathbf{s_0}:o_{0_1}$
	LUT2	$\mathbf{s_1}:o_{1_2}, \mathbf{s_2}:u_2$	$\mathbf{s_0}:o_{0_2}$
	CHKB	$\mathbf{c}:C_5, \mathbf{s_0}:u_0$	T
α_4	CHKB	$\mathbf{c}:C_7, \mathbf{s_1}:u_1$	$\mathbf{s_0}:o_{0_2}$
	AND	$T, \mathbf{s_0}:o_{0_2}$	$\mathbf{s_0}:o_{0_1}$
α_5	LUT2	$\mathbf{s_1}:u_1, \mathbf{s_0}:u_0$	$\mathbf{s_0}:o_{0_2}$
	COPY	$\mathbf{s_1}:o_{1_1}$	$\mathbf{s_0}:o_{0_1}$
	CHKB	$\mathbf{c}:C_3, \mathbf{s_0}:u_0$	T
	CHKB	$\mathbf{c}:C_5, \mathbf{s_1}:u_1$	$\mathbf{s_0}:o_{0_2}$
	AND	$T, \mathbf{s_0}:o_{0_2}$	$\mathbf{s_0}:o_{0_1}$
	MOV	$\mathbf{c}:C_2$	T
α_6	TOL	$\mathbf{s_1}:o_{1_2}, \mathbf{s_1}:u_1$	$\mathbf{s_0}:o_{0_2}$
	AND	$\mathbf{s_0}:o_{0_1}, \mathbf{s_0}:o_{0_2}$	$\mathbf{s_0}:0_{0_1}$
	MOV	$\mathbf{c}:C_1$	T
	TOL	$\mathbf{s_2}:o_{2_2}, \mathbf{s_2}:u_2$	$\mathbf{s_0}:o_{0_2}$
	AND	$\mathbf{s_0}:o_{0_1}, \mathbf{s_0}:o_{0_2}$	$\mathbf{s_0}:0_{0_1}$

المنارة للاستشارات
www.manaraa.com

CHAPTER 9

CONCLUSIONS AND FUTURE RESEARCH

Accompanying the growing complexity of systems is a severe security threat — that hard-to-detect hidden functionality in system components could be exploited to take control of the system. Current strategies for securing complex systems are predominantly focused on development of suitable intrusion detection systems, often ignoring the very real possibility of hidden functionality in the intrusion detection systems themselves. The threat of hidden functionality is especially severe for critical infrastructure systems due to the increased likelihood of sophisticated state sponsored attacks.

The broad contribution of this dissertation is an alternate approach to secure systems by clearly identifying a minimal set of components that need to be trusted, and ensuring that such components are indeed worthy of trust.

9.1 Contributions

The specific contributions of this dissertation are architectures for securing two major critical infrastructure domains: data dissemination systems (DNS, dynamic look up service) and SCADA systems. For each domain, the proposed security architecture identified

1. a specification of TCB functionality, to be executed inside a trustworthy boundary
2. a security protocol, which is a specification of the nature of interactions between untrusted components and the TCB modules.

www.manaraa.com

For DNS we identified the TCB function to be a simple atomic relay function. For dynamic DNS the TCB functionality supported enabled establishment of a shared secret between any user and a module associated with a look-up server. In addition, the TCB functionality enabled the module to maintain and index ordered Merkle tree. For SCADA systems the TCB functionality included the ability to maintain merkle hash trees, and the ability to support a simple instruction set.

The following publications resulted from this research:

1. Trustworthy TCB for DNS Servers, International Journal of Network Security, Vol.14, No.3, PP. 187-205, May 2012.
2. Minimizing the TCB for Securing SCADA Systems, The 7 annual CSIIR workshop 2011, ACM ICPS.
3. An Efficient Trusted Computing Base (TCB) for a SCADA System Monitor, The 10th International Information and Telecommunication Technologies Conference, I2TS 2011, Floripa, Brazil.
4. An Efficient TCB for a Generic Data Dissemination System, International Conference on Communications in China: Communications Theory and Security (CTS), ICCC12-CTS, China.
5. An Efficient TCB for a Generic Content Distribution System, International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery (CyberC) 2012.
6. A Security Architecture for SCADA systems, submitted to the IEEE transactions on Cybernetics (2013 March).

9.2 Future Research

The first pre-requisite for deployment of TCB based security solutions is the actual availability of TCB modules/chips. Towards this end, the work that has been performed is a small first step — identification of a functional specification of such chips. In arriving at an appropriate functional specification, some of our main goals have been to

المنارة للاستشارات
www.manaraa.com

1. reduce computational and memory requirement inside TCB chips,
2. reduce interface complexity (size of inputs and outputs to/from TCB module), and
3. simplify the TCB protocol — which is a specification of a sequence of interactions with the TCB modules — to realize the desired assurances.

The proposed functional specification is merely *a* specification, and not *the* specification. Just as there are numerous ways to realize a block-cipher or a hash function, there are numerous ways to arrive at a "set of TCB functions" (which can be leveraged to realize the same set of desired assurances). The functional specifications in this dissertation — for DNS modules and STCB modules — are however, the first of their kind, and should therefore be evaluated against similar competing efforts in the future.

While the basis for choosing one of several competing hash functions (that are equally strong from a security perspective) — for example, factors like complexity of hardware and software implementation, speed, delay, etc., are well appreciated, a quantitative basis for evaluating two competing sets of functional specifications needs further investigation. As any modification to the DNS module / STCB functionality will affect the corresponding security protocol (which specifies *how* such functionality is utilized), any new functional specification should be accompanied by a corresponding protocol specification. Thus, evaluation of competing efforts should also consider trade-offs between TCB complexity and overhead for the security protocol.

المنارة للاستشارات
www.manaraa.com

REFERENCES

[1] "21 Steps to improve security of SCADA systems," 301/903-3777, The Presidents Critical Infrastructure Protection Board, Office of Energy Assurance, U.S. Department of Energy.

[2] *IEEE P711 Standard for a Cryptographic Protocol for Cyber Security of Substation Serial Links*, Tech. Rep., https://www.digitalbond.com/scadapedia/standards/ieee-p711.

[3] "Overview of Cyber Vulnerabilities," US-CERT: Control Systems Security Program.

[4] "Trusted Computing Group," http://www.trustedcomputinggroup.org.

[5] *Dns Survey*, Tech. Rep., 2006, http://dns.measurement-factory.com/surveys/200608.html.

[6] "ICS-CERT Incident Response Summary Report," 2012.

[7] "Exposing One of Chinas Cyber Espionage Units," Feb. 2013, http://intelreport.mandiant.com/Mandiant_APT1_Report.pdf.

[8] A. Anagnostopoulos, M. Goodrich, and R. Tamassia, "Persistent authenticated dictionaries and their applications," *Information Security*, 2001, pp. 379–393.

[9] R. Arends, R. Austein, M. Larson, D. Massey, and S. Rose, *DNS security introduction and requirements*, Tech. Rep., RFC 4033, March, 2005.

[10] J. Bau and J. C. Mitchell, "A security evaluation of DNSSEC with NSEC3," *Proceedings of the 17th Annual Network and Distributed System Security Symposium (NDSS)*, 2010.

[11] J. Benoit, *An Introduction to Cryptography as Applied to the Smart Grid*, Cooper Power Systems, 2011.

[12] D. J. Bernstein, "DNSCurve: Usable security for DNS," *Posted at: http://dnscurve.org*, 2010.

المنارة للاستشارات
www.manaraa.com

[13] R. Berthier, W. H. Sanders, and H. Khurana, "Intrusion detection for advanced metering infrastructures: Requirements and architectural directions," *Smart Grid Communications (SmartGridComm), 2010 First IEEE International Conference on*. IEEE, 2010, pp. 350–355.

[14] S. A. Boyer, *SCADA: supervisory control and data acquisition*, International Society of Automation, 2009.

[15] S. Bratus, N. DCunha, E. Sparks, and S. Smith, "TOCTOU, traps, and trusted computing," *Trusted Computing-Challenges and Applications*, 2008, pp. 14–32.

[16] A. Buldas, P. Laud, and H. Lipmaa, "Accountable certificate management using undeniable attestations," *Proceedings of the 7th ACM conference on Computer and communications security*. ACM, 2000, pp. 9–17.

[17] A. R. Chavez, "Position Paper: Protecting Process Control Systems against Lifecycle Attacks Using Trust Anchors," .

[18] G. M. Coates, K. M. Hopkinson, S. R. Graham, and S. H. Kurkowski, "A Trust System Architecture for SCADA Network Securiy," *IEEE Transactions on Power Delivery*, January 2010.

[19] CommunityDNS, "DNSSEC A Way Forward for TLD Registries," 2009.

[20] P. Devanbu, M. Gertz, A. Kwong, C. Martel, G. Nuckolls, and S. G. Stubblebine, "Flexible authentication of XML documents," *Proceedings of the 8th ACM conference on Computer and Communications Security*. ACM, 2001, pp. 136–145.

[21] P. Devanbu, M. Gertz, C. Martel, and S. Stubblebine, "Authentic third-party data publication," *Data and Application Security*, 2002, pp. 101–112.

[22] C. Fetzer and T. Jim, *Incentives and Disincentives for DNSSEC Deployment*, trevor/papers/dnssec-incentives.pdf, 2004.

[23] D. P. Fidler, "Was Stuxnet an act of war? Decoding a cyberattack," *Security & Privacy, IEEE*, vol. 9.4, 2011, pp. 56–59.

[24] R. Gennaro, A. Lysyanskaya, T. Malkin, S. Micali, and T. Rabin, *Tamper Proof Security: Theoretical Foundations for Security Against Hardware Tampering*, Theory of Cryptography Conference, Cambridge, MA, 2004.

[25] M. Goodrich, R. Tamassia, N. Triandopoulos, and R. Cohen, "Authenticated data structures for graph and geometric searching," *Topics in CryptologyCT-RSA 2003*, 2003, pp. 295–313.

المنارة للاستشارات
www.manaraa.com

[26] M. T. Goodrich, R. Tamassia, and A. Schwerin, "Implementation of an authenticated dictionary with skip lists and commutative hashing," *DARPA Information Survivability Conference & Exposition II, 2001. DISCEX'01. Proceedings*. IEEE, 2001, vol. 2, pp. 68–82.

[27] T. Goodspeed, S. Bratus, R. Melgares, R. Speers, and S. W. Smith, "Api-do: Tools for Exploring the Wireless Attack Surface in Smart Meters," *The 45th IEEE Hawaii International Conference on System Sciences*, 2012, pp. 2133–2140.

[28] S. Gorman, "Chinese hackers suspected in long-term Nortel breach," *The Wall Street Journal (*, February 2013.

[29] T. C. G. T. W. Group et al., "TPM Main Part 3 Commands," *Specification available online at: https://www. trustedcomputinggroup. org/specs/TPM/Main_Part3_Rev94. zip*, vol. 29, 2006.

[30] M. Hentea, "Improving security for SCADA control systems," *Interdisciplinary Journal of Information, Knowledge, and Management*, vol. 3, 2008, pp. 73–86.

[31] J. Hieb, J. Graham, and S. Patel, "Security enhancements for distributed control systems," *Critical Infrastructure Protection :*, 2007, pp. 133–146.

[32] D. Kaminsky, "Catching up with Kaminsky," *Network Security*, vol. 9, September 2008, pp. 4–7.

[33] D. Kaminsky, "DNS 2008 and the New (old) Nature of Critical Infrastructure," *Black-Hat DC, February*, 2009.

[34] R. H. Katz and G. Borriello, "Contemporary logic design," 2005.

[35] B. Lampson, M. Abadi, M. Burrows, and E. Wobber, "Authentication in distributed systems: Theory and practice," *ACM Transactions on Computer Systems (TOCS)*, vol. 10, no. 4, 1992, pp. 265–310.

[36] B. Laurie, G. Sisson, R. Arends, and D. B. Nominet, "DNS Security (DNSSEC) Hashed Authenticated Denial of Existence," *RFC*, vol. 5155, March 2008.

[37] D. Lie, C. A. Thekkath, and M. Horowitz, "Implementing an Untrusted Operating System on Trusted Hardware," *Proceedings of the 19th ACM Symposium on Operating Systems Principles*. October 2003, pp. 178–192, ACM Press.

[38] M. Majdalawieh, F. Parisi-Presicce, and D. Wijesekera, "DNPSec: Distributed network protocol version 3 (DNP3) security framework," *Advances in Computer, Information, and Systems Sciences, and Engineering :*, 2006, pp. 227–234.

[39] C. Martel, G. Nuckolls, P. Devanbu, M. Gertz, A. Kwong, and S. Stubblebine, "A general model for authentic data publication," *Algorithmica (Springer)*, 2001.

المنارة للاستشارات
www.manaraa.com

[40] A. Matrosov, E. Rodionov, D. Harley, and J. Malcho, "Stuxnet under the microscope," *eset, September*, 2010.

[41] J. M. McCune, B. Parno, A. Perrig, M. K. Reiter, and A. Seshadri, "How low can you go?: recommendations for hardware-supported minimal TCB code execution," *ACM SIGARCH Computer Architecture News*. ACM, 2008, vol. 36, pp. 14–25.

[42] R. C. Merkle, "Protocols for Public Key Cryptosystems," *Proceedings of the 1980 IEEE Symposium on Security*. 1980, and Privacy.

[43] P. Mockapetris, "Domain Names - Implementation and Specifications," *RFC*, vol. 1035, December 1987.

[44] D. Moore, V. Paxson, S. Savage, C. Shannon, S. Staniford, and N. Weaver, "Inside the Slammer Worm," *IEEE Security and Privacy*, vol. 1, no. 4, July 2003, pp. 33–39.

[45] R. Needham and M. Schroeder, "Using encryption for authentication in large networks of computers," *Communications of the ACM*, vol. 21, no. 12, December 1978.

[46] B. C. Neuman and T. Ts'o, "Kerberos: An Authentication Service for Computer Networks," *IEEE Communications*, September 1994, pp. 33–38.

[47] A. Patel, J. C. Junior, and J. M. Pedersen, *An Intelligent Collaborative Intrusion Detection and Prevention System for Smart Grid Environments*, Computer Standards & Interfaces, 2013.

[48] J. Pollet, "Developing a Solid SCADA Security Strategy," *Sensors for Industry Conference, Houston, Texas, USA*, December 2002, pp. 19–21.

[49] M. Ramkumar, "On the scalability of an efficient "Nonscalable" key distribution scheme," *World of Wireless, Mobile and Multimedia Networks, 2008. WoWMoM 2008. 2008 International Symposium on a*. IEEE, 2008, pp. 1–6.

[50] M. Ramkumar, "Trustworthy Computing Under Resource Constraints With the DOWN Policy," *IEEE Transactions on Secure and Dependable Computing, pp*, vol. 5, no. 1, 2008, pp. 49–61.

[51] B. Robert-son, "Integrating Security into SCADA Solutions," *NISCC SCADA Security Conference*, vol. 2003.

[52] L. F. Sarmenta, M. Van Dijk, C. W. O'Donnell, J. Rhodes, and S. Devadas, "Virtual monotonic counters and count-limited objects using a TPM without a trusted OS," *Proceedings of the first ACM workshop on Scalable trusted computing*. ACM, 2006, pp. 27–42.

[53] A. Shahzad and S. Musa, *Cryptography and Authentication Placement to Provide Secure Channel for SCADA Communication*, International Journal of Security, 2012.

المنارة للاستشارات

www.manaraa.com

[54] S. W. Smith and S. Weingart, "Building a High-Performance Programmable Secure Coprocessor," *Computer Networks*, 1999, pp. 831–860.

[55] A. Sood and R. Enbody, "Targeted Cyber Attacks-A Superset of Advanced Persistent Threats," *Security & Privacy, IEEE*, vol. 11.1, 2013, pp. 54–61.

[56] A. D. Sorbo, *Network Security - Sk-DNSSEC: an alternative to the Public Key scheme*, doctoral dissertation, Department of Computer Science, University of Salerno, Baronissi, Italy.

[57] E. R. Sparks and E. R. Sparks, *A Security Assessment of Trusted Platform Modules Computer Science Technical Report TR2007-597*, Tech. Rep., Technical report, Department of Computer Science Dartmouth College, 2007.

[58] T. Specification, "Architecture Overview," *Specification Revision*, vol. 1, 2004.

[59] T. Specification, "Trusted platform module main specification Version 1.2," *Revision 94*, vol. 1, 2006.

[60] J. Stamp, J. Dillinger, W. Young, and J. DePoy, "Common Vulnerabilities in Critical Infrastructure Control Systems," *Sandia National Laboratories report SAND*, 2003, pp. 2003–1772.

[61] K. Stouffer, J. Falco, and K. Kent, "Guide to Supervisory Control and Data Acquisition (SCADA) and Industrial Control Systems Security," *NIST Special Publication*, 2006, pp. 800–82.

[62] G. E. Suh, D. Clarke, B. Gassend, M. Van Dijk, and S. Devadas, "AEGIS: architecture for tamper-evident and tamper-resistant processing," *Proceedings of the 17th annual international conference on Supercomputing*. ACM, 2003, pp. 160–171.

[63] I. Trial, "Use Standard for SCADA Serial Link Cryptographic Modules and Protocol," *P*, 2008, pp. 2008–08.

[64] P. Tsang and S. Smith, "YASIR: A low-latency, high-integrity security retrofit for legacy SCADA systems," *Proceedings of The Ifip Tc 11 23 rd International Information Security Conference*, Boston, 2008, Springer.

[65] R. J. Turk, *Cyber incidents involving control systems*, Idaho National Engineering and Environmental Laboratory, 2005.

[66] M. Van Dijk, L. F. Sarmenta, J. Rhodes, and S. Devadas, *Securing shared untrusted storage by using TPM 1.2 without requiring a trusted OS*, Tech. Rep., Citeseer, 2007.

[67] P. C. van Oorschot, A. Somayaji, and G. Wurster, *Hardware-Assisted Circumvention of Self-Hashing Software Tamper Resistance*, IEEE Transactions on Dependable and Secure Computing, 2005.

المنارة للاستشارات
www.manaraa.com

[68] A. Velagapalli and M. Ramkumar, “An Efficient Trusted Computing Base (TCB) for a SCADA System Monitor,” *10th International Information and Telecommunication Technologies Conference*. IEEE, 2011.

[69] A. Velagapalli and M. Ramkumar, “Minimizing the TCB for securing SCADA systems,” *Proceedings of the Seventh Annual Workshop on Cyber Security and Information Intelligence Research*. ACM, 2011, p. 19.

[70] A. Velagapalli and M. Ramkumar, “An Efficient TCB for a Generic Data Dissemination System,” *International Conference on Communications in China: Communications Theory and Security (ICCC'12–CTS)*. IEEE, 2012.

[71] A. Velagapalli and M. Ramkumar, “Trustworthy tcb for dns servers,” *International Journal of Network Security*, vol. 14, no. 4, 2012, pp. 187–205.

[72] A. Velagapalli and M. Ramkumar, “A Security Architecture for SCADA Systems,” submitted for IEEE transactions on Cybernetics, 2013.

[73] P. Vixie, O. Gudmundsson, and D. Eastlake, *B. Wellington,” Secret Key Transaction Authentication for DNS (TSIG)*, Tech. Rep., RFC 2845, May, 2000.

[74] Y. Wang and B. Chu, “sSCADA: Securing SCADA infrastructure communications,” *Cryptology ePrint Archive, Report*, 2004.

[75] N. Weaver, V. Paxson, S. Staniford, and R. Cunningham, “A taxonomy of computer worms,” *Proceedings of the 2003 ACM workshop on Rapid malcode*, DC, USAOctober Washington, DC, USA, 2003, pp. 27–27, October Washington.

[76] S. Weiler and J. Ihren, *Minimally covering NSEC records and DNSSEC on-line signing*, Tech. Rep., RFC 4470, April, 2006.

[77] R. Wightman, *Cyber attacks on Texas utility*, Tech. Rep., Nov. 2011, http://www.washingtontimes.com/news/2011/nov/18/hackers-apparently-based-in-russia-attacked-a-publ.

[78] R. Wightman, “Spear Phishing Attempt,” *Digital Bond*, February 2013, https://www.digitalbond.com/blog/2012/06/07/spear-phishing-attempt/.

[79] R. Wojtczuk and J. Rutkowska, “Attacking SMM memory via Intel CPU cache poisoning,” *Invisible Things Lab*, 2009.

[80] A. Wright, J. Kinast, and J. McCarty, “Low-latency cryptographic protection for SCADA communications,” *Applied Cryptography and Network Security*, Springer, Berlin, 2004.

[81] J. Wright, “KillerBee: Framework and tools for exploiting ZigBee and IEEE 802.15.4 networks,” *Version*, vol. 1.0, 2010.

للاستشارات

www.manaraa.com

[82] B. Zhu, A. Joseph, and S. Sastry, "A Taxonomy of Cyber Attacks on SCADA Systems," *Proceedings of the 2011 International Conference on Internet of Things and 4th International Conference on Cyber, Physical and Social Computing*, Washington, DC, USA, 2011, ITHINGSCPSCOM '11, pp. 380–388, IEEE Computer Society.

[83] B. Zhu and S. Sastry, "SCADA-specific intrusion detection/prevention systems: a survey and taxonomy," *Proceedings of the 1st Workshop on Secure Control Systems (SCS)*, 2010.

المنارة للاستشارات
www.manaraa.com

APPENDIX A

STCB INSTRUCTION SET

المنارة للاستشارات
www.manaraa.com

This appendix illustrates a possible design of an STCB instruction set by outlining memory layout for addressable volatile registers in the STCB module and identifying useful logical operations on values stored in such registers. The memory layout $\mathbf{Y}$ is given in Table A.1, where index represents the position. For simplicity, we have associated each index with a more-easy-to-remember mnemonic.

A register used for storing the intermediate results during the execution of an algorithm $\mathbf{A}$ is given index 0 and mnemonic TMP. $\mathbf{Y}$[1-3] are values received in an authenticated sensor report. $\mathbf{Y}$[4-10] correspond to values in a sensor state $\mathbf{s_0}$, $\mathbf{Y}$[11-16] correspond to sensor state $\mathbf{s_1}$, $\mathbf{Y}$[17-22] corresponding to sensor state $\mathbf{s_2}$, $\mathbf{Y}$[23-28] correspond to sensor state $\mathbf{s_3}$. $\mathbf{Y}$ [29-37] correspond to the l constants. For illustration let the maximum number of related sensors be $q = 3$; the number of outputs of each $\mathcal{U}$ be $w = 3$; and the number of constants $l = 8$.
newspacing1

The algorithm $\mathbf{A}$ consists of m sets of three byte instructions (m $\times$ 3, m = 8) of the form (o, a_1, a_2') where

1. the first byte o specifies one of 256 possible operations including 0 (no operation);
2. the second byte a_1 specifies the address of the first operand (an address between 0 and 64);
3. the third byte $a_2' = a_2 \parallel i$ specifies using its MSBs a_2, the address of the second operand (between 0 and 64); the two LSBs $i = \{0, 1, 2, 3\}$ specify the address of destination of the result of the operation as follows:

i	output address
0	TMP
1	E01 (o_{0_1})
2	E02 (o_{0_2})
3	E03 (o_{0_3})

المنارة للاستشارات
www.manaraa.com

Table A.1

Memory Layout of $\mathbf{Y}$

Index	Mnemonic	Index	Mnemonic	Index	Mnemonic	Index	Mnemonic
0	TMP						
1	RID ($\tilde{S}$)	2	RVAL($\tilde{v}$)	3	RTS ($\tilde{t}$)		
4	SID0 (S_0)	5	V0 (u_0)	6	T0 (t_0)	7	TM (τ_0)
8	E01 (o_{0_1})	9	E02 (o_{0_2})	10	E03 (o_{0_3})		
11	SID1 (S_1)	12	V1 (u_1)	13	T1 (t_1)		
14	E11 (o_{1_1})	15	E12 (o_{1_2})	16	E13 (o_{1_3})		
17	SID2 (S_2)	18	V2 (u_2)	19	T2 (t_2)		
20	E21 (o_{2_1})	21	E22 (o_{2_2})	22	E23 (o_{2_3})		
23	SID3 (S_3)	24	V3 (u_3)	25	T3 (t_3)		
26	E31 (o_{3_1})	27	E32 (o_{3_2})	28	E33 (o_{3_3})		
29	C0	30	C1	31	C2	32	C3
33	C4	34	C5	35	C6	36	C7
37	C8						

Thus, the instructions $\mathbf{A}$ can only modify contents of locations TMP, E01, E02 and E03.

A.1 Opcodes

A partial listing of the interpretation of some opcodes are depicted in Table A.2.

Table A.3 illustrates the execution of some sample 3 byte instructions, depicting the type of operation, the operands involved, and the location of the output.

In general, while two bytes in an instruction specifies two inputs and one output, operations like ENC3, ADD3, AND3, OR3, etc., have *three* inputs and one output. For such operations, the third input is the same as the output (which can be TMP, E01, E02 or E03). On execution of the operation, the output overwrites the third input.

Operations with mnemonic ending with 4 also have three inputs and one output, but the third input is distinct from the output. This is achieved by using the first address byte a_1 to

المنارة للاستشارات
www.manaraa.com

Table A.2

A Partial Listing of Opcodes and their Interpretation

OPCODE	Interpretation
0	No operation
CHKC	Next two bytes indicate identifier for a set of constants
ULUT	Use 1-D LUT
ULUT2	Use 2-D LUT
ULUT3	Use 3-D LUT
AND11	Logical AND operation
AND10	Logical AND operation after negating the first operand
AND00	Logical AND operation after negating both operands
AND3	Logical AND of three operands
OR11	Logical OR operation
OR10	Logical OR operation after negating the first operand
OR00	Logical OR operation after negating both operands
AND3	Logical AND of three operands
ADD11	Add first and second operand
ADD10	Add first operand with negative of second operand
ADD00	Add negative of both operands
ADD3	Addition of three operands
LT	first operand $<$ than second
LTE	first operand $\leq$ than second operand
EQ	both operands are equal
ENCS3	Strict Enclosure
ENC3	Enclosure
CENC3	Circular Enclosure
CMP	Compare if first operand is </==/> than second
LIM4	Check if operand 2 is within lower and upper limits
⋮	

www.manaraa.com

Table A.3

Examples Illustrating 3-byte Instructions.

Instruction	Interpretation
(AND00,C4,T1 $\|$ 0)	$!Y[\mathrm{C4}] \wedge !\mathrm{Y}[\mathrm{T1}] \rightarrow \mathrm{Y}[\mathrm{TMP}]$
(OR00, V2, TMP $\|$ 1)	$Y[\mathrm{V2}] \vee \mathrm{Y}[\mathrm{TMP}] \rightarrow \mathrm{Y}[\mathrm{E01}]$
(OR10, V3, C8 $\|$ 2)	$Y[\mathrm{V3}] \vee !\mathrm{Y}[\mathrm{C8}] \rightarrow \mathrm{Y}[\mathrm{E02}]$
(LT, V3, C8 $\|$ 3)	$Y[\mathrm{E03}] = \mathrm{Y}[\mathrm{V3}] < \mathrm{Y}[\mathrm{C8}]?1:0$
(ENCS3, V2, T1 $\|$ 1)	$Y[\mathrm{E01}] = (\mathrm{Y}[\mathrm{V2}] < \mathrm{Y}[\mathrm{E01}] < \mathrm{Y}[\mathrm{T1}])?1:0$
(ENC3, V2, T1 $\|$ 2	$Y[\mathrm{E02}] = (\mathrm{Y}[\mathrm{V2}] \leq \mathrm{Y}[\mathrm{E02}] \leq \mathrm{Y}[\mathrm{T1}])?1:0$
(CMP,V0,C5 $\|$ 1)	IF $Y[\mathrm{V0}] < \mathrm{Y}[\mathrm{C5}]$ then $Y[\mathrm{E01}] = -1$
	IF $Y[\mathrm{V0}] > \mathrm{Y}[\mathrm{C5}]$ then $Y[\mathrm{E01}] = 1$
	IF $(Y[\mathrm{V0}] = \mathrm{Y}[\mathrm{C5}])$ then $Y[\mathrm{E01}] = 0$
(CMP3, C1,C5 $\|$ 1)	IF $Y[\mathrm{E01}] < \mathrm{Y}[\mathrm{C1}]$ then $Y[\mathrm{E01}] = -1$
	IF $Y[\mathrm{E01}] > \mathrm{Y}[\mathrm{C5}]$ then $Y[\mathrm{E01}] = 1$
	IF $Y[\mathrm{C1}] \leq \mathrm{Y}[\mathrm{E01}] \leq \mathrm{Y}[\mathrm{C5}]$ then $Y[\mathrm{E01}] = 1$
(ADD3, E11,C5 $\|$ 2)	$Y[\mathrm{E02}] = \mathrm{Y}[\mathrm{E02}] + \mathrm{Y}[\mathrm{E11}] + \mathrm{Y}[\mathrm{C5}]$
(LIM4,C1,V0 $\|$ 3)	$Y[\mathrm{E03}] = (\mathrm{Y}[\mathrm{C1}] < \mathrm{Y}[\mathrm{V0}] < \mathrm{Y}[\mathrm{C1}+1])?1:0$

المنارة للاستشارات
www.manaraa.com

specify two operands. Specifically, if $a_1 = x$, two operands are $Y[x]$ and $Y[x+1]$. The byte a_2 (as usual) specifies the third input (6 MSBs) and the output location (two LSBs).

While most operations are trivial and obvious from the mnemonic employed for the operation, some are not. An algorithmic representation of the execution of some not so obvious operations is provided in Figure **??**.

A.1.1 Internal Functions

Internal function (see Figure A.1) $f_{exec}()$ execute the 24 byte instruction in STCB memory $\mathbf{v}$. Specifically, $f_{exec}()$ makes 8 calls to $f_{eoc}()$ (which executes one three byte instruction o, a_1, a_2').

```
f_exec() {
  FOR (j=0 TO 7)
    x := f_eoc(v[3j], v[3j+1], v[3j+2]);
    IF (x) RETURN x;
}
```

Figure A.1

Internal function $f_{exec}()$

Function $f_{eoc}(o, a_1, a_2')$ has three possible return values

1. 0 - on successful execution;
2. 1 - if $o = 0$, signifying that no more instructions are available.
3. ERROR - if any of the *preconditions* necessary for execution of the instruction is not satisfied.

المنارة للاستشارات
www.manaraa.com

If $f_{eoc}()$ returns 0 the function $f_{exec}()$ proceeds with the next instruction (unless it is the last instruction). If $f_{eoc}()$ returns 1, f_{exec} stops the execution and returns 1; if f_{eoc} returns ERROR, $f_{exec}()$ returns ERROR.

The first byte specifies an instruction. For most operations, byte a_1 specifies an input address (between 0 and 64), and byte a'_2 simultaneously specifies an input address (between 0 and 64) and an output address (between 0 and 3).

Specifically, $a'_2 = a_2 \parallel i$ where the six MSBs a_2 represents an input address and the two LSBs $op \in \{0, 1, 2, 3\}$ represents an output address $out = \{$ TMP,E01,E02,E03$\}$.

Figure A.2 provides a partial listing of function $f_{eoc}()$. Note that for all operations with a mnemonic that ends with 3, one input is the same as the output $Y[out]$. The operation CENC3 (circular enclosure) returns true if input $Y[out]$ is circularly enclosed by values $Y[a_1]$ and $Y[a_2]$. A value x is circularly enclosed by v_1 and v_2 is strictly enclosed by v_1 and v_2 or if $v_2 < v_1$ *and* $x < v_2$; or if $v_2 < v_1$ and $x > v_1$.

المنارة للاستشارات
www.manaraa.com

```
x := f_eoc(o, a1, a2')
{
  IF (o=0) RETURN 1;
  IF (o=CCHK)
    IF (Y[C0] ≠ a1 + (a2' << 8)) RETURN ERROR;
    ELSE RETURN 0;
  a2 := a2' >> 2; // six MSBs of a2 is operand 2 address;
  out := a2' << 6;//two LSBs of a2 is output address;
  IF (out = 0) out := V0;
  ELSE IF (out = 1) out := E01;
  ELSE IF (out = 2) out := E02;
  ELSE out := E03;
  ELSE IF (o = ULUT)
    IF !(Y[C1] < Y[a1] < Y[C2])) RETURN ERROR ;
    RETURN 0;
  IF (o = ULUT2)
    IF !(Y[C1] ≤ Y[a1] < Y[C2]) RETURN ERROR ;
    IF !(Y[C3] ≤ Y[a2] < Y[C4]) RETURN ERROR ;
    RETURN 0;
  IF (o = ULUT3)
    IF !(Y[C1] ≤ Y[a1] < Y[C2]) RETURN ERROR ;
    IF !(Y[C3] ≤ Y[a2] < Y[C4]) RETURN ERROR ;
    IF !(Y[C5] ≤ Y[out] < Y[C6]) RETURN ERROR ;
    RETURN 0;
  ELSE IF (o = LIM4)
    Y[out] := (Y[a1] < Y[a2] < Y[a1 + 1]); RETURN 0;
  ELSE IF (o = CENC3)
    Y[out] := (Y[a1] < Y[out] < Y[a2]) ∨ (Y[a2] < Y[a1] < Y[out])
    Y[out] := Y[out] : ∨(Y[out] < Y[a2] < Y[a1]);
    RETURN 0;
  ELSE IF (o = ENCS3)
    Y[out] := (Y[a1] < Y[out] < Y[a2]); RETURN 0;
  ELSE IF (o = ENC3)
    Y[out] := (Y[a1] ≤ Y[out] ≤ Y[a2]); RETURN 0;
  ELSE IF ···
    ⋮
}
```

Figure A.2

Internal function $f_{eoc}()$

المنارة للاستشارات
www.manaraa.com

A.2 Illustration of STCB System Design

In this section we illustrate the process of construction of the design tree by considering a simple SCADA system for controlling a water-tank, with four sensors and two actuators. The four sensors labelled HH, HI, LO and LL are level sensors placed at different heights in the water tank (see Figure **??**). An actuator PP turns on/off the pump which pumps water into the tank, and an actuator VL turns on/off the outlet valve.

We shall represent by PP, VL, HH, HI, LO and LL, the identities of the six sensors/actuators. We shall denote by p, v, hh, hi, lo, ll their respective instantaneous states. Specifically,

1. $p = 1$ if the pump is on; $p = 0$ if the pump is off;
2. $v = 1$ if valve is on; $v = 0$ when valve is turned off (no discharge);
3. $hh = 1$ implies water level at or above above the height at which the sensor HH is placed, $hh = 0$ implies water level below position of sensor HH. The same logic holds for other sensor states hi, lo and ll.

A.2.1 Design Steps

The steps in the design of the system are as follows:

1. specify unique identities for each sensor/actuator.
2. specify minimal acceptable frequency of reports from the sensor (by specifying duration of validity).
3. specify up to three dependent sensors/actuators for each sensor/actuator,
4. specify up to 24 bytes of instructions for each sensor (if necessary),
5. specify constant values, if required
6. construct Merkle hash tree and compute root ξ_s.

المنارة للاستشارات
www.manaraa.com

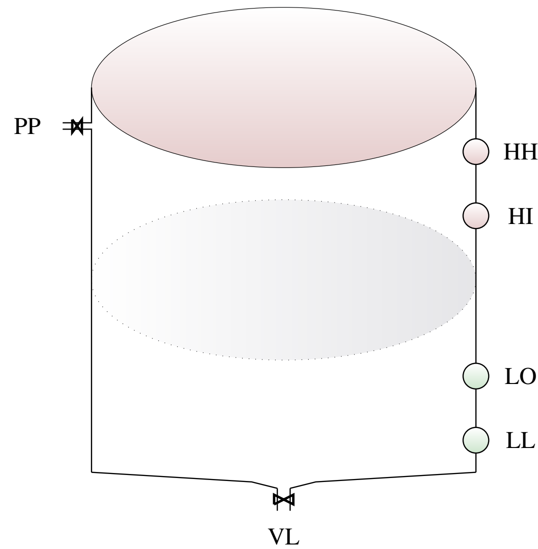

Figure A.3

Water tank SCADA system

The freedom to specify the dependent sensors, instructions, and constants is used to describe the physics of the system.

The first two steps are trivial. For the first step we shall assume that the sensor/actuator identities are PP, VL, HH, HI, LO and LL. For the second step we shall impose that level sensors are expected to report their state (0 or 1) at least once every ten seconds, and that the actuators PP and VL are required to report at least once every 5 seconds.

A.2.1.1 Valid and Invalid States

Among the sixteen possible states of four bits representing hh, hi, lo and ll all but five are unacceptable as they indicate failure of a sensor. For example, if $hh = 1$, then

المنارة للاستشارات
www.manaraa.com

$hi = lo = ll = 1$. If $lo = 1$ the it follows that $ll = 1$, and so on. The Karnaugh map [34] in Table A.4 depicts the five acceptable conditions for sensors HH, HI, LO and LL.

Table A.4

Karnaugh Maps for Water Tank System.

$hh=0, hi=0$	$hh=0, hi=1$	$hh=1, hi=1$	$hh=1, hi=0$	
1	0	0	0	$lo=0, ll=0$
1	0	0	0	$lo=0, ll=1$
1	1	1	0	$lo=1, ll=1$
0	0	0	0	$lo=1, ll=0$

$p=0, v=0$	$p=0, v=1$	$p=1, v=1$	$p=1, v=0$	
0	0	0	1	$hh=0, ll=0$
1	1	1	1	$hh=0, ll=1$
1	1	0	0	$hh=1, ll=1$
0	0	0	0	$hh=1, ll=0$

From inspection of the map it is obvious that the current sensor states are acceptable only if the expression

$$e_1 = (\bar{h}h \wedge \bar{h}i \wedge \bar{l}o) \vee (\bar{h}h \wedge \bar{h}i \wedge ll) \vee (hi \wedge lo \wedge ll) \quad \text{(A.1)}$$

evaluates to TRUE.

Furthermore, it is desired that

1. if water level is above HH then pump should be off
2. if water level is below LL then pump should be on and valve should be off.

A Karnaugh map depicting acceptable states of values p, v, hh and ll is also depicted in Table A.4.

www.manaraa.com

From inspection of the map

$$e_2 = (\bar{p} \wedge ll) \vee (\bar{h}h \wedge ll) + (p \wedge \bar{v} \wedge \bar{h}h)$$

evaluates to TRUE for acceptable states. Overall, the water tank system is deemed to be in an acceptable state only if both e_1 and e_2 are true. In other words, evaluation of the function $\mathcal{F}()$ for this SCADA system (to determine of the system is in an acceptable state) boils down to evaluation of $e_1 \wedge e_2$.

A.2.2 Design Tree Leaves

For this particular example, no predefined constants are required to evaluate $\mathcal{F}()$. Thus, the design tree will possess only six leaves - one corresponding to each sensor/actuator.

Recall that the design records have the sensor id , identities of all the related sensors, and an instruction set (24 byte instruction) $\mathbf{A}$. Those values mentioned in design record are specified by the designer. All the rest of values like are set to 0 at design time.

Let us assume that e_1 is evaluated every time a fresh value corresponding to sensor HH is provided. For this purpose the current values of three other sensors HI, LO and LL are required. Towards this end the designer specifies HI, LO and LL as dependent leaves for leaf HH. More specifically, in the design leaf for sensor HH, we have

$$S_0 = \mathrm{HH}, \mathrm{S}_{01} = \mathrm{HI}, \mathrm{S}_{02} = \mathrm{LO}, \text{ and } \mathrm{S}_{03} = \mathrm{LL}$$

The 24 bytes $\mathbf{A}$ specified in leaf HH evaluates e_1.

Similarly, let us assume that e_2 is evaluated every time a fresh value corresponding to sensor PP is provided. For this purpose as current states of actuator VL, and sensors HH

المنارة للاستشارات
www.manaraa.com

and LL , are required, they are specified as the dependent sensors of PP. In other words, in the leaf for sensor PP, we have

$$S_1 = \text{PP}, \text{S}_{11} = \text{VL}, \text{S}_{12} = \text{HH}, \text{ and } \text{S}_{13} = \text{LL}$$

The 24 bytes $\mathbf{A}$ specified in leaf PP evaluates e_2.

The other four leaves - viz, VL, HI, LO and LL do not require dependent leaves to be specified. Furthermore, no $\mathbf{A}$ need to be specified for such leaves. The table below depicts the contents of values of the six leaves in the design tree:

S_{id}	PP	VL	HH	HI	L0	LL
S_{id1}	VL	0	HI	0	0	0
S_{id2}	HH	0	LO	0	0	0
S_{id3}	LL	0	LL	0	0	0

A.2.3 Instructions for Leaves PP and HH

The instructions specified in leaf HH evaluates the e_1. Before the instructions are executed $F_{\text{upd}}()$ ensures the following

1. The current contents of leaf HH - values - are stored in locations are stored in locations $\mathbf{s_0}$
2. Contents of a fresh report from HH (indicating sensor identity, sensed values and time stamp) are stored in locations $\mathbf{r}$.
3. $Y[\text{SID0}] = \text{HH} = \text{Y}[\text{RID}]$
4. Value $Y[\text{V0}]$ is updated (by setting it equal to $Y[\text{RVAL}]$)

المنارة للاستشارات
www.manaraa.com

5. Value $Y[\mathrm{T0}]$ is updated based on time stamp $Y[\mathrm{RTS}]$
6. The current contents of leaf HI are stored in locations are stored in locations $\mathbf{s_1}$.
7. The current contents of leaf LO are stored in locations are stored in locations $\mathbf{s_2}$.
8. The current contents of leaf LL are stored in locations are stored in locations $\mathbf{s_3}$.

For computing e_1, the required values are

$$hh = Y[\mathrm{V0}], \mathrm{hi} = \mathrm{Y[V1]}, \mathrm{lo} = \mathrm{Y[V2]} \text{ and } \mathrm{ll} = \mathrm{Y[V3]}$$

The 24 bytes in **A**compute

$$e_1 \quad = \quad (\bar{h}h \wedge \bar{h}i \wedge \bar{l}o) \vee (\bar{h}h \wedge \bar{h}i \wedge ll) \vee (hi \wedge lo \wedge ll)$$

using the following sequence of six operations

1. $Y[\bar{\mathrm{V}}0] \wedge Y[\bar{\mathrm{V}}1] \rightarrow Y[\mathrm{TMP}]$ or (AND00, V0, V1 || 0)
2. $Y[\bar{\mathrm{V}}2] \vee Y[\mathrm{V3}] \rightarrow \mathrm{Y[E01]}$ or (OR10, V3, V2 || 1)
3. $Y[\mathrm{TMP}] \wedge \mathrm{Y[E01]} \rightarrow \mathrm{Y[E01]}$ or (AND11,24, 0 || 1)
4. $Y[\mathrm{V1}] \wedge \mathrm{Y[V2]} \rightarrow \mathrm{Y[TMP]}$ or (AND11,V2,V1 || 0)
5. $Y[\mathrm{TMP}] \wedge \mathrm{Y[V3]} \rightarrow \mathrm{Y[TMP]}$ or (AND11, V3, 0 || 0)
6. $Y[\mathrm{TMP}] \vee \mathrm{Y[E01]} \rightarrow \mathrm{Y[E01]}$ or (OR11, E01, 0 || 1)

As only six instructions are required, the last two instructions in **A** are set to (0,0,0). At the end of the execution , location E01 - which corresponds to o_{0_1} of leaf HH, contains the result e_1 of the evaluation.

The 24 byte instruction specified in leaf PP evaluates the e_2, and $e_1 \wedge e_2$. Before the instructions are executed, $F_{\mathrm{upd}}()$ ensures the following

1. The current contents of leaf PP - values at $\mathbf{s_0}$

المنارة للاستشارات
www.manaraa.com

2. Contents of a fresh report from PP are stored in **r**
3. $Y[\text{SID0}] = \text{PP} = \text{Y[RID]}$
4. Value $Y[\text{V0}]$ is updated
5. Value $Y[\text{T0}]$ is updated
6. $\mathbf{s_1}$ = VL
7. $\mathbf{s_2}$= HH.
8. $\mathbf{s_3}$ = LL.

For computing e_2, the required values are

$$p = Y[\text{V0}], \text{v} = \text{Y[V1]}, \text{hh} = \text{Y[V2]}, \text{ll} = \text{Y[V3]}, \text{ and } \text{e}_1 = \text{Y[E21]}$$

Note that e_1 is the value loaded to $Y[\text{E21}]$. Also every time $F_{\text{upd}}()$ is performed $Y[\text{TM}]$ is updated to $\min(Y[\text{T0}], \tau_1', \tau_2', \tau_3')$

The instruction set **A** in leaf PP is executed to compute

$$\begin{aligned} e_2 &= (\bar{p} \wedge ll) \vee (\bar{h}h \wedge ll) + (p \wedge \bar{v} \wedge \bar{h}h) \\ &= (!Y[\text{V0}] \wedge \text{Y[V3]}) \vee (\text{!Y[V2]} \wedge \text{Y[V3]}) \vee (\text{Y[V0]}\wedge\text{!Y[V1]} \wedge \text{Y[V2]}), \end{aligned}$$

after which e_2 is made available in location $Y[\text{E01}]$. This value is combined with e_1 in $Y[\text{E21}]$ to compute and $e_1 \wedge e_2$, using the following sequence of six operations

1. $Y[\bar{\text{V}}0] \wedge Y[\bar{\text{V}}2] \rightarrow Y[\text{TMP}]$ or (AND00, V0, V2 || 0)
2. $Y[\text{V3}] \vee \text{Y[TMP]} \rightarrow \text{Y[TMP]}$ or (OR11, V3, TMP || 0)
3. $Y[\text{V0}]\wedge\text{!Y[V1]} \rightarrow \text{Y[E01]}$ or (AND10, V0, V1 || 1)
4. $Y[\text{E01}] \wedge \text{Y[V2]} \rightarrow \text{Y[E01]}$ or (AND11,E01, V2 || 1)
5. $Y[\text{E01}] \vee \text{Y[TMP]} \rightarrow \text{Y[E01]}$ or (OR11, E01, TMP || 1)
6. $Y[\text{E01}] \wedge \text{Y[E12]} \rightarrow \text{Y[E01]}$ or (AND11, E01, E12 || 1), $(e_1 \wedge e_2)$

Ultimately, the state of the system is indicated by the value o_{1_1} in sensor state of PP.

المنارة للاستشارات
www.manaraa.com