Toward Survivable Intrusion-Tolerant Open-Source SCADA

Thomas Tantillo
Department of Computer Science at Johns Hopkins University
tantillo@cs.jhu.edu

***Abstract*—As vital components of critical infrastructure, SCADA systems must continue to operate correctly and at their expected level of performance at all times. However, current SCADA systems are vulnerable to intrusions, and even a single compromise can cause catastrophic consequences. We present the architecture of and initial steps toward the first intrusion-tolerant open-source SCADA system that is survivable over the required long system lifetimes. We perform a case study of a hypothetical deployment on the Eastern Seaboard of the United States to validate that our SCADA system will meet SCADA application latency requirements.**

I. Introduction

Supervisory Control and Data Acquisition (SCADA) systems form the backbone of critical infrastructure, including the power grid and water supply systems. As vital components of critical infrastructure, SCADA systems must operate correctly and at their expected level of performance at all times. Modern SCADA systems employ fault tolerance techniques to overcome standard faults, such as hardware crashes, but cannot survive intrusions that compromise part of the system. Compromised SCADA system components can send malicious information to system administrators or to remote equipment, with catastrophic consequences such as physical equipment damage, blackouts, and significant economic loss.

The intrusion tolerance gap exhibited by SCADA systems stems from the fact that security against cyber attacks was not a primary focus. Many SCADA systems were designed to run over private networks and were assumed to be protected by an "air gap." Today, SCADA systems are migrating to Internet-connected IP networks for cost-effectiveness and scalability, but this transition makes the systems more accessible to attack. Compounding the problem, SCADA systems are high-value targets, can be in service for a decade or longer, and may run legacy components that lack the latest security patches. Under such conditions, even a well-protected SCADA system is likely to suffer compromises over its lifetime.

Attempts at developing proprietary intrusion-tolerant SCADA systems have not gained traction because of a lack of incentive. Power service is a heavily-regulated industry where the power companies closely follow the regulation requirements and have little incentive to go beyond them. In addition, SCADA system manufacturers have little incentive to implement capabilities that are not demanded by power companies. Intrusion tolerance is not currently on the regulators' checklists and therefore there is no push to develop it from the power companies or the SCADA manufacturers, and regulators may even be unaware that the problem is solvable.

In our view, the work in [6] represents the state of the art in addressing the intrusion tolerance gap. That work added intrusion tolerance to a Siemens proprietary SCADA product, creating a prototype that was able to operate correctly and at its expected level of performance in the presence of a successful intrusion. This was accomplished by running several replicas of the SCADA Master in parallel and synchronizing their state using Prime [1], an intrusion-tolerant replication engine that guarantees performance under attack. However, that approach is insufficient: the replicas are simply identical copies of one another, and an attacker can reuse a single exploit to compromise all replicas. Moreover, even with sufficiently diverse replicas, given long SCADA system lifetimes, an attacker will eventually be able to compromise enough replicas to violate safety. Additionally, SCADA system components depend on timely communication to operate correctly, which can be disrupted by an intrusion in the underlying network. Adding to the technical limitations, the prototype was proprietary and was not broadly visible to regulators and power companies.

In this work, we describe our preliminary work constructing the first survivable intrusion-tolerant open-source SCADA system — one that continues operating correctly and at its expected level of performance throughout its intended long lifetime, even in the presence of successful intrusions by malicious attackers. We choose pvbrowser [8] as our base SCADA system because it is an open-source solution that is used in real-world deployments for power distribution and industrial control, providing an easier path towards adoption in practice. We use Prime [1] as the intrusion-tolerant replication engine because it provides performance guarantees while under attack. In order to make Prime and the SCADA system survivable over the system lifetime, we employ software diversity and proactive recovery, obtaining a defense across space and time [7], making it harder for an attacker to control many replicas simultaneously. An attacker must compromise a sufficient number of diverse replicas within a short period of time to succeed. Finally, we use Spines [5] to create an intrusion-tolerant overlay network that provides timely and reliable communication between the SCADA system components.

Our goal is to release our survivable intrusion-tolerant SCADA system as open source. This can impact the entire ecosystem by making SCADA manufacturers, power companies, and regulators aware of the need for intrusion tolerance and the existence of solutions that they can learn from. In due time, this may change regulatory requirements, and our ideas (or some variations) may be implemented by SCADA manufacturers in their own systems. Also, by working with the open source SCADA community, we can have a direct impact by making pvbrowser, which is used in real-world deployments for the power grid, survivable and intrusion tolerant.

II. System Components

Our survivable intrusion-tolerant SCADA system builds on several components: current SCADA systems, an intrusion-

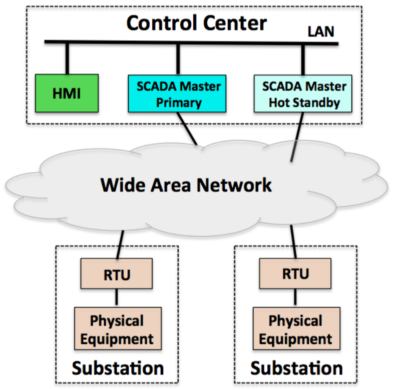

Fig. 1. Typical SCADA System Architecture

tolerant replication engine that guarantees performance under attack, a defense across space and time to support long system lifetimes, and an underlying intrusion-tolerant network that respects SCADA application latency constraints.

A. *Current SCADA Systems*

The architecture of current SCADA systems is shown in Figure 1. **Remote Terminal Units (RTUs)**, located in substations, connect to devices within those substations and translate signals (e.g., current, phase, voltage) into digital data that is sent to the control center over a wide-area network. RTUs control the devices and, for many activities, are capable of autonomous local operation. A **SCADA Master**, located in the control center, periodically polls the RTUs to obtain their current status using industrial communication protocols. The SCADA Master compares the collected values to pre-determined thresholds and raises an alarm if a threshold is crossed. The SCADA Master also issues supervisory control commands to the RTUs (e.g., to change the value of a set point). Control and monitoring are often implemented by two separate processes (i.e., a supervisory control process and a data acquisition process). A **Human-Machine Interface (HMI)** queries the SCADA Master and presents the current status of the infrastructure to the operator, in graphical form, enabling the operator to take action.

Modern SCADA systems employ fault tolerance techniques to overcome benign faults. The SCADA Master is often deployed in a hot-standby configuration, where the primary processes the collected data and communicates with the HMI workstation and RTUs, while a backup executes exactly the same operations but has its output suppressed. The backup is responsible for taking over if the primary fails. Unfortunately, the hot-standby approach is ineffective against malicious intrusions: a malicious primary may compromise the operation of the system but still appear to the backup as functioning properly, preventing the backup from assuming responsibility.

SCADA Performance Requirements. SCADA applications have stringent timeliness requirements of 100 to 200 milliseconds [3]. It is vital to ensure system correctness and timeliness during both normal-case operation and while under attack.

B. *Prime: Correct and Timely Operation Under Attack*

Prime is an intrusion-tolerant state machine replication engine that remains correct and makes progress despite the Byzantine (arbitrary) behavior of a subset of the participants (no more than f out of $3f$+1). Unlike many intrusion-tolerant state machine replication protocols that only guarantee liveness (eventual progress), Prime guarantees good performance even while the system is under attack. Prime bounds the delay that can be introduced by compromised replicas by strictly monitoring their performance against a dynamic threshold derived from measurements of network round-trip times. This strong latency guarantee makes Prime an excellent fit for time-sensitive systems like SCADA. Some other intrusion-tolerant protocols ensure good *average* performance under attack [2], [9], but do not offer latency guarantees for individual operations.

C. *Defense Across Space and Time*

Despite its strong correctness and performance guarantees, Prime alone is not sufficient to build a survivable SCADA system. If the replicas are simply identical copies of one another, then an attacker can reuse a single exploit to compromise all replicas. Even if replicas are sufficiently diverse, given enough time, an attacker will eventually be able to compromise more than f replicas, violating Prime's requirements.

Software diversity and proactive recovery, which together provide a defense across space and time, make Prime survivable over long system lifetimes [7]. For defense across space, we use the MultiCompiler [4], which uses techniques such as stack padding, no-op insertion, equivalent instruction substitution, and function reordering to obfuscate the code layout of an application. For defense across time, replicas are periodically rejuvenated in round-robin fashion, forcing the attacker to compromise f+1 replicas in a short amount of time to succeed. Upon rejuvenation, each replica generates a new cryptographic key and a unique diverse variant, retrieves the correct state from the other replicas, and rejoins the system. To tolerate f compromises and k rejuvenations simultaneously, Prime requires a total of $3f$+$2k$+1 replicas [10].

D. *Spines: Timely Intrusion-Tolerant Overlay Network*

Spines [5] is an overlay messaging toolkit that provides intrusion-tolerant messaging while guaranteeing well-defined delivery semantics. Spines supports both timely messaging and reliable messaging, which are appropriate for SCADA monitoring and control messages respectively.

III. System Architecture

The survivable intrusion-tolerant SCADA system architecture is shown in Figure 2. Rather than use a hot-standby approach, the architecture deploys replicas of the SCADA Master that use Prime for synchronization. The system runs a total of N=$3f$+$2k$+1 replicas to tolerate the simultaneous compromise of f replicas and the simultaneous rejuvenation of k replicas. For protection against benign failures at a single physical location, replicas are placed in geographically dispersed data centers. All communication in the SCADA system uses Spines. Although the replicas are semantically equivalent, their code is diversified (represented by different colors), forcing an attacker to craft variant-specific attacks. Replicas are periodically rejuvenated in round-robin fashion to a clean state to remove potential intrusions. After rejuvenation, a new variant is automatically generated using the MultiCompiler,

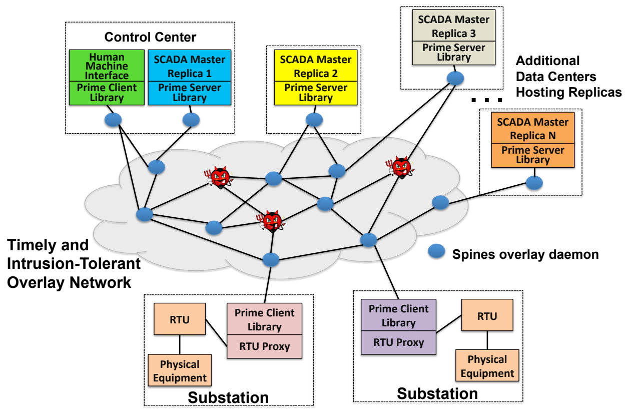

Fig. 2. Survivable Intrusion-Tolerant SCADA System Architecture

which with very high probability is diverse from all past, current and future variants, meaning that a previously successful attack against an old variant will not be successful against this variant with high probability. In addition, that replica generates a new cryptographic key, invalidating its previous key that may be controlled by an attacker. Finally, that replica validates the persistent application state with other replicas, and acquires a clean copy from other replicas if necessary.

We design the architecture to support the latency needs of SCADA systems. We target one-way latencies of no more than 100 milliseconds during normal operation and no more than 200 milliseconds while under attack. We validate this using a case study (Section IV-A).

A. *Extending Prime to Support SCADA Systems*

Server-Driven Polling. Currently, Prime assumes a *client-driven* mode of operation, where clients submit operations to the replicated service. Although a subset of SCADA communication follows this pattern (e.g., the HMI submits a query to the SCADA Master), the majority of communication is *server-driven*, where the SCADA Master periodically polls the RTUs for their latest status in response to a reoccurring timeout. The challenge of server-driven operation is to ensure that the SCADA Master replicas generate identical polling request messages. Since replicas are on different physical machines, clocks may not be perfectly synchronized, and responding to a local timeout may lead to state divergence. Similar to techniques presented in proprietary research [6], we plan to extend Prime to support *logical timeouts*, which expire at an agreed-upon logical time at all correct replicas.

Prime Client Library. Non-replicated applications (e.g., the HMI) that wish to interact with a Prime-replicated application cannot simply send an update to or receive a message from a single replica, since that replica may be compromised. Instead, they must send updates to at least f+k+1 replicas and need to wait for f+1 identical copies of a message, from different replicas, to ensure that message is valid.[1] Rather than requiring each application to support this functionality, we are implementing the *Prime client library*. The client library forwards each application message to f+k+1 replicas and only passes received messages to the application once it has collected identical copies of that message from at least f+1 different replicas. Note that the client library will discard unauthenticated messages and filter out well-formed but invalid ones. In the survivable intrusion-tolerant SCADA system, the HMI and RTU Proxy use the Prime client library for communication with the replicated SCADA Master.

[1]Since at most f replicas may be compromised, the delivered message is guaranteed to be valid because at least one of the f+1 replicas is correct.

B. *Extending Current SCADA Systems*

RTU Proxy. Since large SCADA systems may have hundreds or even thousands of RTUs, replicating RTUs would require a substantial additional hardware investment. Moreover, many legacy RTUs deployed today lack sufficient computational power to generate and verify, in a timely manner, the digital signatures used by Prime. As a result, rather than replicating each RTU, we plan on implementing a generic RTU proxy that uses the Prime client library to enable unmodified RTUs to communicate with the replicated SCADA Master and to ensure that RTUs only receive legitimate poll requests and control commands. Without the RTU proxy, it is impossible to make this guarantee, since RTUs act on any message they receive from SCADA Masters and have no voting mechanism to validate commands. Note that the RTU proxy does not provide intrusion tolerance against compromised RTUs, but the damage a compromised RTU may cause is more localized than a SCADA Master.

IV. Preliminary Results

A. *Latency Case Study: Eastern Seaboard Power Grid*

To validate that our survivable intrusion-tolerant SCADA system will meet SCADA application latency constraints, we analyze the latency that would be experienced by updates in our SCADA system. Specifically, we looked at U.S. power grid deployments, selected the deployment with the largest geographic diameter (approximately 300 miles), and imposed it on the Eastern Seaboard, where we have access to several data centers. Based on our experience designing and deploying overlay networks on the wide area network, this diameter equates to approximately 5 milliseconds of latency. We validated this estimation with measurements between three of the data centers that we have access to on the Eastern Seaboard.

Given these approximations, communication between any two SCADA Master replicas, which are co-located with Spines overlay nodes in data centers, is about 5 milliseconds. In addition, each ordering operation in Prime, which involves several rounds of communication, experiences a total latency of approximately 30 milliseconds. Unlike the Prime replicas, substations (i.e., RTUs) are not located in data centers and must connect to data centers using somewhat slower connections, which we estimate will result in approximately 7 milliseconds latency between a substation and its nearest data center.

The one-way path of an update originating from an RTU proxy in a substation must reach the nearest data center, cross the entire diameter of the overlay network (in the worst case), and must undergo at most two Prime orderings among the SCADA Master replicas — one ordering to agree upon the data reported from the RTU proxy and one to agree upon the resulting action, if any. The total latency experienced for such an update is $7 + 5 + 30 + 30 = 72$ milliseconds. This estimate does not include time spent to perform cryptographic operations, such as digital signatures and digest computations,

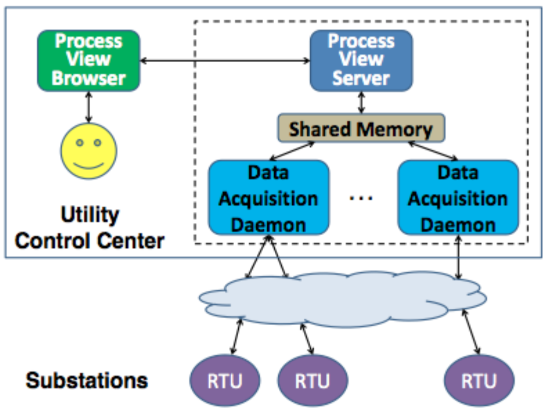

Fig. 3. pvbrowser Architecture

but the remaining 28 milliseconds of the 100 millisecond budget are sufficient to perform these operations.

In the case of replica compromises, the one-way latency experienced by the update can be increased by the additional time to perform a view change to replace a compromised leader. In a system of N=6 replicas (f=1, k=1), at most two view changes must be performed during the lifetime of the update to elect a non-compromised and non-rejuvenating replica as leader. Since the number of rounds of communication in a view change is comparable to that in orderings, each view change results in approximately 30 milliseconds of additional latency. Even with this additional latency, the total expected latency of the update in the presence of replica compromises is still within the target 200 millisecond budget.

B. *Implementation Using pvbrowser*

We conducted a survey of available open-source SCADA systems to identify which one to use as a base. The metrics used for evaluation included project maturity, availability of RTU protocol drivers, and use in real-world systems. We discovered a few candidates, but ultimately decided on pvbrowser because of its simplicity, good documentation, large support for RTU communication protocols, and experience in real-world deployments, where it has been used in a power distribution system covering approximately 10,000 square kilometers and monitoring more than 50 power switches.

The architecture of pvbrowser is depicted in Figure 3. The SCADA Master is implemented by two processes: the process view server, which exercises supervisory control, and the data acquisition daemon, which communicates with RTUs. A master server can run many data acquisition daemons at the same time, each one communicating with RTUs. The process view server and the data acquisition daemon reside on the same machine and communicate using shared memory. The process view browser is an HMI workstation that communicates with the process view server by exchanging ASCII text over TCP/IP. The process view server sends the browser the current status of RTUs (containing widgets, graphs, and other objects) to display. The operator sends commands to the process view server via the browser, which are then passed to the data acquisition daemon to be sent to RTUs.

Toward Survivable Intrusion-Tolerant pvbrowser. To replicate the SCADA Master of pvbrowswer, we must replicate the pvbrowser process view server and data acquisition daemon processes using Prime. To ensure that replicas of the data acquisition daemon generate identical poll requests, they will use the logical timeout protocol (Section III-A). The process view browser and the RTUs will not be replicated but will communicate with proxy processes that use the Prime client library to validate the data sent by the process view server and data acquisition daemon replicas, respectively. This approach ensures that the process view browser only processes valid data and the RTUs only process valid commands and poll requests. We will deploy a Spines overlay network between the data acquisition daemon and the RTUs to ensure that critical control and monitoring messages are delivered even if part of the network is compromised.

Replicating pvbrowser and making it intrusion-tolerant presents several challenges, requiring extensive changes to the architecture. Currently, pvbrowser does not support replication and sources of nondeterminism must be located and removed in order to support the deterministic requirements of the state machine replication approach. One particular source of nondeterminism is the shared memory that is used for communication between the process view server and data acquisition daemon (the relative timing of reads and writes may be different across replicas). To date, we have replaced the shared memory with sockets for message passing, enabling an initial replication of pvbrowser.

Acknowledgement

I thank Marco Platania, Amy Babay, Jonathan Kirsch, Brian Coan, and Daniel Obenshain for their collaboration, as well as Yair Amir for his continual guidance. This work was supported in part by DARPA grant N660001-1-2-4014. Its contents are solely the responsibility of the authors and do not represent the official view of DARPA or the Department of Defense.

References

[1] Y. Amir, B. Coan, J. Kirsch, and J. Lane. Byzantine replication under attack. In *Dependable Systems and Networks With FTCS and DCC, 2008. DSN 2008. IEEE International Conference on*, pages 197–206, June 2008.

[2] A. Clement, E. Wong, L. Alvisi, M. Dahlin, and M. Marchetti. Making Byzantine fault tolerant systems tolerate Byzantine faults. In *Proc. NSDI*, NSDI'09, pages 153–168. USENIX Association, 2009.

[3] J. Deshpande, A. Locke, and M. Madden. Smart choices for the smart grid. 2011. Alcatel-Lucent Technolgy White Paper.

[4] A. Homescu, S. Neisius, P. Larsen, S. Brunthaler, and M. Franz. Profile-guided automated software diversity. In *Code Generation and Optimization (CGO), 2013 IEEE/ACM International Symposium on*, pages 1–11. IEEE, 2013.

[5] Johns Hopkins University Distributed Systems and Newtorks Lab. Spines overlay messaging system. www.spines.org. Accessed: 2015-04-08.

[6] J. Kirsch, S. Goose, Y. Amir, D. Wei, and P. Skare. Survivable SCADA via intrusion-tolerant replication. *Smart Grid, IEEE Transactions on*, 5(1):60–70, Jan 2014.

[7] M. Platania, D. Obenshain, T. Tantillo, R. Sharma, and Y. Amir. Towards a practical survivable intrusion tolerant replication system. In *SRDS 2014*, pages 242–252. IEEE, 2014.

[8] pvbrowser. Simple process visualization. http://pvbrowser.de/pvbrowser/index.php. Accessed: 2015-04-08.

[9] G. Santos Veronese, M. Correia, A. Bessani, and L. C. Lung. Spin one's wheels? Byzantine fault tolerance with a spinning primary. In *Reliable Distributed Systems, 2009. SRDS '09. 28th IEEE International Symposium on*, pages 135–144, Sept 2009.

[10] P. Sousa, A. N. Bessani, M. Correia, N. F. Neves, and P. Verissimo. Highly available intrusion-tolerant services with proactive-reactive recovery. *IEEE Trans. Parallel Distrib. Syst.*, 21(4):452–465, Apr. 2010.