Chapter 7

PROTECTING INFRASTRUCTURE ASSETS FROM REAL-TIME AND RUN-TIME THREATS

Jonathan Jenkins and Mike Burmester

Abstract Real-time availability with integrity is a crucial security requirement for critical infrastructure assets – delays in reporting device states or computations may result in equipment damage, perhaps even catastrophic failure. However, it is also necessary to address malicious software-based threats. Trusted computing (TC) is a security paradigm that enables application platforms to enforce the integrity of execution targets. A TC architecture can be combined with a real-time access control system to help protect against real-time availability and malware threats. However TC architectures offer only static (load-time) protection, so it is still necessary to address the possibility of run-time (execution) attacks. This paper focuses on the protection afforded by TC platforms to critical infrastructure assets. The paper defines a threat model, analyzes vulnerabilities, proposes services and tools that guarantee real-time availability with integrity, and demonstrates how they can be used to protect communications of an IEC61850-90-5-compliant substation automation system in an electricity grid. Also, it discusses the impact of run-time attacks on TC-compliant critical infrastructure assets.

Keywords: Trusted computing, real-time threats, power grid communications

1. Introduction

The Stuxnet worm has demonstrated the effectiveness of cyber attacks on industrial facilities. The potentially devastating effects of such attacks are of great concern and highlight the need for solutions that provide effective protection for industrial facilities and, in particular, critical infrastructure systems. The crucial nature of the services provided by critical infrastructure systems and the vulnerabilities found in these systems necessitate an approach to security that focuses on both real-time availability and malware threats. New

J. Butts and S. Shenoi (Eds.): Critical Infrastructure Protection VII, IFIP AICT 417, pp. 97–110, 2013.

technologies that are based on the trusted computing paradigm are needed to augment trust in emerging infrastructure systems and to mitigate threats from intentional and unintentional actors.

Trusted computing (TC) as defined by the Trusted Computing Group [26] is a technology that: (i) supports application platforms for securing distributed systems using trust engines that attest to the integrity of the systems; (ii) provides sealed storage; and (iii) enforces the expected behavior of the systems. This prevents the execution of untrusted (potentially malicious) software and also addresses insider threats. However TC architectures offer only static (load-time) protection, so the possibility of run-time (execution) attacks still has to be addressed.

In a critical infrastructure environment, it is vital that plant operators can access system state information in real-time – correct messages delivered at the wrong time can lead to erroneous responses and system failure. Communications must, therefore, be subject to real-time constraints with availability guaranteed within strict time bounds, a fact that renders most commonly-adopted cyber security paradigms inapplicable.

The three primary cyber security goals are confidentiality, integrity and availability. However, in the case of critical infrastructure systems, real-time availability with integrity is a primary goal, with privacy often only a secondary goal. This leads to a new security paradigm in which robustness in real-time is the main goal. This impacts implementation aspects because real-time behavior cannot be captured by traditional security paradigms.

It is difficult to provide effective protection to critical infrastructure systems with current operating systems, at least with regard to real-time availability and run-time integrity. A promising solution is to identify the key vulnerabilities of critical infrastructure systems and use a TC architecture that prevents the exploitation of the identified vulnerabilities.

This paper combines a TC architecture with a real-time access control infrastructure to implement effective protection against real-time availability and malware threats. Run-time threats are analyzed and approaches for mitigating them are discussed. The threats are combated by ensuring that network devices of TC-compliant systems operate with a limited enumerated set of functions and employ dynamic integrity monitoring that satisfies the tight real-time availability constraints of critical infrastructure systems.

2. Related Work

TC technologies have been used in various applications to augment trust in network devices and platforms (see, e.g., [16]), but there has been very limited work on applying TC to critical infrastructure assets. Metke and Ekl [18] propose the combination of a public key infrastructure with TC to secure critical infrastructure components, but they do not address the problem of run-time threats.

Considerable work has been done on defending systems from run-time attacks such as buffer overflows, but despite all the research, computer systems

and networks continue to be vulnerable to these attacks. Some researchers (see, e.g., [9, 10, 20]) have proposed the use of taint analysis and dynamic integrity monitoring of run-time software behavior to identify and track security violations. Various methods exist to enable tracking, such as code instrumentation, binary re-writing and architectural tracking support. Although serious efforts have been made to optimize the performance of monitoring processes [9], the performance overhead associated with dynamic measurements can be very significant. Architectural tracking methods are interesting, but they require expensive extensions to processors.

3. Trusted Computing

The Trusted Computing Group [26] has published specifications of architectures and interfaces for several computing implementations. Platforms based on these specifications are expected to meet functional and reliability requirements of computer systems that allow increased assurance and trust. As such, they are well suited to support and protect critical infrastructure assets. The trusted platform module [24] (TPM) and trusted network connect (TNC) [25] are two components that can help address the security threats of critical infrastructures.

The TPM is a TC architecture that binds data to platform configurations of hardware systems to enhance software security. It has two basic capabilities, remote attestation and sealed storage, and is supported by a range of cryptographic primitives and keys. The TPM architecture is defined in terms of trusted engines called roots of trust that are used to establish trust in the expected behavior of a system. There are three mandatory roots of trust: (i) root of trust for measurement (RTM), which makes reliable integrity measurements; (ii) root of trust for storage (RTS), which protect keys and data entrusted to the TPM; and (iii) root of trust for reporting (RTR), which exposes shielded locations for storing integrity measurements and attests to the authenticity of stored values.

Security is based on an integrity-protected boot process in which executable code and associated configuration data are measured before being executed – this requires a hash of the BIOS code to be stored in a platform configuration register (PCR). For remote attestation, the TPM uses an attestation identity key to assert the state of its current software environment to a third party by signing its current PCR values. Sealed storage is used to protect the cryptographic keys. The keys are released for encryption, decryption and authentication purposes, conditional on the current software state (using current PCR values). The Trusted Computing Group requires TPM modules to be physically protected from tampering. This includes binding the TPM to physical parts of the platform (e.g., the motherboard) so that it cannot be transferred.

The trusted network connect (TNC) is a TC interoperability architecture for trusted access control based on TPMs. TNC is distinguished from other interoperability architectures by the requirement that the operating system configurations of the client and server are checked before a communication

channel is established. A trusted link between a client and server is established only if:

- The identities of the client and server are trusted. A distributed public key infrastructure is used to establish trust links between a root authority and the TPMs of the client and server.
- The client is allowed real-time access to the server.
- The identities of the client and server are authenticated. A root of trust on the TPMs of both parties is invoked to release the required keys to execute a handshake protocol [25]. The TPM releases the keys only if the current configuration states of the operating systems of the parties permit the release.
- The handshake protocol is properly executed.
- The integrity and (if necessary) the confidentiality of the communicated data are enforced by the TPM.

4. Security Framework and Threat Model

The TPM prevents compromised components of TC-compliant systems from executing. As a result, if run-time threats are excluded, then malicious (Byzantine) threats are reduced to denial-of-service (DoS) threats. This is a radical departure from the traditional threat model for computer systems because, unlike Byzantine faults, DoS faults are overt – they are self-revealing and, hence, are detectable.

Critical infrastructure assets can be protected from such faults using reliability mechanisms such as replication and redundancy. It is well-known that tolerating f Byzantine faults in a distributed system requires $(2f + 1)$ redundancy (with reliable broadcast) [12]. For a DoS threat model, the faults are overt. Therefore, tolerating f DoS faults requires only $(f + 1)$ redundancy.

Two kinds of faults may affect a TC-compliant critical infrastructure: natural faults (including accidents) and adversarial (intentional, malicious or insider) DoS faults (excluding run-time attacks). Natural faults can be predicted in the sense that an upper bound on the probability of the events can be estimated. Redundancy can then be used to reduce the probability to a value below an acceptable threshold.

Malicious DoS faults cannot be predicted. However they are overt and, because of the TPM and TNC integrity verification, must be physical (e.g., involve the tampering of the TPM chip). Thus, there is a cost involved and one way to thwart such faults is to make the cost high enough to prevent them. Several security approaches use economics and risk analysis based on replication and redundancy [17] to address threat models with overt faults. These approaches assume a bound on adversarial resources and the presence of an architecture with sufficient redundancy to make DoS attacks prohibitively expensive.

Our approach for critical infrastructure protection assumes an architecture with sufficient redundancy such that: (i) the probability that the system will fail due to a natural fault is negligible (e.g., less than 2^{-20}); and (ii) the cost of a successful DoS attack by compromising system components is prohibitive for a resource-bounded adversary.

Our security framework for a critical infrastructure $\mathcal{C}$ comprises:

- A real-time model that captures its functionality, including a faults distribution $\mathcal{F}$ on components.
- A set $\mathcal{S}$ of specifications, policies, constraints and security requirements that identify the vulnerabilities $\mathcal{V}$ that must be addressed.
- An $\mathcal{S}$-profiler that emulates the behavior of $\mathcal{C}$ in real-time subject to the specifications $\mathcal{S}$. The profiler is defined by a software program that runs in real-time.
- A proof that the $\mathcal{S}$-profiler adheres to the security requirements specified by $\mathcal{S}$ in real-time in the presence of faults/disasters, accidents and malicious behavior.

This framework captures integrity and confidentiality and addresses availability in real-time. The $\mathcal{S}$-profiler is used to measure the actual time required to protect system resources. Note that traditional approaches based on formal methods or security models do not capture real-time availability.

4.1 Security Framework

Let $\mathcal{C}$ be a real-time $\mathcal{S}$-compliant critical infrastructure with faults distribution $\mathcal{F}$, specifications $\mathcal{S}$ and vulnerabilities $\mathcal{V}$. Then, $\mathcal{C}$ is modeled by a finite hybrid real-time automaton with faults [8]: $\mathcal{C} = (\tau, A, Q, q_0, D, \mathcal{F})$ with a time schedule $\tau : t_1, t_2 \ldots$; a finite set of actions A that includes a special symbol $\perp$; a finite set of states $Q \neq \emptyset$ that is partitioned into safe states Q_s, critical states Q_c and terminal states Q_t; an initial state $q_0 \in Q_s$; and a time-triggered transition function $D \subset Q \times Q \times A$. D is deterministic when $a \in A \backslash \{\perp\}$ and probabilistic when $a = \perp$; in this case, the posteriori state is selected by nature according to $\mathcal{F}$.

The parties involved in a critical infrastructure are specified by $\mathcal{S}$, e.g., intelligent electronic devices, operators, the adversary and nature (environment). Nature controls the temporal and spatial aspects of all events, schedules state transitions in a timely manner according to τ using the distribution $\mathcal{F}$ to select a strategy for component failure, and resolves concurrency issues by linking events to their actual start times.

For our application involving secure electricity grid communications, the transition frequency $\delta = t_{i+1} - t_i$ is 4 ms, which is the latency specified by IEC 61850-90-5 [13]. It is important that the critical infrastructure protection mechanisms adhere to the time-frame τ because command/control information

that arrives late may result in the system transitioning to a critical/terminal state.

A semantic security threat model is extended to capture real-time events. The threat model restricts the adversary to exploiting the system vulnerabilities $\mathcal{V}$ specified in $\mathcal{S}$, in particular, those identified by system policies, vulnerability assessments and gray-box penetration testing. The vulnerabilities involve system components such as control systems, embedded systems and communication channels.

4.2 The Good, the Bad and the Ugly

The inclusion of TC results in a threat model with significant benefits and unique limitations. This section provides an analysis of the effects of TC on the threat model.

The Good. The TPM is an interface that protects system components from behaving in an unexpected way during a malicious attack. This is achieved by having an integrity-protected boot process and using protected capabilities to access shielded locations for: (i) PCRs to verify integrity; and (ii) cryptographic keys for integrity and confidentiality.

Before an authorized program is executed, an integrity check of its state against a stored PCR configuration is performed. If the check fails, the program is assumed to have been compromised and is not executed by the operating system.

The Bad. The TPM allows only authorized software programs to execute. Therefore, the integrity of system software, which includes the operating system, is a fundamental requirement for ensuring trust in the computing infrastructure. The system software must be well designed, and not have security holes, backdoors or other vulnerabilities that could be exploited by an adversary.

A vulnerability in the operating system may allow an adversary to bypass the protection offered by the TPM. There are many reasons why software programs used in critical infrastructures may have faulty designs. A major reason is the complexity and architectural constraints of the execution environment (i.e., operating system and hardware). Other reasons are poor software development practices and inadequate applications security. With such systems, a dynamic approach is typically used in which program flaws are addressed with patches. However, this approach does not work for critical infrastructures.

The Ugly. The TPM provides integrity verification only at load-time, not at run-time. By exploiting a vulnerability in the operating system, an adversary may be able to change the execution flow of a program, e.g., by using a buffer overflow attack. Several run-time attacks [1], such as those that use metamorphic malware (e.g., the self-camouflaging Frankenstein [19]) or more generally, return-oriented programming (ROP) [21], require the adversary to

be able to control the flow of execution on the stack. However, there are ways to prevent this [11]. Unfortunately, these approaches do not combat run-time attacks that exploit system vulnerabilities.

5. Real-Time Availability Threats

Real-time availability threats exploit the time required to deliver system resources. For example, in an electrical grid, controllers must have access to state transition information in real-time. In substation automation systems, synchrophasor streams are sent to controllers via local networks. Protecting such streams from threats involving deletion (availability), corruption (integrity) and privacy is crucial. Typically, the synchrophasor reporting latency should be less than 10 ms to prevent cascading faults [13].

Real-time availability faults occur when the time taken to deliver information is longer than the specified latency. Real-time faults may be natural (e.g., critical state information may get dropped or not arrive in time to be processed) or malicious (e.g., an insider may corrupt synchrophasor data). Note that quality of service (QoS) is not a protection mechanism for critical infrastructures: an average delay less than the latency requirement may still result in a real-time failure if the real delay exceeds the requirement.

5.1 Access Control Systems

Access control systems are trust infrastructures that manage access to computer and network resources. Early approaches include the Bell-LaPadula model [2] that enforces confidentiality policies and the Biba model [4] that enforces integrity policies. These are not dynamic and attempts to make them dynamic are not scalable. In role-based access control (RBAC) [22], roles are assigned to access permissions and users are assigned roles. RBAC scales better than the earlier models, but it is not suitable for highly dynamic systems. Extensions such as the temporal RBAC [3] and the generalized temporal RBAC [15] allow the periodic enabling of roles. However, when the numbers of subject and object attributes become large, the number of roles grows exponentially [27].

The attribute-based access control (ABAC) model [27] assigns attributes to subjects and objects. Authorization is defined for subject descriptors consisting of attribute conditions. These may include environment attributes such as time, day and temperature.

ABAC encompasses the functionality of RBAC by treating roles and security labels as attributes. It captures dynamic environmental and temporal attributes. However, in highly dynamic real-time applications the event space that determines the attribute values can get very large, making any attempt to capture real-time availability scenarios not scalable.

5.2 Enforcing Need-to-Get-Now Policies

The communications systems of critical infrastructure are often bandwidth-constrained. As such, it is important to have an access control mechanism that can guarantee real-time availability to high-priority packets. The Internet Engineering Task Force has proposed two networking architectures, differentiated services (DiffServ) [5] and integrated services (IntServ) [6], that statically partition the bandwidth among different classes of traffic to offer quality of service. However, these architectures do not guarantee real-time availability. Protecting critical infrastructure assets from real-time availability threats requires that packets with the highest priority are guaranteed to be delivered. Consequently, the network bandwidth must be: (i) reserved for the highest priority packets and; (ii) sufficient to enable all the highest priority packets to be delivered.

To address this issue, we propose a real-time attribute based access control mechanism that extends the functionality of DiffServ and IntServ. For any subject s, the availability of a resource o at time t_i is determined based on a real-time attribute $attr(x; t_i)$, $x \in \{s, o\}$ with values in a linearly-ordered set $(\mathcal{L}, \succeq)$ of availability labels. $attr(x, t_i)$ is called a priority label when $x = s$ and a congestion label when $x = o$. The availability labels are dynamically determined based on user events, temporal events, context of the requested service and system events.

To enforce real-time availability, the policy is to guarantee that high-priority (critical) packets are forwarded in real-time when there are faults (caused by nature or the adversary). For this purpose, we shall assume that the network infrastructure has sufficient redundancy to guarantee the delivery of high-priority packets at a specified rate. The following packet forwarding protocol is used:

- **Queuing:** Let P be a packet received by an edge router R and Q the queue of P. If $priority(P) \succeq priority(P')$ for some $P' \in Q$ or if $Q = \emptyset$, then put P in Q and drop all packets $P'' \in Q$ with $priority(P) \succ priority(P'')$. Else, drop P. This implies that all packets in Q have the same priority and, by our earlier assumption, no high-priority packet is dropped.

- **Forwarding:** The first packet P in the queue Q of edge router R is forwarded if and only if $priority(P) \succeq congestion(R)$.

This protocol guarantees the delivery of high-priority packets assuming that the specified rate of high-priority packets is not exceeded.

6. Run-Time Attacks

Run-time attacks involve the imposition of unintended behavior on an executing software target. In an ROP attack [21], there is no need to inject new malicious code – code in local libraries may be used.

The ROP attack overwrites the stack with addresses that point to existing code in a library (e.g., the standard C library `libc`). Instead of calling a

function, it sets return addresses on the stack to the middle of instruction sequences that end with a return instruction. After executing the instructions, the return takes the next address from the stack where execution continues and increments the stack pointer. Thus, the stack pointer acts as an instruction pointer and determines the program control flow. Interested readers are referred to [11, 21] for additional details.

These types of attacks use small instruction sequences (e.g., from `libc` instead of entire functions). The adversary must be able to control the stack flow to implement the attack, and there are ways to prevent this [11]. However, there are run-time attacks that exploit other vulnerabilities using minimal footprints. There is, therefore, a need to guarantee during run-time that the execution of a program cannot deviate from the prescribed flow path.

One critical characteristic of the TC model is its reliance on digests of static execution targets (software) as key measurements of platform integrity. In order to materially impact the integrity of the trusted platform, the adversary must attack the platform software without requiring the presence of malicious software that would be detected by TC structures. ROP attacks [21] do not require additional malicious code to be injected into a platform, but assume that the attacker has already diverted control of execution. The most salient way to achieve this initial diversion under our constraints is via a network-based attack. If TPM hosts are successfully exploited by run-time network-based attacks, it is possible to alter their behavior in a manner that is not detected by TC methodologies. However, the TNC architecture enables network endpoints to decide based on shared trusted integrity information whether or not to accept connections. Consequently, devices that implement TNC structures and protocols have the capability to restrict network associations to allow only trusted remote entities, thereby limiting the ability of the adversary to launch network-based attacks.

7. Secure Electricity Grid Communications

We demonstrate the effectiveness of our approach by using it to secure communications in an IEC61850-90-5-compliant substation automation system against real-time availability and run-time integrity threats. The IEC 61850 document [14] provides an advanced object-oriented semantics for information exchange in power system automation applications, SCADA systems, substation automation and distribution automation. The IEC 61850-90-5 extension [13] specifies the use of the IP transport protocol with data encapsulated in IP packets and formulated so that they can be distributed in wide area network (WAN) environments. This allows for low-cost wide area monitoring, protection and control.

IEC 61850 and its extension do not address security issues. In this paper, we show how robustness can be achieved using a TC architecture with TPMs. This involves integrating the substation automation system infrastructure with: (i) a TPM interface with built-in non-migratable trust as described in Section 3; (ii) a Kerberos multicast authentication service [7]; (iii) a real-time attribute-

based access control system as described in Section 5.2; and (iv) cryptographic services for AES and HMAC (SHA2). Protection from run-time attacks is achieved under the assumption that all trusted software is well-designed and has no flaws or other vulnerabilities (Section 4.2). This assumption is reasonable for critical infrastructure assets with minimal operating systems.

7.1 Trusted IEC 61850-90-5 Profiler

SISCO recently released an open-source software package [23] containing a sample profiler that emulates IEC 61850-90-5 systems. The profiler does not support any security services. However, we use the profiler to demonstrate real-time security for our application involving electricity grid communications.

7.2 Experimental Testbed

The experimental testbed comprised seventeen workstations connected via a Cisco Catalyst 3560G series PoE 48 switch and other switches. The workstations were TPM-enabled and ran Ubuntu Linux and Windows. Eight workstations had an Intel Xeon 5120 1.86 GHz x2 CPU with 2 GB memory and the remaining nine had an Intel Xeon E5506 2.13 GHz x4 CPU and 6 GB memory. A dedicated machine running the krb5 API acted as the Kerberos 5 release 1.10 server, which interfaced directly with the Kerberos API.

7.3 Trusted Substation Automation

A substation automation system consists of multiple substations connected via an intranet. To establish a trusted substation automation system, each component in a zone should be TC-compliant. For trusted interoperability, the TNC platform is used (Section 3). This addresses passive attacks as well as active attacks, including insider attacks. The TPM interface prevents authorized components that are compromised from behaving maliciously, in particular, from contributing to a DoS attack. Furthermore, TNC network access control prevents external DoS attacks.

By using a real-time access control service to manage data feeds, real-time availability is guaranteed for high-priority packets provided that the substation automation network has sufficient redundancy to be resilient when only high-priority packets are sent.

AES was used in the counter mode to generate a stream cipher, with successive values of the counter *ctr* encrypted with the shared secret key to generate a keystream. The keystream was bitwise XORed with the payload m of the packet. The resulting string c together with the value of the counter *ctr* comprised the ciphertext.

In the case of decryption, the AES keystream was generated by encrypting *ctr* with the shared secret key. The result was then XORed with c to get m.

Note that when performing encryption and decryption, the keystream can be computed offline. Only the bitwise XOR operation must be performed online.

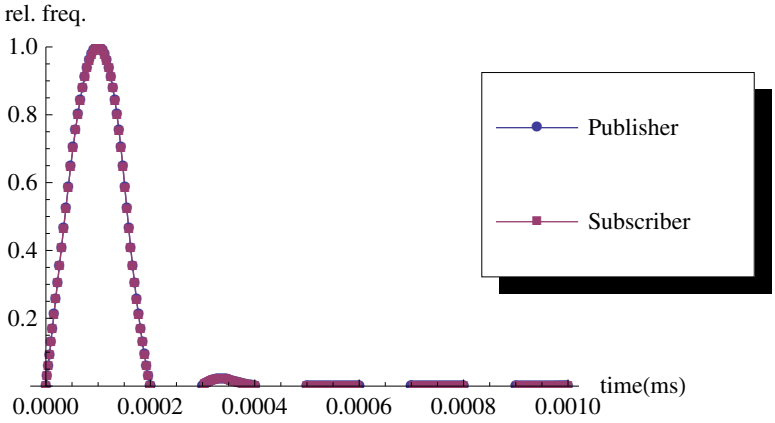

Figure 1. AES counter mode encryption times for publisher and subscriber.

As shown in Figure 1, over the course of 10,000 packet transmissions, the average time needed to generate the keystream at the publisher was 0.0001786 ms with standard deviation $\sigma = 5.37275 \times 10^{-6}$ ms. At the subscriber, the average time was 0.0001133 ms with $\sigma = 0.000229$ ms.

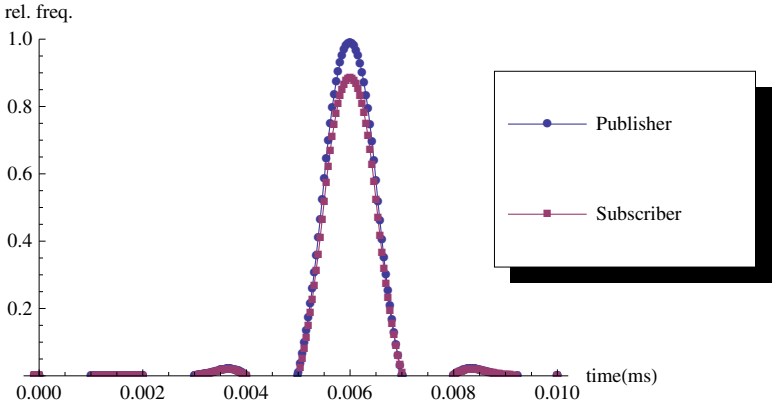

Figure 2. HMAC generation and authentication times for publisher and subscriber.

Authentication was performed by the publisher and subscriber computing HMACs of the ciphertext using SHA2. Over the course of 10,000 packet transmissions the average time at the publisher was 0.006346 ms with $\sigma = 0.001108$ ms; at the subscriber, the average time was 0.007352 ms with $\sigma = 0.002272$ ms (Figure 2).

The average time for generating random numbers was 0.010871 ms with $\sigma = 0.000399$ ms. The average time to lock a TPM key was 0.48788 ms with

$\sigma = 0.021401$ ms, and the average time to unlock a TPM key was 0.36454 ms with $\sigma = 0.051601$ ms.

A publisher is required to build a packet, encrypt the payload, calculate the HMAC and finally transmit the packet. Each subscriber that receives the packet must verify the HMAC and decrypt the payload. Thus, the end-to-end time per packet is $enc+dec+2mac \approx 2mac$ if the keystream is generated offline and the cost of XORing is discounted.

Using the average time estimates, the average end-to-end time per packet is 0.012697 ms. This does not include the time required for transmission, IEC 61850-90-5 header generation, IP protocol formatting and other general processing. We estimate these to be roughly 0.00069 ms for the publisher and 0.54 ms for the subscriber (the subscriber's value includes queuing delays). Thus, the latency per packet is bounded by 1.3 ms, which is less than the 4 ms latency specified by IEC 61850-90-5. This guarantees real-time availability. If a fresh key is required, then an extra 0.36454 ms is needed to unlock the key and 0.48788 ms to lock it. A random number generator may be used to generate the keys offline.

8. Conclusions

A trusted computing architecture can be combined with a real-time access control system to help protect critical infrastructure assets against real-time availability and malware threats. However trusted computing architectures offer only static (load-time) protection, so it is still necessary to address the possibility of run-time (execution) attacks.

The trusted computing methodology presented in this paper mitigates run-time threats to critical infrastructure assets by limiting the openings for exploitation on platform software and detecting run-time compromises, thereby guaranteeing integrity and real-time availability. Experimental results involving the protection of communications of an IEC61850-90-5 compliant substation automation system used in power grids demonstrate the effectiveness of the methodology in critical infrastructure environments.

Acknowledgements

This research was supported by the National Science Foundation under Grant No. 1027217. The authors would also like to thank J. D. Allen, V. Chrissikopoulos, S. Easton, R. van Engelen, D. Guidry, J. Jenkins, P. Kotzanikolaou, J. Lawrence, X. Liu, E. Magkos, S. Ty and X. Yuan for helpful discussions and comments.

References

[1] A. Baratloo, N. Singh and T. Tsai, Transparent run-time defense against stack smashing attacks, *Proceedings of the USENIX Annual Technical Conference*, p. 21, 2000.

[2] E. Bell and L. La Padula, Secure Computer System: Unified Exposition and Multics Interpretation, Technical Report ESD-TR-75-306, MITRE Corporation, Bedford, Massachusetts, 1976.

[3] E. Bertino, P. Bonatti and E. Ferrari, TRBAC: A temporal role-based access control model, *ACM Transactions on Information and System Security*, vol. 4(3), pp. 191–233, 2001.

[4] K. Biba, Integrity Considerations for Secure Computer Systems, Technical Report ESD-TR-76-372, MITRE Corporation, Bedford, Massachusetts, 1977.

[5] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang and W. Weiss, An Architecture for Differentiated Services, RFC 2475, 1998.

[6] R. Braden, D. Clark and S. Shenker, Integrated Services in the Internet Architecture: An Overview, RFC 1633, 1994.

[7] M. Burmester, J. Lawrence, D. Guidry, S. Easton, S. Ty, X. Liu, X. Yuan and J. Jenkins, Towards a secure electricity grid, *Proceedings of the Eighth IEEE International Conference on Intelligent Sensors, Sensor Networks and Information Processing*, 2013.

[8] M. Burmester, E. Magkos and V. Chrissikopoulos, Modeling security in cyber-physical systems, *International Journal of Critical Infrastructure Protection*, vol. 5(3-4), pp. 118–126, 2012.

[9] W. Chang, B. Streiff and C. Lin, Efficient and extensible security enforcement using dynamic data flow analysis, *Proceedings of the Fifteenth ACM Conference on Computer and Communications Security*, pp. 39–50, 2008.

[10] W. Cheng, Q. Zhao, B. Yu and S. Hiroshige, TaintTrace: Efficient flow tracing with dynamic binary rewriting, *Proceedings of the Eleventh IEEE Symposium on Computers and Communications*, pp. 749–754, 2006.

[11] L. Davi, A. Sadeghi and M. Winandy, ROPdefender: A detection tool to defend against return-oriented programming attacks, *Proceedings of the Sixth ACM Symposium on Information, Computer and Communications Security*, pp. 40–51, 2011.

[12] D. Dolev, C. Dwork, O. Waarts and M. Yung, Perfectly secure message transmission, *Journal of the ACM*, vol. 40(1), pp. 17–47, 1993.

[13] International Electrotechnical Commission, IEC/TR 61850-90-5, Edition 1.0 2012-05, Power Systems Management and Associated Information Exchange – Data and Communications Security, Geneva, Switzerland, 2012.

[14] International Electrotechnical Commission, IEC/TR 61850-1, Edition 2.0, Communication Networks and Systems in Substations for Power Utility Automation – Part 1: Introduction and Overview, Geneva, Switzerland, 2013.

[15] J. Joshi, E. Bertino, U. Latif and A. Ghafoor, A generalized temporal role-based access control model, *IEEE Transactions on Knowledge and Data Engineering*, vol. 17(1), pp. 4–23, 2005.

[16] A. Leicher, N. Kuntze and A. Schmidt, Implementation of a trusted ticket system, *Proceedings of the Twenty-Fourth IFIP TC 11 International Information Security Conference*, pp. 152–163, 2009.

[17] M. Lelarge and J. Bolot, Economic incentives to increase security in the Internet: The case for insurance, *Proceedings of the Twenty-Eighth Conference on Computer Communications*, pp. 1494–1502, 2009.

[18] A. Metke and R. Ekl, Smart grid security technology, *Proceedings of the First IEEE PES Conference on Innovative Smart Grid Technologies*, 2010.

[19] V. Mohan and K. Hamlen, Frankenstein: Stitching malware from benign binaries, *Proceedings of the Sixth USENIX Workshop on Offensive Technologies*, p. 8, 2012.

[20] F. Qin, C. Wang, Z. Li, H. Kim, Y. Zhou and Y. Wu, LIFT: A low-overhead practical information flow tracking system for detecting security attacks, *Proceedings of the Thirty-Ninth Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 135–148, 2006.

[21] R. Roemer, E. Buchanan, H. Shacham and S. Savage, Return-oriented programming: Systems, languages and applications, *ACM Transactions on Information and System Security*, vol. 15(1), pp. 2:1–2:34, 2012.

[22] R. Sandhu, E. Coyne, H. Feinstein and C. Youman, Role-based access control models, *IEEE Computer*, vol. 29(2), pp. 38–47, 1996.

[23] SISCO, Cisco and SISCO collaborate on open source synchrophasor framework, Press Release, Sterling Heights, Michigan (`www.sisconet.com/downloads/90-5_Cisco_SISCO.pdf`), 2011.

[24] Trusted Computing Group, TPM Main Specification, Level 2, Version 1.2, Revision 116, Beaverton, Oregon (`www.trustedcomputinggroup.org/resources/tpm_main_specification`), 2011.

[25] Trusted Computing Group, TCG Trusted Network Connect TNC Architecture for Interoperability; Specification 1.5; Revision 3, Beaverton, Oregon (`www.trustedcomputinggroup.org/files/resource_files/2884F884-1A4B-B294- D001FAE2E17EA3EB/TNC_Architecture_v1_5_r3-1.pdf`), 2012.

[26] Trusted Computing Group, Trusted Computing Group, Beaverton, Oregon (`www.trustedcomputinggroup.org`).

[27] E. Yuan and J. Tong, Attribute-based access control (ABAC) for web services, *Proceedings of the IEEE International Conference on Web Services*, pp. 561–569, 2005.