A Trusted Computing Architecture for Critical Infrastructure Protection

Mike Burmester
Department of Computer Science
Florida State University
Tallahassee, FL 32306, USA
Email: burmester@cs.fsu.edu

***Abstract*—Most critical infrastructures can be modeled as cyber-physical systems whose cyber components control underlying physical processes so as to optimize system objectives based on physical properties/constraints and the current and estimated state of the system. Such systems usually require performance guarantees and support for security: wrongly received or missed commands can render the entire system unstable. Yet, securing cyber-physical systems with heterogeneous components is still an open and challenging problem.**

In this paper we propose a trusted computing architecture for critical infrastructure protection based on the trusted computing paradigm. We discuss the threat model, the vulnerabilities, real-time availability, run-time integrity and show how to get resilience against intentional and unintentional faults by using trusted computing enabled components and an access control structure that enforces need-to-get-now (availability) policies. We conclude by showing how this approach can be used to secure substation automation systems of an IEC/TR 61850-90-5-compliant electricity grid.

***Keywords*- cyber-physical systems; critical infrastructures; electricity grid; Universal Composability; IEC/TR 61850-90-5;**

I. Introduction

The Stuxnet worm has shown how vulnerable industrial facilities are to malware attacks. The crucial nature of services provided by critical infrastructure systems and the potentially devastating effects of such attacks highlight the necessity for solutions that provide effective protection from malware threats. However real-time availability is also vitally important. Controllers should have access to state information in real-time: correct messages delivered at the wrong time may lead to erroneous responses and system failure. To mitigate such threats from both intentional and unintentional actors new technologies that augment trust in emerging infrastructure systems are needed. The Trusted Computing (TC) paradigm [1] supports application platforms for secure distributed applications and enforces the integrity of execution targets. This prevents the execution of untrusted software and also addresses insider threats. However TC architectures offer only static (load-time) protection, so one has to address the possibility of run-time attacks that bypass TC protection and build on exploits that subvert machine code execution.

A. *Our contribution*

We describe a Trusted Computing (TC) architecture that combines TC engines with a real-time access control infrastructure for system protection.

The paper is organized as follows. In Section II the protection afforded by TC platforms is discussed. In Section III we define a threat model for TC-compliant critical infrastructures and a security framework that takes into account natural faults, system vulnerabilities, bad actors and captures real-time system behavior. In Section IV we describe an access control system that supports need-to-get-now policies. In Section V we discuss run-time attacks and their impact on TC-compliant systems and consider methods to mitigate such attacks. Ultimately, to thwart run-time attacks, TC-compliant devices will have to operate with a limited enumerated set of functions and, employ real-time dynamic integrity monitoring. Finally, in Section VI we show how our architecture can be used to secure substation automation systems of an IEC/TR 61850-90-5-compliant electricity grid.

B. *Related Work*

Trusted Computing (TC) technologies have been used in various applications to augment trust in distributed systems. In [2] the authors integrate TC components with Kerberos [3] to implement authorization based on attestation. In [4] a timing side-channel is used for verifying TPM proximity, and in [5] a TPM is used to secure direct recording of electronic voting machines by binding voter choices to keys in trusted storage. In [6] a small trusted hardware device, TrInc, based on a trusted counter and a key is used to address equivocation in distributed systems.

There is a lot of work in the literature for protecting systems from run-time threats, yet these remain a major concern. Recent proposals use taint analysis and dynamic integrity monitoring of run-time software behavior to identify and track security violations [7], [8], [9]. Various methods exist to enable tracking, such as code instrumentation, binary re-writing and architectural tracking support. Though effort is made in some papers to optimize the performance of the monitoring process (through identifying unnecessary checks, etc.), there is a non-trivial performance overhead associated with dynamic measurement [7].

II. Trusted Computing for critical infrastructure protection

The Trusted Computing Group (TCG) [1] has published specifications for architectures and interfaces for several computing implementations. Platforms based on these are expected to meet the functional and reliability requirements of computer systems and provide increased assurance of trust. As such, they are well suited to support and protect critical infrastructures. The Trusted Platform Module (TPM) [10] and the Trusted Network Connect (TNC) [11] are two such architectures.

The TPM 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. TPMs employ trusted engines, called *roots of trust*, to establish trust in the expected behavior of the system. Trust is based on an integrity protected boot process in which executable code and associated configuration data is measured before it is executed—this requires that a hash of the BIOS code is stored in a Platform Configuration Register (PCR). For remote attestation the TPM uses an attestation identity key to assert the state of the current software environment to a third party—by signing PCR values. Sealed storage is used to protect cryptographic keys. To encrypt/decrypt/authenticate, keys are released, conditional on the current software state (using current PCR values). The TCG requires that TPMs be physically protected from tampering. This includes binding the TPM to physical parts of the platform (*e.g.* the motherboard).

The TNC is a Trusted Computing (TC) architecture for trusted network applications. What distinguishes TNC from other interoperability architectures is the requirement that the OS configuration of the client and server is checked prior to a communication channel being established. A trusted link between a client and server is established only if:

1) The identity of the client and server is trusted. A Public Key Infrastructure is used to establish trust-links between a Root Authority and the TPMs of the client/server.
2) The client has real-time access to the server.
3) The client and server are authenticated. A root of trust on the TPM of both parties is invoked to release the required keys to execute a handshake protocol [11].
4) The integrity of communicated data, and if necessary the confidentiality, is enforced by the TPM.

III. A threat model and security framework

The TPM prevents compromised components of TC-compliant systems from executing. As a result, if we exclude run-time (execution) threats, malicious (Byzantine) threats are reduced to DoS threats that can be addressed with redundancy.

There are two kinds of faults that may affect a TC-compliant critical infrastructure: natural (this includes accidents) and adversarial (intentional/malicious/insider). Natural faults can be predicted, in the sense that an upper bound on the probability of such faults can be estimated. Redundancy can then be used to reduce this probability to 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 tampering the TPM chip). So there is a cost involved. One way to thwart them is to make the cost high enough to prevent them. There are several security models that use economics and risk analysis based on redundancy [12] that are appropriate for threat models with overt faults. These assume a bound on adversarial resources and an architecture with sufficient redundancy to make such DoS attacks prohibitively expensive.

In our approach for critical infrastructure protection we shall assume an architecture that has sufficient redundancy so that the probability that the system will fail due to natural faults is negligible (*e.g.,* less than 2^{-20}) and, 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 $\mathcal{C}$ consists of:

1) A real-time model that captures its functionality including a faults distribution $\mathcal{F}$ on components [13].
2) A set $\mathcal{S}$ of specifications, policies and security requirements that identifies the vulnerabilities $\mathcal{V}$ that need to be protected.
3) An $\mathcal{S}$-profiler that emulates the behavior of $\mathcal{C}$ in real-time subject to $\mathcal{S}$. The profiler is defined by a software program that runs in real-time.
4) A proof that the $\mathcal{S}$-profiler adheres to the security requirements of $\mathcal{S}$ in real-time in the presence of faults/disasters, accidents and malicious actors.

A. *A security framework for critical infrastructures*

We model a critical infrastructure by a finite state hybrid real-time automaton $\mathcal{C} = (\tau, A, Q, q_0, D, \mathcal{F}, \mathcal{S}, \mathcal{V})$ [13], with $\tau : t_1, t_2 \dots$ a time schedule, A a finite set of actions that includes a special symbol "$\perp$", $Q \neq \emptyset$ a finite set of states that is partitioned into *safe* states Q_s, *critical* states Q_c, and *terminal* states Q_t, $q_0 \in Q_s$ an initial state, $D \subset Q \times Q \times A$ a transition function that is time triggered, $\mathcal{F}$ a distribution on Nature's faults, $\mathcal{S}$ a set of specifications and $\mathcal{V}$ a set of vulnerabilities.

$$D(t_i) : \; q \xrightarrow{t_i, a} q' \;\text{ for } (q, q', a) \in D \text{ and } t_i \in \tau,$$

describes the transition that action a causes at time t_i. 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 of the real-time automaton $\mathcal{C}$ are those specified by $\mathcal{S}$: intelligent electronic devices, operators, the adversary and Nature (the Environment). Nature controls the temporal and location aspects of all events and schedules state transitions in a timely manner according to τ, using the distribution $\mathcal{F}$ to select from among her strategies for component failure, and resolves concurrency issues by linking events to their actual start time.

It is important that protection mechanisms of a critical infrastructure adhere to the time-frame τ: command/control information that arrives late may result in the system transitioning to a critical/terminal state. For our application in Section VI the latency $\delta = t_{i+1} - t_i$ specified by IEC/TR 61850-90-5 is $\sim 1\,ms$.

We use a semantic security threat model extended to capture real-time events. This restricts the adversary to exploiting the vulnerabilities $\mathcal{V}$ specified in $\mathcal{S}$, in particular: those identified by the policies of the system, vulnerability assessments, and gray-box penetration testing [14]. These involve the components of the system such as control systems, embedded systems and communication channels.

B. The good, the bad and the ugly

1) The good: The TPM protects system components from behaving in an unexpected way. In particular, prior to the execution of any trusted program an integrity check of its state (against a stored PCR configuration) is required. Consequently if the program is compromised it will not be executed by the OS.

2) The bad: The TPM allows only trusted software to execute. Therefore the integrity of trusted software (which includes the OS) is a fundamental requirement in order to ensure trust in the computing infrastructure. The system software must be well designed, with no security holes backdoors or vulnerabilities that could be exploited by an adversary. An exploit in the OS may allow the adversary to bypass the protection offered by the TPM. There are several reasons why the design of software programs may be faulty. A major reason is the complexity of the execution environment (the OS and CPU hardware). Another is poor software development practices.

3) The ugly: '*S*ecurity is not necessarily composable' [16]. Proof-carrying code is not closed with respect to composability in general. An interesting example involving routing protocols is discussed in [15], where it is shown that a routing protocol that is secure in isolation is not secure when executed concurrently with itself. Consequently the TPM provides integrity guarantees only at load-time, not run-time.

An exploit of the OS may make it possible for the adversary to change the execution flow of a trusted program. There are several run-time attacks [17] that use metamorphic malware such as the self-camouflaging Frankenstein [18] or more generally, return oriented programming (ROP) [19]. For these the adversary must be able to control the execution flow on the stack, and there are ways to prevent this [20]; however as pointed out earlier, even if there are no exploits, concurrent execution of trusted code (that is not Universally Composable) may lead to untrusted behavior.

IV. Real-time availability threats for critical infrastructures

Real-time availability threats for critical infrastructures exploit the time required to deliver system resources. For example, in an electricity grid controllers must have access to state information in real-time. For substation automation systems, synchrophasor streams are sent to controllers via local networks. Protecting such streams is crucial. Typically the synchrophasor reporting latency should less than $10\,ms$ (to prevent cascading faults [21]). Real-time availability faults occur when the time taken to deliver such 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) or Best Effort mechanisms are not appropriate for real-time threat protection: an average delay less than the latency may still result in real-time failure if the actual delay exceeds the latency.

A. Access control

Access control models are trust infrastructures that manage the resources of computer or network systems. Early models include Bell-LaPadula [22] that enforces need-to-know (confidentiality) policies and Biba [23] that enforces integrity policies. Role Based Access Control (RBAC) [24] supports authorization policies and scales better than previous models. Extensions such as Temporal RBAC [25] and Generalized Temporal RBAC [26] address real-time issues, however they also do not scale well when the number of subject and object attributes becomes large [27]. Attribute-Based Access Control (ABAC) [27] assigns attributes to subjects and objects. Authorization is defined for subject descriptors that may include environment attributes such as *time, day, temperature*, etc. ABAC encompasses the functionality of RBAC by treating roles and security labels as attributes and captures dynamic environmental and temporal attributes.

B. Access control that enforces need-to-get-now policies

The communication system of a critical infrastructure may be bandwidth-constrained. As such, it is important to employ an access control model that can guarantee real-time availability to high priority packets. The Internet Engineering Task Force has proposed two computer networking architectures, the Differentiated Services (DiffServ) [28] and the Integrated Services (IntServ) [29], that statically partition the bandwidth among different classes of traffic for QoS. However these do not *guarantee* real-time availability. For a critical infrastructure, protection from real-time availability threats requires that packets with the highest priority are guaranteed delivery. Consequently the network bandwidth must be: (a) reserved for the highest priority packets and, (b) sufficient so that all such packets are delivered.

To address this issue, we propose a real-time attribute based access control mechanism that extends the functionality of IntServ and DiffServ. For any subject s, availability to 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. The label of $attr(x,t_i)$ is called *priority* when $x = s$ and *congestion* when $x = o$. Availability labels are dynamically determined based on user events, temporal events, the context of the requested service and system events.

To enforce real-time availability, high (critical) priority packets must be 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 delivery of high priority packets at a specified rate when only such packets are transmitted (Section III). The following packet forwarding protocol enforces need-to-get-now policies:

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

Forwarding. The lead (high priority) packet P in queue Q of edge router R is forwarded if and only if, $priority(P) \succeq congestion(R)$.

This protocol guarantees delivery of high priority packets assuming that the specified rate for high priority packets is not exceeded. A packet forwarding protocol that guarantees delivery for high priority packets and offers QoS for all other packets is presented in [30].

V. Run-time attacks

Run-time attacks involve the imposition of unintended behavior on a software target that is executing. In a return oriented programming (ROP) attack [19] there is no need to inject new malicious code: instead code in local libraries may be used. ROP overwrites the stack with addresses that point to existing code. 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 instruction, the return takes the next address from the stack where execution continues, and increments the stack pointer. For more details the reader is referred to [19], [20].

Attacks of this kind use small instruction sequences instead of whole functions. Clearly the adversary must be able to control the stack flow for such an attack, and there are ways to prevent this [20]. However there may be run-time attacks, that exploit other vulnerabilities using a minimal footprint, or that exploit concurrency. There is therefore a need to guarantee that during run-time the execution of a program cannot deviate from the prescribed path.

A. Run-time threats for TC-compliant systems

The TC model relies on digests of static execution targets (software) as key integrity measurements. In order to materially impact the integrity of a TPM, the adversary must attack the platform software without requiring the presence of malicious software that would be detected by the TC structure. ROP attacks [19] do not require additional malicious code to be injected onto the platform, but assume that the attacker can divert control of execution. One way to achieve this is with a network-based attack. If TPM hosts are successfully exploited by such attacks, it will be possible to alter their behavior in a manner that is not detected by TC methodologies. However TNC architectures only enable trusted network connections, thereby limiting the adversary's ability to launch remote network-based attacks.

To mitigate run-time threats for critical infrastructures, any successful approach will have to:

Restrict exploits on platform software. To reduce the attack surface, devices will need to be constrained on functionality and resource usage, and operate with a structured, well-defined, enumerated set of functions.

Employ methods that detect run-time compromise. There is substantial work in the literature on techniques for dynamically detecting software faults [7], [8], [9]. Although most techniques carry significant computational overhead, critical infrastructure systems can justifiably be expected to bear the requisite computational resources. Dynamic integrity monitoring is a technique that shows promise for protecting infrastructures from run-time failures. However, the time required to address such threats may increase the likelihood of real-time availability failure.

VI. Secure communication for the electricity grid

To demonstrate the effectiveness of our approach we show how it can be used to protect the communication of an IEC61850-90-5-compliant substation automation system (SAS) of an electricity grid against real-time availability and run-time integrity threats. IEC/TR 61850 [31] is a Technical Report of the International Electrotechnical Commission that offers advanced object oriented semantics for information exchange in power system automation applications, SCADA, and system protection. IEC/TR 61850-90-5 [21] extends it by specifying the use of the IP transport protocol so that data can be distributed to wide area network environments for low cost monitoring, protection and control. Both IEC/TR 61850 and its extension do not address security issues.

For SAS communication robustness we use a TC architecture that integrates: a TPM with built-in non-migratable trust (Section II), a Kerberos multicast authentication service [30], a real-time attribute-based access control system (Section IV-B) and cryptographic services for AES and HMAC (SHA2). Protection against run-time attacks is assured if all trusted software is well designed with no flaws (Section III-B1). This assumption is reasonable for critical infrastructures with a minimal OS.

A. A trusted IEC/TR 61850-90-5 profiler

SISCO recently released an open source software package [32] with a sample profiler that emulates IEC/TR 61850-90-5-compliant systems. This profiler does not support any security services. To capture real-time and run-time security we extended the profiler as outlined above.

B. The testbed

A testbed with 17 TPM-enabled workstations connected via a CISCO Catalyst 3560G series PoE48 switch was established. The machines ran Ubuntu Linux and Windows with an Intel Xeon E5506 2.13GHz x 4 CPU and 6GB memory. A dedicated machine running the krb5 API acted as a Kerberos 5 release 1.10 server, interfacing directly with the Kerberos API.

C. Trusted substation automation systems

Trusted Computing for SAS is established by using TPM and TNC interfaces (Section II) and a real-time access control service to manage data feeds (Section IV). This prevents trusted components that get compromised from behaving abnormally, and guarantees real-time availability for high priority packets, provided the SAS network has sufficient redundancy (Section IV-B). The following experiments were conducted.

Encryption/Decryption. AES was used in counter mode to generate a keystream with a shared secret key. The keystream was bitwise XORed with the payload of the packet. The ciphertext was the resulting string together with the value of the counter. For decryption, the same keystream is generated and XORed with the ciphertext. Note that the keystream can be computed offline, with only the bitwise XOR done online. Over a course of 500 packet transmissions the average time required for generating the keystream was: at the producer $0.02175\,ms$ with standard deviation $\sigma = 0.00457\,ms$, and at the subscriber $0.02413\,ms$ with $\sigma = 0.00420\,ms$.

Authentication. Both producers and subscribers computed an HMAC of the ciphertext using SHA2. Over a course of 500 packet transmissions the average time was: at the producer $0.03063\,ms$ with $\sigma = 0.00178\,ms$, and at the subscriber $0.03786\,ms$ with $\sigma = 0.00167\,ms$.

A producer must build the packet, encrypt the payload, calculate an HMAC and transmit the packet. Each subscriber that receives the packet must verify the HMAC and decrypt the payload. The end-to-end time per packet is: $enc + dec + 2mac \approx 2mac$, if the keystream is generated offline. Using the earlier average estimates we get $0.06849\,ms$. This does not include transmission time, IEC/TR 61850-90-5 header generation, IP protocol formatting and other general processing requirements. These are roughly $0.00069\,ms$ for the producer and $0.54\,ms$ for the subscriber (the subscriber's value includes queuing delays). So the latency per packet is bounded by $1.3\,ms$, which is less then the $4\,ms$ latency specified by IEC/TR 61850-90-5. For more details the reader is referred to [33].

Acknowledgment

This material is based upon work supported by the National Science Foundation Grant No. 1027217. The author 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, comments and contributions.

References

[1] TCG, *http://www.trustedcomputinggroup.org/*

[2] A. Leicher, N. Kuntze, and A. U. Schmidt, "Implementation of a trusted ticket system," *IFIP Advances in Information and Communication Technology*, vol. 297, pp. 152–163, 2009.

[3] C. Neuman, T. Yu, Hartman, and K. Raeburn, "The Kerberos Network Authentication Service (V5)," *RFC 4120*, July 2005.

[4] R. A. Fink, A. T. Sherman, A. O. Mitchell, and D. C. Challener, "Catching the Cuckoo: Verifying TPM Proximity Using a Quote Timing Side-Channel," in *TRUST*, ser. Lecture Notes in Computer Science, J. M. McCune, B. Balacheff, A. Perrig, A.-R. Sadeghi, A. Sasse, and Y. Beres, Eds., vol. 6740. Springer, 2011, pp. 294–301.

[5] R. A. Fink, A. T. Sherman, and R. Carback, "TPM meets DRE: reducing the trust base for electronic voting using trusted platform modules," *IEEE Transactions on Information Forensics and Security*, vol. 4, no. 4, pp. 628–637, 2009.

[6] D. Levin, J. R. Douceur, J. R. Lorch, and T. Moscibroda, "TrInc: small trusted hardware for large distributed systems," in *NSDI'09: Proc. 6th USENIX Symp. Networked Systems Design and Implementation*. Berkeley, CA, USA: USENIX Association, 2009, pp. 1–14.

[7] W. Chang, B. Streiff, and C. Lin, "Efficient and extensible security enforcement using dynamic data flow analysis," in *ACM Conf. Comp. and Comm. Security*, 2008, pp. 39–50.

[8] W. Cheng, Q. Zhao, B. Yu, and S. Hiroshige, "TaintTrace: Efficient flow tracing with dynamic binary rewriting," *Proc. 11th IEEE Symp. Computers and Communications (ISCC '06)*, 2006, pp. 749 – 754.

[9] F. Qin, C. Wang, Z. Li, H. seop Kim, Y. Zhou, and Y. Wu, “Lift: A low-overhead practical information flow tracking system for detecting security attacks,” in *39th IEEE/ACM Int. Symp. Microarchitecture, (MICRO-39)*, 2006, pp. 135–148.

[10] TCG, “TPM Structures, Level 2, Ver. 1.2, Rev. 116, Communication networks and systems for power utility automation,” March 2011.

[11] ——, “Trusted Network Connect Architecture for Interoperability; Specification 1.3; Rev. 6,” April 2008.

[12] M. Lelarge and J. Bolot, “Economic incentives to increase security in the Internet: The case for insurance,” *INFOCOM 2009*, pp. 1494–1502, 2009.

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

[14] DHS and CPNI, “Cyber Security Assessments of Industrial Control Systems, *Control Systems Security Program, National Cyber Security Division*”, November, 2010.

[15] M. Burmester and B. de Medeiros, “On the Security of Route Discovery in MANETs”. IEEE Transactions on Mobile Computing, 8(9): 1180-1188, 2009.

[16] R. Canetti, “Universally composable security: A new paradigm for cryptographic protocols”, in *Proc. IEEE Symp. Found. of Comp. Science (FOCS01)*, pp. 136-145, 2001.

[17] A. Baratloo, N. Singh, and T. Tsai, “Transparent run-time defense against stack smashing attacks”, in *Proc. USENIX Annual Technical Conference*, ser. ATEC ’00. Berkeley, CA, USA: USENIX Association, pp. 21–21, 2000.

[18] V. Mohan and K. W. Hamlen, “Frankenstein: Stitching malware from benign binaries,” in *WOOT*, E. Bursztein and T. Dullien, Eds. USENIX Association, 2012, pp. 77–84.

[19] R. Roemer, E. Buchanan, H. Shacham, and S. Savage, “Return-Oriented Programming: Systems, Languages, and Applications,” *ACM Trans. Inf. Syst. Secur.*, vol. 15, no. 1, pp. 2:1–2:34, Mar. 2012.

[20] Davi, Lucas and Sadeghi, Ahmad-Reza and Winandy, Marcel, “Ropdefender: a detection tool to defend against return-oriented programming attacks,” *Proc. 6th ACM Symp. Inform. Computer and Communications Security*, ASIACCS ’11. New York, NY, USA: ACM, 2011, pp. 40–51.

[21] IEC, “IEC/TR 61850-90-5, Edition 1.0 2012-05, Technical Report, Power systems management and associated information exchange – Data and communications security,” May 2012, *http://webstore.iec.ch/*

[22] E. D. Bell and J. L. La Padula, “Secure computer system: Unified exposition and Multics interpretation,” Bedford, MA, 1976.

[23] K. J. Biba, “*Integrity considerations for secure computer systems*,” MITRE Corp., Tech. Rep., 1977.

[24] R. S. Sandhu, E. J. Coyne, and C. E. Feinstein, H. L.and Youman, “Role-Based Access Control Models,” *IEEE Computer*, vol. 29, no. 2, pp. 38–47, 1996.

[25] E. Bertino, P. A. Bonatti, and E. Ferrari, “TRBAC: A temporal role-based access control model,” *ACM Trans. Inf. and Syst. Security (TISSEC)*, vol. 4, no. 3, pp. 191–233, 2001.

[26] J. Joshi, E. Bertino, U. Latif, and A. Ghafoor, “A generalized Temporal Role-Based Access Control model,” IEEE Trans. Knowledge and Data Engineering, vol. 17, pp. 4–23, 2005.

[27] E. Yuan and J. Tong, “Attribute based access control (ABAC) for Web services,” in *Proc. 2005 IEEE Int. Conf. on Web Services* (ICWS 2005), IEEE, 2005, pp. 561–569.

[28] S. Blake, D. Black, D. Carlson, M., Z. E., Wang, and W. Weiss, “An Architecture for Differentiated Services”, *RFC 2475*, Dec. 1998.

[29] S. Shenker, R. Braden, and D. Clark, “Integrated Services in the Internet Architecture: An Overview,” *IETF Request for Comments (RFC)*, vol. 1633, 1994.

[30] Burmester, M., Laurence, J., Guidry, D., Easton, S., Ty, S., Liu, X., Yan, X., & Jenkins, J. “Towards a Secure Electricity Grid,” in *Proc. IEEE 8th Inter. Conf. Intell. Sensors, Sensor Networks and Inf. Processing (ISSNIP 2013)*, Melbourne, Australia, 2013.

[31] IEC/TR 61850-1, 1st Ed., 2003-04, TR Communication networks and systems substations, 2003, *http://webstore.iec.ch/*

[32] SISCO, “CISCO and SISCO Collaborate on Open Source Synchrophasor Framework, Press Release,” 2011. *http://www.sisconet.com/downloads/90-5_Cisco_SISCO.pdf*

[33] D. Guidry, M. Burmester, X. Yuan, X. Liu, J. Jenkins, and S. Easton, “Techniques for Securing Substation Automation Systems,” in *Proc. 7th Inter. Workshop Crit. Inform. Infrastr. Security, CRITIS (2012)*, Lillehammer, Norway, 2012.