Toward Survivable SCADA

Jonathan Kirsch
Siemens Technology to Business Center
1995 University Avenue
Berkeley, CA
jonathan.kirsch@siemens.com

Stuart Goose
Siemens Technology to Business Center
1995 University Avenue
Berkeley, CA
stuart.goose@siemens.com

Yair Amir*
The Johns Hopkins University
3400 N. Charles Street
Baltimore, MD
yairamir@cs.jhu.edu

Paul Skare
Pacific Northwest National Laboratory
Richland, WA
paul.skare@pnnl.gov

ABSTRACT

This paper reports on our experience designing and implementing the first *survivable* SCADA system – one capable of providing correct behavior with minimal performance degradation even during a cyber attack that compromises part of the system. We describe the challenges we faced while attempting to integrate modern intrusion-tolerant protocols with the SCADA architecture and present the techniques we developed to overcome these challenges.

Categories and Subject Descriptors

D.4.5 [**Operating Systems**]: Reliability—*Fault tolerance*

General Terms

Security, Reliability

Keywords

Survivability, SCADA, High Availability

1. INTRODUCTION

Critical infrastructure systems provide vital services such as electricity generation and distribution, water treatment, and traffic control. These systems are often geographically distributed and rely heavily on communication networks for control and data (e.g., SCADA). Although many of these systems were designed to operate on isolated, private networks, this assumption no longer holds as these systems migrate to modern IP-based deployments. Historically, society has come to expect critical services to be continually available, but their growing reliance upon networks makes them increasingly vulnerable to malicious attacks.

Cyber attacks, especially insider attacks, are on the rise,[1] and the cost of service disruption is severe, especially for critical infrastructure systems.[2] As in the corporate world, critical infrastructure systems deploy maintstream IT security products, such as firewalls and antivirus checkers, to protect against external threats, but it is impossible to prevent all attacks. Moreover, insider attacks have already effectively breached the system's security perimeter and leave the control system exposed to compromise. The unfortunate reality is that state-of-the-art security technologies provide insufficient protection against contemporary and emerging threats.

The field of *intrusion tolerance* has been motivated by the key question: Is it possible for a critical application to act as its own firewall, enabling it to *survive* even in the face of malicious attacks that result in a partial system compromise? By survive, we mean that the application not only continues to operate correctly, but that it also operates with minimal performance degradation. These twin properties are essential for maintaining the high availability of critical infrastructure systems.

Over the last decade, using intrusion-tolerant protocols to achieve consistent global state (e.g., [5, 1, 6]) has been shown to be an effective technique for building highly available systems able to withstand partial compromises. While earlier intrusion-tolerant replication systems guaranteed correctness, recent protocols [1, 6] also guarantee minimal performance degradation whilst under attack and are hence *survivable*. Importantly, these recent protocols have also demonstrated the ability to scale to support thousands of clients. Survivability is maintained as long as no more than a threshold fraction of the replicas (typically f out of $3f+1$ [10]) is compromised. A distinguishing feature of intrusion-tolerant systems is that they do not require prior knowledge of attack signatures and behaviors.

This paper reports on our experiences to date building

*Yair Amir's work was partially supported by the National Science Foundation under grant 0716620.

CSIIRW '11, October 12-14, Oak Ridge, Tennessee, USA

[1] McAfee reported more than 14 million unique pieces of malware in its Q3 2010 threat report, almost quadrupling since 2007.

[2] McAfee [3] estimates that the cost of downtime can be as high as six million dollars per day.

the first survivable SCADA system. We employ intrusion-tolerant replication using the state machine approach [11, 9]. Specifically, we use the Prime replication protocol [1] as an important building block, but, as elaborated upon below, we were confronted with fundamental architectural mismatches when analyzing the integration with SCADA. After developing solutions to these obstacles, we translated this proven family of survivability protocols from the research domain into a practical reality.

The novel contributions of this paper are: (i) the design and development of three new protocols necessary to integrate intrusion tolerance with a SCADA architecture; (ii) methods for eliminating the hardware cost of replication (crucial in many commercial settings) without sacrificing crash fault tolerance; (iii) results to validate the architecture, showing that our system currently performs sufficiently well to meet the needs of large-scale SCADA systems.

2. SURVIVABLE SCADA: FROM RESEARCH TO REALITY

Since SCADA systems are high-value targets for attackers, it would be beneficial to harden them via intrusion-tolerant replication. Unfortunately, while it would seem that SCADA should be "just another application," capable of running seamlessly on top of an intrusion-tolerant replication engine, there are several challenges specific to SCADA that are significant barriers to a practical integration. This section describes these obstacles and gives an overview of the new algorithms we developed to overcome them.

A SCADA system consists of the following main components:

- One or more **Remote Terminal Units (RTUs)**, which communicate with, and aggregate data from, local sensors in the field. Some larger systems can have several thousand RTUs.
- A **SCADA Server**, which periodically polls the RTUs and maintains a real-time database containing the current state of each one. The SCADA Server can also send supervisory control commands to the RTUs.
- One or more **Human Machine Interface (HMI)** workstations, which periodically query the SCADA Server so that the state of the RTUs can be graphically displayed for a human operator.

The highest value asset is the SCADA Server, and its compromise can have serious consequences for the control and monitoring of the entire system. Therefore, the SCADA Server is the prime candidate for protecting via replication. Figure 1 depicts a minimal configuration of the survivable SCADA architecture, where the system can survive the compromise of one of the SCADA Server replicas.

There is an important difference between the SCADA Server and the typical client-server applications that leverage replication. Most client-server applications are *client driven*: the server takes actions as the result of executing client requests; the reliability of the communication link between client and server is ensured by client retransmissions; and server replies are sent to the requesting client. In contrast, communication between the SCADA Server and the RTUs is primarily *server driven*, where the SCADA Server can spontaneously send a message (e.g., a poll request).[3]

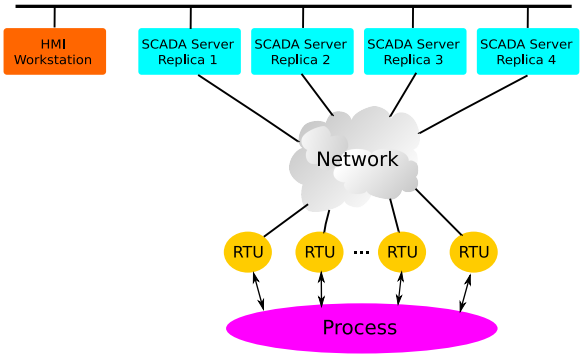

Figure 1: Survivable SCADA Architecture.

This seemingly small difference has large implications on the functionality required from the replication engine. Indeed, bringing intrusion tolerance techniques from the research setting to the practical SCADA setting necessitated the invention of several new algorithms, which are described in the following subsections.

2.1 Scalable Intrusion-Tolerant Synchronization

That the SCADA Server spontaneously initiates messages implies that it takes action based on the passage of time. Absent synchronized clocks, the passage of time is observed in a non-deterministic way at different SCADA Server replicas; if each SCADA Server replica were to make a state transition based on the passage of time on its local clock, the replicas could become inconsistent with one another. Thus, a mechanism is needed to enable the replicas to agree on the logical point in time at which a time-based action should be taken. Of course, this mechanism must be intrusion-tolerant so that malicious replicas cannot disrupt the agreement.

Since large SCADA systems may contain thousands of RTUs, each of which may be polled individually, the synchronization mechanism must scale with the number of entities being polled (or, put another way, with the number of different synchronization points being agreed upon). Existing techniques [8] require a number of messages proportional to the number of synchronization points requested by the server application. To meet the scalability requirements of the SCADA Server, we invented a new protocol that requires a constant number of messages to be exchanged, independent of the polling period of the application. Our protocol makes no assumptions about the relative speeds of the replicas' local clocks and prevents compromised replicas from arbitrarily advancing or delaying the agreed upon time.

Intuitively, the protocol works by having each replica periodically send a message containing its local clock value and a number uniquely identifying the last synchronization point the replica believes to have arrived (as measured on its local clock). These messages are agreed upon using the intrusion-tolerant replication protocol, and a replica can act on the synchronization point when it is convinced that enough time has elapsed on the local clock of at least one correct replica.

[3]Some SCADA communication follows the traditional client-server pattern, such as the request/reply protocol between an HMI workstation and the SCADA Server.

2.2 Intrusion-Tolerant Reliable Channels

Many existing SCADA systems make use of a reliable transport protocol, such as TCP, to pass messages between the SCADA Server and the RTUs. In contrast, intrusion-tolerant replication systems tend to use UDP and implement their own reliability.[4] In such systems, the replication engine is responsible for passing application messages between clients and the server replicas.

Unfortunately, since existing intrusion-tolerant replication systems implicitly assume that the application is client driven, they provide only limited support for reliable communication between a client and the server replicas. Most use a transaction-based protocol, where the client retransmits its request if it does not receive a response within a timeout period. This approach makes additional implicit assumptions, namely that the server application will always generate a response that can be used by the client as an acknowledgement, and that this acknowledgement message will be sent to the requesting client.

SCADA applications require a more flexible approach to achieving reliability. Since the SCADA server replicas can spontaneously initiate messages, the system needs a mechanism for reliable communication from the replicas to each RTU (and to the HMI workstation). In addition, the execution of a message (e.g., a poll response) by the replicas may cause them to send a reply to an entirely different entity (e.g., the HMI workstation), or to not send a reply at all.

To address this issue, we developed two protocols that together implement the abstraction of an *intrusion-tolerant reliable channel* between clients (i.e., the HMI workstation and the RTUs) and the SCADA Server replicas. Each protocol handles a different communication direction. Since the endpoints of the channel are asymmetric (i.e., one is a single client process and the other is a set of server replicas), using two unidirectional protocols allows us to optimize performance in each direction. The reliable channel provides FIFO delivery between a client and the replicas, with semantics similar to that of a non-blocking TCP socket.

Several factors make it challenging to design an efficient intrusion-tolerant reliable channel. For example, the channel must ensure the desired semantics even when the compromised replicas (or a malicious client) send invalid messages, attempt to delay messages, or try to cause the correct participants to consume excessive resources. In addition, the channel implementation must be efficient, since the replicas may communicate with many RTU endpoints.

Due to space limitations, we omit a full description of the reliable channel implementation here. Instead, we highlight one of its important properties: the replicas do not need to agree upon the order in which to execute client acknowledgements. This significantly reduces the overhead of the protocol, because agreement is the most expensive operation in an intrusion-tolerant replication system. Interestingly, this property violates the state machine approach (where each replica executes the same client messages in the same order), but it does so in a way carefully designed to ensure correctness. In particular, the replicas *do* agree on when a connection with a client is established or should be terminated, and thus they all agree on the status of the channel.

[4]As noted in [5], TCP is poorly suited to systems with potentially faulty receivers because, by failing to send acknowledgements, the faulty receivers can require correct replicas to buffer an unbounded number of messages.

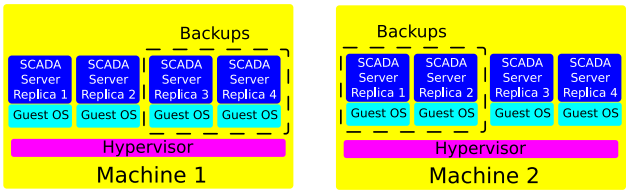

Figure 2: Virtualization-based Architecture.

3. SURVIVABILITY WITH NO ADDITIONAL HARDWARE

As the entity controlling the entire system, the SCADA Server must be protected against crashes. Current SCADA systems typically use a primary/hot-standby approach to provide crash fault tolerance, with each running on its own physical machine.

Our survivable SCADA architecture requires running four replicas to tolerate one compromise. The simplest deployment strategy is to place each replica on its own physical machine, thus providing maximal crash fault tolerance. However, while the cost of a few additional machines may not be an issue for some SCADA customers, others will be reluctant to adopt a solution requiring twice as much hardware as the existing approach (both because of the cost of the hardware itself and the additional management cost). Indeed, our experiences have taught us that reducing the hardware cost of replication is a key design requirement if our survivable SCADA architecture is to be adopted in practice.

To address this issue, we developed a solution that uses virtualization to enable four replicas to run on only two physical machines (like the current primary/hot-standby approach) but without sacrificing crash fault tolerance. Specifically, the SCADA Server application continues providing correct, timely operation even if one of the physical machines crashes.

Before describing our approach in more detail, we note that the use of virtualization raises as an issue an important practical trade-off. Using virtualization saves on hardware but requires trusting the hypervisor to operate correctly; using four separate physical machines requires no additional trust assumptions but at the cost of more hardware. Which configuration to prefer ultimately depends on deployment-specific factors, but we believe cost constraints will lead many to opt for the virtualization-based approach. In this case, rather than blindly trusting the hypervisor, a more prudent approach is to attempt to endow the hypervisor with as much trustworthiness as possible (e.g., by measuring and verifying its integrity in real-time [2], by protecting its control flow [13], etc.).

We conclude this section by giving some intuition about our virtualization-based architecture. We first note that virtualization itself is not enough to provide the necessary degree of crash fault tolerance. To see why, note that any placement of four replicas on two physical machines can result in a scenario in which the crash of one of the physical machines leaves fewer than three running replicas (which is the minimum number needed to ensure that the system remains available). In order to provide continued operation in the face of a physical machine crash, we leverage a recent virtualization technique called *transparent high availability*, or *fault tolerance*. This technique associates with each virtual machine (VM) a backup VM that runs in lockstep with

its primary. The backup runs on a different physical machine from the primary and is automatically activated by the hypervisor if the primary crashes (see Figure 2). The backup is instantiated within several hundred milliseconds, in exactly the same state the primary was in.

We have explored two different fault tolerance solutions, one using VMware's Fault Tolerance approach [12] and the other using open-source tools based on the Xen hypervisor [4] and a synchronization protocol called Remus [7]. While our testing indicates that both solutions can meet our needs, there are some key differences between them. Besides the obvious price difference, the two solutions have different performance characteristics. In VMware's approach the backup replicas consume the same amount of processing resources as the primaries, while in Xen/Remus the backups remain passive until failover. However, the primary/backup synchronization in VMware introduces less latency than the checkpoint-based approach of Xen/Remus. There are also subtle but important differences in the failover semantics: VMware prevents "split-brain" scenarios from occurring but requires shared storage, while Xen/Remus does not depend on shared storage but currently cannot prevent split-brain behavior in certain cases if the link between the two physical machines breaks. This can be mitigated in practice by using NIC teaming.

4. DEPLOYMENT CONSIDERATIONS

In this section we comment on the results of some preliminary performance benchmarks that we have run on our replication engine. We also discuss what impact the replication engine has on SCADA application processing.

We are interested in both the throughput of our replication engine (i.e., the number of client messages that it can process per second) and the amount of latency added to the processing of SCADA events by the replica agreement protocol. Intuitively, throughput is important because a higher throughput means that the system can scale to larger numbers of RTUs; latency is important because it must be low enough to facilitate the real-time control and monitoring required of a SCADA application.

The current implementation of our replication engine can handle more than 10,000 messages per second with an average latency of approximately 100 ms. We expect this performance to be sufficient for most SCADA systems, even large-scale systems with several thousand RTUs being polled approximately once per second. Encouragingly, we also plan to implement several optimizations that we expect will allow the system to scale even further. Clearly, depending on the performance requirements of the SCADA application in question, it may be prudent to tune the replication engine so that it is optimized more towards throughput or latency.

One trend that works in our favor is the increasing adoption of multi-core processors. Our system leverages this trend by running the replication engine in a separate thread from the SCADA Server application itself, thus allowing the two to execute on different cores. This enables the replica agreement protocol to run in parallel with SCADA application processing, minimizing the extent to which the two compete for CPU resources. It also minimizes the constraints placed on the application by the replication engine. For example, a blocking call in the application will not prevent additional events from being agreed upon (although, of course, the replication engine will not be able to deliver these subsequent events until the application becomes unblocked).

5. CONCLUSIONS

This paper documented our experiences to date in designing and implementing the first survivable SCADA system. We described the unique requirements imposed by the SCADA architecture and gave an overview of several new techniques facilitating the integration of intrusion-tolerant replication and SCADA. Our initial experimental results show that our replication engine can perform well enough to meet the needs of even large-scale SCADA systems containing thousands of RTUs.

6. REFERENCES

[1] Y. Amir, B. Coan, J. Kirsch, and J. Lane. Prime: Byzantine replication under attack. *IEEE Transactions on Dependable and Secure Computing*, 8(4):564–577, 2011.

[2] A. M. Azab, P. Ning, Z. Wang, X. Jiang, X. Zhang, and N. C. Skalsky. HyperSentry: Enabling stealthy in-context measurement of hypervisor integrity. In *Proceedings of the 17th ACM Converence on Computer and Communications Security*, pages 38–49, 2010.

[3] S. Baker, S. Waterman, and G. Ivanov. In the crossfire: Critical infrastructure in the age of cyber war, 2009.

[4] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield. Xen and the art of virtualization. In *Proceedings of the 19th ACM symposium on operating systems principles*, pages 164–177, 2003.

[5] M. Castro and B. Liskov. Practical Byzantine fault tolerance and proactive recovery. *ACM Trans. Comput. Syst.*, 20(4):398–461, 2002.

[6] A. Clement, E. Wong, L. Alvisi, M. Dahlin, and M. Marchetti. Making Byzantine fault tolerant systems tolerate Byzantine faults. In *Proceedings of the 6th USENIX Symposium on Networked Systems Design and Implementation*, pages 153–168, 2009.

[7] B. Cully, G. Lefebvre, D. Meyer, M. Feeley, N. Hutchinson, and A. Warfield. Remus: High availability via asynchronous virtual machine replication. In *Proceedings of the 5th USENIX Symposium on Networked Systems Design and Implementation*, pages 161–174, 2008.

[8] J. Kirsch. *Intrusion-Tolerant Replication Under Attack*. PhD thesis, The Johns Hopkins University, Baltimore, Maryland, 2010.

[9] L. Lamport. Time, clocks, and the ordering of events in a distributed system. *Commun. ACM*, 21(7):558–565, 1978.

[10] L. Lamport, R. Shostak, and M. Pease. The Byzantine generals problem. *ACM Trans. Program. Lang. Syst.*, 4(3):382–401, 1982.

[11] F. B. Schneider. Implementing fault-tolerant services using the state machine approach: A tutorial. *ACM Computing Surveys*, 22(4):299–319, 1990.

[12] VMware. Protecting mission-critical workloads with VMware Fault Tolerance, 2009.

[13] Z. Wang and X. Jiang. HyperSafe: A lightweight approach to provide lifetime hypervisor control-flow integrity. In *Proceedings of the 31st IEEE Symposium on Security and Privacy*, pages 380–395, 2010.