Chapter 14

MODELING AND MACHINE-CHECKING BUMP-IN-THE-WIRE SECURITY FOR INDUSTRIAL CONTROL SYSTEMS

Mehdi Sabraoui, Jeffrey Hieb, Adrian Lauf and James Graham

Abstract This chapter describes the formal modeling and machine-checking of a bump-in-the-wire device that secures field device communications in industrial control networks. Field devices serve as the connection points between computer-based control systems and the physical processes being controlled. Industrial control network traffic is routinely checked for transmission errors, but limited mechanisms are available for combating attacks that exploit industrial control protocols to target critical infrastructure assets.

This chapter focuses on a bump-in-the-wire solution that can be retrofitted on field devices to provide security functionality. The TLA+ formal specification language in combination with the isolation guarantees provided by the seL4 microkernel are used to demonstrate that the bump-in-the-wire solution provides important security and liveness properties. The resulting machine-checked system correctly applies hash-based message authentication to verify the authenticity of incoming messages while being resistant to attacks.

Keywords: Industrial control systems, security, formal methods, verification

1. Introduction

In 2014, the National Institute of Standards and Technology (NIST) released a cyber security framework [36] intended to enhance the cyber security postures of critical infrastructure assets. Industrial control systems are used across the critical infrastructure sectors, including chemical, critical manufacturing, dams, energy, food and agriculture, nuclear facilities, transportation systems and water treatment systems. These installations are often unpatched, offer low resilience to unexpected network traffic and incorporate few mechanisms that protect against malicious activities [43]. The NIST framework –

Published by Springer Nature Switzerland AG 2019
J. Staggs and S. Shenoi (Eds.): Critical Infrastructure Protection XIII, IFIP AICT 570, pp. 271–288, 2019.
https://doi.org/10.1007/978-3-030-34647-8_14

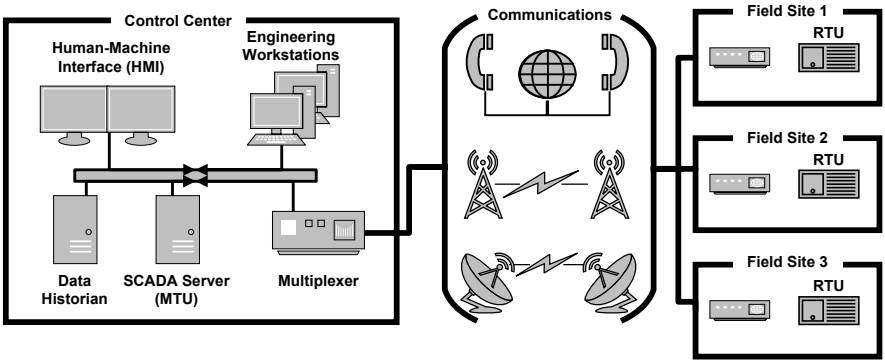

Figure 1. Typical industrial control system.

and the executive order that drove it [38] – highlight the need for strong cyber security in critical infrastructure assets that have historically focused on safety and physical security.

A secure system, like a safe system, should be designed to preserve certain properties and the implementation should satisfy these properties. Thus, the problem of developing a secure system involves two sub-problems: (i) formally express the core security properties of the system; and (ii) prove that the core security properties hold for every possible system state. Whether the system is a virtual banking application or a water treatment plant control system, high degrees of trust in the system design and implementation are paramount.

An industrial control system is a general term that describes myriad configurations of networked computer systems that operate and control physical equipment and processes in infrastructure assets. These systems make heavy use of mature, low-power and low-overhead technologies such as RS-232 communications and decades-old networking protocols to pass control commands and data between devices [21].

Figure 1 shows a typical industrial control system with three major components: (i) central control unit (or master terminal unit (MTU)) at the control center; (ii) remote control units (or remote terminal units (RTUs) or more generally field devices) at field sites; and (iii) communications equipment and protocols that link the central and remote units. The central control unit has a human-machine interface (HMI) used for plant operations. The field devices, which perform sensing and control, range from simple circuit boards powered by embedded controllers to expensive programmable logic controllers (PLCs) with racks of components. Since the master or central control unit gathers data from the field devices and sets operating parameters and issues commands to field devices to orchestrate plant operations, the overall system is often referred to as a supervisory control and data acquisition (SCADA) system.

Unlike typical enterprise information technology networks where confidentiality is a priority, industrial control networks value availability and integrity

Table 1. Modbus ASCII protocol data unit.

Start	Address	Function	Data	LRC	End
":"	2 bytes	2 bytes	Up to 504 bytes	2 bytes	"\r\n"

above confidentiality [33] – keeping the data in the system private is not as important as keeping the system running properly. Cyber threats to industrial control systems reflect this priority – attackers seek to disturb and disrupt the controlled processes rather than exfiltrate sensitive data. Attacks that disrupt network flows in industrial control systems can be devastating from the financial and safety perspectives [14, 42, 44, 46]. The importance of maintaining availability disincentivizes regular system changes or updates for fear of unscheduled downtime.

Attacks on SCADA protocols are often trivial because the protocols have few, if any, security features. Modbus, one of the most popular SCADA protocols, was developed in 1979 [35]. Modbus is a simple, connectionless protocol that can be modeled rigorously. This research considers the Modbus ASCII protocol [35].

Figure 1 shows the layout of a Modbus ASCII protocol data unit (PDU). It comprises a colon ":" to signal a new packet, two bytes for the recipient address, two bytes for the function code; up to 504 bytes for payload data, two bytes for a longitudinal redundancy check (LRC) that helps detect transmission errors and an ending character sequence "\r\n."

A fully verified microkernel such as seL4 [27] provides a platform for constructing software solutions for high assurance environments. Instead of a security statement such as "this system works for my expected inputs," seL4 enables a high-assurance statement to be made about the system: "this system works as expected for every possible input at an affordable cost." The platform provides process isolation through isolated address spaces. This supports the development of logically-contained processes, where each component acts on its own accord and communications between each component are controlled strictly.

The seL4 architecture enables a properly-designed microkernel environment to be treated as a distributed system on a single chip. A concurrency modeling language such as TLA+ [30–32] can be used to model the states and interactions of components, and each trace of state transitions can be machine-checked to conform to the modeled security properties (e.g., the secret key is confined to a certain component and only valid packets can reach the inner components). PlusCal, an algorithm language translatable to TLA+, can be used to model the sequential steps in the operation of each component.

This research focuses on a bump-in-the-wire solution that can be retrofitted on SCADA field devices to provide security functionality. It leverages TLA+ and PlusCal to model the components of the Modbus-based bump-in-the-wire

security device that is designed with the seL4 microkernel in mind. The TLA+ formal specification language in combination with the isolation guarantees provided by the seL4 microkernel are used to demonstrate that the bump-in-the-wire device provides the desired security and liveness properties. The resulting machine-checked system correctly applies hash-based message authentication to verify the authenticity of incoming messages while being resistant to attacks. In particular, the verified specification assures that only valid, authenticated Modbus packets flow across a field device, that a compromised component cannot break the security properties of adjacent components, that the device is not vulnerable to replay and spoofing attacks, and that the device is resilient to malformed and malicious traffic.

2. Background

The threats to traditional information technology infrastructure are well documented, but when industrial control systems merge with traditional information technology networks, the threats become even more significant in their scope and magnitude. Attacks such as the 2000 Maroochy Water breach [1] and the 2010 Stuxnet worm [3, 14, 15] demonstrate that even isolated industrial control systems are vulnerable to external threats. A Kaspersky Lab report [24] states that more than 10% of all blocked threats on industrial control computers originated from removable media such as USB drives. An attacker who gains access to an industrial control network finds little in the way of security mechanisms. Industrial communications protocols have limited, if any, security mechanisms and the SCADA protocols often lack basic authentication and integrity checking [34]. Meanwhile research has demonstrated that certain interactions involving industrial control protocols (e.g., Modbus TCP) and their carrier protocols (e.g., TCP/IP) can defeat security and integrity guarantees [13].

The longevity and uptime requirements imposed on industrial control systems present unique challenges to keeping them secure. Physical systems can be difficult to test, upgrade and replace. Thus, the ages of physical systems and their software are often measured in decades and the downtime costs are significant. Fragile, out-of-date firmware and protocol stacks are common and they exhibit anomalous behavior with non-conforming, let alone, malicious, traffic. Unexpected network traffic can cause heavy machinery to act in unpredictable ways. Any system state that has not been explicitly evaluated during the design phase poses a risk of disruption.

2.1 Industrial Control System Security

Efforts to address industrial control system security are generally aligned with the categories defined in the NIST framework [36] – detect, protect, identify, recover and respond. Detection leverages monitoring methodologies and tools for generating logs for low-performance field devices in SCADA systems, exporting logging data without fear of opening new attack vectors and con-

ducting analyses without impacting system operations (see, e.g., [18, 41]). Protection involves the development and application of security mechanisms in software [2] or hardware [8]. Identify involves cataloging system assets and potential risks with efforts concentrating on the system as a whole [18] as well as on individual components such as vendor software and hardware [23–25]. Recovery involves picking up the pieces after an incident (accident or malicious) and incorporating the lessons learned in preparing for future incidents [6, 45]. Finally, response involves all activities that follow the detection of an incident (whether successful or not), including communication, analyses and mitigation [9, 39]. The work described in this chapter falls in the protection and detection categories, with the cryptographic signing mechanism supporting both forgery protection and detection.

2.2 seL4 and CAmkES

The seL4 microkernel has been fully verified from design to implementation to provide an exceptionally high level of assurance [26, 27]. It has evolved from the OKL4 family of microkernels that were reduced in size to the point where guarantees of bug-free code could still be realized. The seL4 microkernel has the verified ability to logically separate processes and implement highly-specified channels of communications between components in an architecture. If a cell in the kernel is compromised, it is shown that the other cells still maintain their desired security properties. This enables an abstract implementation of Rushby's separation kernel [40] for reducing a large security kernel into smaller, more easily provable, components that mimic a distributed system.

CAmkES is a component platform designed to address the increasing complexity and unreliability of embedded systems by facilitating the modular design of system services [16, 29]. CAmkES supports microkernel development; its design favors a low-overhead approach to accommodate the challenges involved in microkernel development. The CAmkES architecture provides a component model, standard interfaces (including support for user-defined interfaces) and user-defined interactions between components using these interfaces.

2.3 TLA+ and PlusCal

TLA+ is a formal language for modeling and reasoning about concurrent systems [30–32]. PlusCal is a language for modeling algorithms in a much more expressive manner than typical programming languages. TLA+ and PlusCal use mathematical notation to expand their reach beyond programming languages to allow for rigorous definitions and descriptions of algorithms and systems. The mathematical notation also facilitates model checking and proofs of properties of algorithms and systems.

PlusCal, which is a programmer-friendly option in the TLA+ toolchain, can be automatically translated to TLA+ and used with the TLC model checker provided by TLA+. TLC is a brute-force model checker that explores states up to a certain number of transitions, raises alerts about properties that have been

violated and provides traces where the violations occurred. This research has employed TLA+ for two categories of checks that are defined and performed for each state and transition reached by the model checker: (i) invariants; and (ii) liveness properties.

Invariants are statements are always true regardless of the system state. An invariant in a simple banking example is: an account balance is never less than zero. This invariant is expressed in TLA+ as follows:

$$\forall\, acct \in Accounts : acct.balance \geq 0 \tag{1}$$

The initial state of a system might satisfy all the invariants, but then the system may never leave its initial state. A system that never changes its state is not useful. Therefore, liveness properties are specified. A liveness property – or temporal property – checks that a property is eventually true. A liveness property for a binary clock that alternates between zero and one is: if the clock bit is currently one, then it will eventually be zero. An example liveness property that is applicable to this research is: if an incoming message is a valid communications packet, then it will eventually be processed and forwarded. This liveness property is specified as follows:

$$message.valid = True \rightsquigarrow forward = True \tag{2}$$

3. Related Work

The DNP3 SCADA communications protocol is commonly used to transmit commands and data between a central operations center and remote substations [10–12]. DNP3 Secure Authentication [19] adds security mechanisms to the basic DNP3 protocol to provide authentication and authorization. Amoah et al. [2] have conducted a formal behavioral analysis of DNP3 Secure Authentication. They employed colored Petri Nets to model common replay, modification and spoofing attacks on the DNP3 Secure Authentication specification, in the process, discovering a previously-unknown attack that could be launched by an attacker with access to data in motion, but without access to the secret key. Their modeling of the communicating entities and an attacker revealed that the snooping of non-aggressive challenge response sessions by the attacker could modify the sequence number and issue a request in the aggressive mode of DNP3 communications. This resulted in the attacker being able to spoof a valid request and replay previous messages sent in the network. This attack specifically targeted critical requests that changed system operations by modifying set-points and setting parameters.

Kuhn and Dray [28] have applied formal modeling and verification techniques to a microprocessor-based device to create a smart token system for controlling access to network hosts. Their objective was to create a civilian security-critical system where formal methods could be used to improve the system; this system could then serve as a testbed for applying formal methods for securing larger projects. The application of formal methods contributed to correcting

inconsistencies in the smart token design, improving its resilience and finding a subtle, but critical, bug that could have completely compromised the security provided by the smart token. Kuhn and Dray noted that the application of formal methods to their project had tangible benefits despite its limited focus and small budget.

The DARPA-funded High-Assurance Cyber Military Systems (HACMS) Program [17] has worked with Boeing to increase the security of an unmanned H-6U helicopter in a practical manner using formally-verified code and the seL4 microkernel. Engineers were able to separate the components of the H-6U into different virtual machines running on an seL4 hypervisor and then convert certain components from the virtual machines to fully-verified native components. Although verifying the entire H-6U system was infeasible, the effort was able to verify select components to expand the trusted computing base of the system and enhance its reliability. The upgraded H-6U was able to stay in flight and complete its test mission despite being actively attacked by multiple compromised components and a professional red team. The attackers were unable to pivot from the compromised components to affect H-6U flight control or mission control.

4. Security Preprocessor Checking Using TLA+

Hieb and Graham [20] have proposed a bump-in-the-wire solution that enhances communications security between control centers and field sites that employ legacy systems. The solution introduces an inline security preprocessor that retrofits raw communications packets with authorization and authentication; the encapsulations are stripped at the field sites. The bump-in-the-wire design does not impact control and configuration operations because the preprocessor is transparent to the control center and field devices. The approach adds some latency; however, since the timing requirements for control systems range from microseconds to seconds, the latency provided by the added security is acceptable in many industrial control systems [19, 20].

The preprocessor of Hieb and Graham [20] incorporates three components: two for communications and a middle component that performs the authorization and authentication checks. This section extends the design by adding higher assurance via guarantees provided by seL4 and model verification using TLA+. The desired properties of the preprocessor are first translated into TLA+. Following this, a model is built that preserves the security properties.

4.1 System Modeling

The new system model has four components that logically separate the security functions: (i) checking that messages conform to the Modbus specification; (ii) signing messages with SHA-2; (iii) networking with raw Modbus messages in a trusted network; and (iv) networking with encapsulated/signed Modbus messages in an untrusted network. The separation isolates critical decisions,

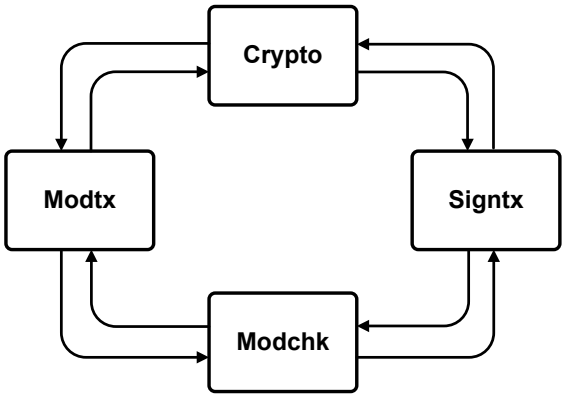

Figure 2. CAmkES components.

rendering exploitation more difficult and supporting parallelism to improve performance should the need arise.

Each of the four components runs as a separate process, confining each critical decision-making block of code to its own memory space. Each component was also modeled separately in TLA+, following which all the components were modeled together to explore the security properties of the system as a whole. Figure 2 shows the four components. Each component provides a message interface and consumes messages from two other components.

Each thread component was specified separately using PlusCal and then collectively as an asynchronous system. An example is the signature checking thread of the Crypto component in Figure 2. Each thread is provided with a queue from which it dequeues a message and performs its processing. The Modtx component can place messages in the queues of Modchk and the message signing thread of Crypto. The Signtx component can place messages in the queue of Modchk and the signature verification thread of Crypto. This system of message passing to queues is an abstraction of the interprocess communications between the four CAmkES components.

TLA+ and its TLC model checker enable sets to be defined and each element of the sets to be used when exploring model states. It would make sense to define a set that includes every possible input for every thread. However, in the case of network communications, it is impossible to specify the set of all possible bits that can be read from or written to a serial port. This task is infeasible even when the message size is restricted to the Modbus specification. Therefore, this work uses specially-crafted packets to test the typical cases and edge cases of valid and invalid messages that could be encountered with the goal of showing that every message is covered by these cases. The serial port was also abstracted as receiving and sending messages one byte at a time to limit the search spaces of incoming and outgoing data to single bytes at a time.

Table 2. Desired system properties.

System	Property
A message in an internal queue has a valid length	Invariant
A message in a networking queue waiting to be printed has been validated	Invariant
Only a well-formed, valid Modbus message reaches the trusted network	Invariant
Only a well-formed, signed Modbus message reaches the untrusted network	Invariant
A message entering the system reaches the opposite component and is eventually consumed	Liveness
A component queue is eventually emptied	Liveness

The checking begins when a set of messages arrive at the Modtx component and a different set of messages arrive at the Signtx component. The simulation ends when all the queues are empty and there are no further states to explore. Microsoft's specification of Cosmos DB [5] is a similar message passing system that uses TLA+.

4.2 Invariants and Liveness Guarantees

Table 2 shows the desired system properties. The four invariants ensure that: (i) only messages with valid lengths make it to the inner components; (ii) messages in the queues of the networking components are checked for validity by the components from where they came and the valid messages are printed to the screen; (iii) only well-formed and properly-verified Modbus messages reach the trusted network; and (iv) only well-formed and properly-signed Modbus messages reach the untrusted network.

Liveness properties are more difficult to check in a large model. However, the two liveness properties in Table 2 ensure that: (i) messages entering the preprocessor from a networking component eventually reach the opposite networking component and are eventually consumed; and (ii) all queues are eventually emptied.

The messages selected to push through the system drive the model. Since it is not possible to model every possible input to the networked components, only certain amounts and types of messages that exercise every state in the model are employed. The TLC model checker keeps track of the number of times each state is reached; states that are reached zero times are of particular interest. Although a concise set of messages has not been found to hit every state in a single run without running out of memory, each state can be reached over multiple runs with different sets of inputs. The inputs include messages that are too short to be valid, messages that are too long to be valid, messages

that are valid, messages that contain incorrect characters, messages that cause restarts, and all possible single-byte inputs.

4.3 Specifying and Checking Properties

Two components interface with an output (i.e., printing to the screen in a virtual machine and printing to a serial port on hardware). The first component, Modtx, reads and transmits Modbus messages while the second component, Signtx, reads and transmits signed Modbus messages.

The Modtx component has a read thread and a write thread. The component reads a byte at a time from the serial buffer until its buffer is full or a colon (`:`) is received to signal a new Modbus packet or a carriage return/line feed (`\r\n`) is received to signal the end of a Modbus packet. The Modtx buffer is large enough to hold the largest Modbus packet as defined in the Modbus specification. When the buffer fills without the `\r\n`, it is cleared and its index is reset. If a colon is received, then the buffer is cleared and the colon is placed at the head of the buffer. When a `\r\n` is received, the Modtx component interprets its buffer as containing a completed Modbus packet. The packet is assigned an ID and is simultaneously passed to both the cryptography component and the Modbus checking component.

The Modtx write thread gathers messages from the Modbus checking component and Crypto component. If a new message with a unique ID comes from the Crypto component, it is stored until a message with the same ID is also received from the Modbus checking component. After the two messages have been received, both messages are checked for validity. If the Crypto component was able to validate the message signature and the Modbus checking determined that the message contained a correctly-formed Modbus packet, then the raw Modbus message is printed to the output. Thus, a message is printed only if both the checking components agree the message is valid.

The invariants and liveness properties in the Modtx specification ensure that only well-formed, verified packets are printed, that the data to be transmitted is eventually transmitted and only whole (not necessarily well-formed) messages reach the inner components.

The top half of Table 3 shows the two invariants and two liveness checks for the Modtx read thread, which reads bytes from the trusted serial port. The fist invariant stipulates that the receiving buffer never exceeds the maximum size of a Modbus message. The second invariant stipulates that the application buffer (holding data to be forwarded to the inner components) never exceeds the maximum size of a Modbus message. The two liveness checks ensure that if a complete Modbus message is received (starting with a colon and ending with `\r\n`) and is under the maximum size, then the message is eventually processed and forwarded to the inner components.

The bottom half of Table 3 shows the two invariants and three liveness checks for the Modtx write thread. The first invariant stipulates that a byte is only sent if the original message is a valid Modbus message. The second invariant stipulates that only valid Modbus characters reach the sending register. The

Table 3. Desired Modtx component properties.

Modtx Read Thread	**Property**
A receiving buffer containing bytes read from the serial port does not hold more than a Modbus message	Invariant
A buffer containing bytes to be forwarded to inner components does not hold more than a Modbus message	Invariant
A well-formed Modbus message in the receiving buffer is eventually processed	Liveness
A well-formed Modbus message is eventually forwarded to inner components	Liveness

Modtx Write Thread	**Property**
A byte is printed to the serial port only if the entire Modbus message is valid	Invariant
Only valid Modbus characters are stored in the serial port register	Invariant
A sending buffer containing a valid Modbus message is eventually emptied	Liveness
A valid Modbus message to be printed is eventually printed	Liveness
A transmit flag that is raised is eventually lowered	Liveness

first liveness check ensures that if there is a message to send (transmit flag is raised) and the message is a valid Modbus message, then the buffer is eventually emptied. The second liveness check ensures that if there is a valid Modbus message to send, then the bytes are eventually sent. The third liveness check ensures that if the transmit flag is raised, then it is eventually lowered to allow a new message to start the process again.

The Signtx component works in a similar manner as the Modtx component, but with two exceptions. Its reading thread looks for a complete encapsulated message instead of a complete Modbus message and it utilizes the Crypto component to validate signatures instead of its signing functionality. The Signtx component assigns ID numbers to incoming messages and forwards decapsulated messages to the Modbus checking component.

4.4 Checking Modbus Properties

The Modbus protocol is modeled within TLA+ and a mechanism is incorporated to check that a given stream of bytes conforms to the Modbus standard [35]. These are necessary to ensure that only valid Modbus packets can traverse the preprocessor.

The Modbus specification is more a definition than a model of behavior. As shown in Table 1, a Modbus message has five simple fields and a longitudinal redundancy check for determining transmission errors. The lengths and character sequences in the fields in a Modbus packet are clearly defined, so it is straightforward to analyze each field. The head of the packet is checked to be the colon. The address and function fields are both checked to be two-byte hexadecimals. The data field is checked to be between zero and 504 bytes long and all the characters are hexadecimal. The end of the data field is found by taking every character from its start to the length of the packet and subtracting four. The longitudinal redundancy field is checked to be two hexadecimal characters and that the value in the packet matches the computed value. The end field is checked to contain exactly `\r\n`.

The Modbus message definition is used as an invariant in other specifications. This requires checking that the length of a received message is within the Modbus size limits. Next, each field of the message is examined individually to determine if it matches its definition. A message is a valid Modbus message when all the components match the definitions in Table 1. The Modchk component passes its decision (valid or invalid) along with the message itself to the opposite component from which the message was received.

4.5 Checking Cryptographic Properties

The Crypto component has two functions: (i) message signing; and (ii) signature verification. The seL4 architecture makes the signing capabilities available only to the Modtx component and the verifying capabilities available only to the Signtx component. This means that signed data only flows one way and unsigned data only flows the other way.

When Modtx receives a raw Modbus message to be signed, it passes the message and message ID to the message signing function in the Crypto component. The Crypto component generates a hash-based message authentication code (HMAC) using the Beringer-Appel verified HMAC and SHA-256 implementations [4, 7] with a preshared key, then forwards the raw Modbus message, the generated HMAC and the message ID to the Signtx component. Alternatively, when Signtx receives a signed message from the untrusted network, it passes the message and the message ID to the signature verification function. The signature verification function separates the raw Modbus message from the received HMAC, generates its own HMAC and compares the two HMACs. It then passes the raw Modbus message, the message ID and the Crypto component decision (valid or invalid) to the Modtx component.

Modeling the Crypto component hides the cryptographic techniques behind a Boolean abstraction as the verification work for the target implementation of HMAC has been done elsewhere [4, 7]. The specification uses an HMAC comparison function that simply returns true or false regardless of the strings being compared. When checking the model, the TLA+ model checker expands the state space for both possibilities, ensuring that the safety or liveness properties are not violated regardless of the decision made by the Crypto component.

Table 4. Desired Crypto component properties.

Message Signing	**Property**
A macMessage is empty or contains a well-formed packet	Invariant
The input and output of the signing function are different	Invariant
A password does not change	Invariant
A message that is received is eventually processed	Liveness
A message that is processed is eventually forwarded	Liveness

Signature Verification	**Property**
A password does not change	Invariant
A message that is flagged as valid is, in fact, valid	Invariant
A message that is invalid is never flagged as valid	Invariant
A message that is received is eventually flagged as valid or invalid	Liveness
A message that is received is eventually forwarded	Liveness

The message signing function has three invariants and two liveness checks (top half of Table 4). The buffer that holds the processed string to be forwarded, macMessage, is checked in all the invariants. The first invariant stipulates that macMessage is either empty or contains a well-formed packet. The second invariant stipulates that the information received and passed by the message signing function are different, demonstrating that processing has taken place. The third invariant stipulates that the password variable is never changed. The two liveness check ensure that a message that is received is eventually processed and forwarded.

The signature verification function has three invariants and two liveliness checks (bottom half of Table 4). As in the case of the message signing function, the first invariant stipulates that the password variable is never changed. The second invariant stipulates that a message marked as valid when it is forwarded to the Modtx component has a validated HMAC. The third invariant stipulates that an invalid message is never flagged as valid. The two liveness checks ensure that, if a message is received, then it is eventually flagged and forwarded to the Modtx component.

5. Discussion

An attacker who targets the network in which the bumper-in-the-wire device operates may attempt to forge messages, or tamper with or replay captured messages. Alternatively, the attacker may attempt to exploit the trust rela-

tionship between the internal network and the device by assuming control and generating malicious traffic that originates from the device as described in [22]. However, the device is immune to these types of attacks because invariants ensure that, regardless of the model state, only valid Modbus messages (whether encapsulated or not) can be printed. Further, if the assumptions described above hold, then the system would not accept messages that have been tampered with (i.e., messages that do not match the attached HMACs). A nonce is included in each encapsulated message to prevent replay attacks.

Restricting the model checking to a feasible state-space required a few tricks. The simplest incarnation of the model completed in roughly ten minutes, but the number of states extended to the tens of millions. Modeling each thread individually enabled around 10,000 states to be explored to verify the invariants and perform the liveness checks. Relying on the verification work applied to the seL4 component architecture enabled the component models to be kept separate while maintaining the queue-based message passing structure of the general model. While these strategies may change according to the project, researchers have shown that even larger projects can see benefits when the state space is restricted to a manageable size [5, 37]

6. Conclusions

Industrial control network traffic is routinely checked for transmission errors, but limited security mechanisms are available for combating attacks that exploit industrial control protocols to target field devices. The bump-in-the-wire solution considered in this research is designed to be retrofitted on field devices to provide the needed security functionality. The TLA+ formal specification language in combination with the isolation guarantees provided by the seL4 microkernel are used to demonstrate that the bump-in-the-wire device provides the desired security and liveness properties. Indeed, the resulting machine-checked system correctly applies hash-based message authentication to verify the authenticity of incoming protocol messages while being resistant to attacks. In particular, the verified specification assures that only valid, authenticated Modbus packets flow across a field device, that a compromised component cannot break the security properties of adjacent components, that the device is not vulnerable to replay and spoofing attacks, and that the device is resilient to malformed and malicious traffic.

Future research will focus on formal proofs at the code level and a verified implementation targeting the seL4 microkernel. The modular design would be conducive to full formal verification because the components are limited in how they interact with other components (if they interact at all). The seL4 proof work provides the isolation and interprocess communications guarantees, facilitating a component-based architecture. The component-based architecture can be abstracted to a distributed system of critical decision points, which TLA+ excels at modeling and checking. Thus, the important next step is to move from a TLA+ specification to executable, verified C code. The verified C code

could be incorporated in the CAmkES build and deployed in a native seL4 application.

References

[1] M. Abrams and J. Weiss, Malicious control system cyber security attack case study – Maroochy Water Services, presented at the *Twenty-Fourth Annual Computer Security Applications Conference*, 2008.

[2] R. Amoah, S. Camtepe and E. Foo, Formal modeling and analysis of DNP3 Secure Authentication, *Journal of Network and Computer Applications*, vol. 59, pp. 345–360, 2016.

[3] N. Anderson, Confirmed: US and Israel created Stuxnet, lost control of it, *Ars Technica*, June 1, 2012.

[4] A. Appel, Verification of a cryptographic primitive: SHA-256, *ACM Transactions on Programming Languages and Systems*, vol. 37(2), article no. 7, 2015.

[5] Azure, Azure Cosmos TLA+ specifications, GitHub (`github.com/Azure/azure-cosmos-tla`), 2018.

[6] M. Bartock, J. Cichonski, M. Souppaya, M. Smith, G. Witte and K. Scarfone, Guide for Cybersecurity Event Recovery, NIST Special Publication 800-184, National Institute of Standards and Technology, Gaithersburg, Maryland, 2016.

[7] L. Beringer, A. Petcher, K. Ye and A. Appel, Verified correctness and security of OpenSSL HMAC, *Proceedings of the Twenty-Fourth USENIX Security Symposium*, pp. 207–221, 2015.

[8] Blue Coat Systems, Blue Coat ICS Protection, Scanner Station Version, USB Malware Defense for Industrial Computers, User Guide, Version 5.3.1, Sunnyvale, California (`docplayer.net/18790337-Blue-coat-ics-protection-scanner-station-version.html`), 2014.

[9] P. Cichonski, T. Millar, T. Grance and K. Scarfone, Computer Security Incident Handling Guide, NIST Special Publication 800-61, Revision 2, National Institute of Standards and Technology, Gaithersburg, Maryland, 2012.

[10] Control Microsystems, DNP3 User and Reference Manual, Kanata, Canada, 2007.

[11] K. Curtis, A DNP3 Protocol Primer (Revision A), DNP3 Users Group, Calgary, Canada (`www.dnp.org/Portals/0/AboutUs/DNP3%20Primer%20Rev%20A.pdf`), 2005.

[12] S. East, J. Butts, M. Papa and S. Shenoi, A taxonomy of attacks on the DNP3 protocol, in *Critical Infrastructure Protection III*, C. Palmer and S. Shenoi (Eds.), Springer, Berlin Heidelberg, Germany, pp. 67–81, 2009.

[13] J. Edmonds, M. Papa and S. Shenoi, Security analysis of multilayer SCADA protocols, in *Critical Infrastructure Protection*, E. Goetz and S. Shenoi (Eds.), Springer, Boston, Massachusetts, pp. 205–221. 2007.

[14] N. Falliere, Stuxnet introduces the first known rootkit for industrial control systems, *Symantec Security Response Blog* (`www.symantec.com/connect/blogs/stuxnet-introduces-first-known-rootkit-scada-devices`), August 6, 2010.

[15] N. Falliere, L. O'Murchu and E. Chien, W32.Stuxnet Dossier, Version 1.4, Symantec, Mountain View, California, 2011.

[16] M. Fernandez, G. Klein, I. Kuz and T. Murray, CAmkES Formalization of a Component Platform, National Information and Communications Technology Research Centre of Excellence (NICTA), Sydney, Australia, 2012.

[17] K. Fisher, J. Launchbury and R. Richards, The HACMS Program: Using formal methods to eliminate exploitable bugs, *Philosophical Transactions, Series A, Mathematical Physical and Engineering Sciences*, vol. 375(2104), article no. 20150401, 2017.

[18] T. Gary, ICS/SCADA smart scanning: Discover and assess IT-based systems in converged IT/OT environments, *Tenable Blog*, June 12, 2018.

[19] G. Gilchrist, Secure authentication for DNP3, *Proceedings of the IEEE Power and Energy Society General Meeting – Conversion and Delivery of Electrical Energy in the 21st Century*, 2008.

[20] J. Hieb, J. Graham, J. Schreiver and K. Moss, Security preprocessor for industrial control networks, *Proceedings of the Seventh International Conference on Information Warfare and Security*, pp. 130–137, 2012.

[21] V. Igure, S. Laughter and R. Williams, Security issues in SCADA networks, *Computers and Security*, vol. 25(7), pp. 498–506, 2006.

[22] Industrial Control Systems Cyber Emergency Response Team (ICS-CERT), Advisory (ICSA-12-231-01B), Sixnet Universal Protocol Undocumented Function Codes (Update B), Idaho Falls, Idaho (`www.us-cert.gov/ics/advisories/ICSA-13-231-01B`), September 17, 2013.

[23] Industrial Control Systems Cyber Emergency Response Team (ICS-CERT), ICS-CERT Advisories, Idaho Falls, Idaho (`ics-cert.us-cert.gov/advisories`), 2019.

[24] Kaspersky Lab ICS CERT, Threat Landscape for Industrial Automation Systems in the Second Half of 2016, Kaspersky Lab, Moscow, Russia, 2017.

[25] Kaspersky Lab ICS CERT, Threat Landscape for Industrial Automation Systems in H2 2017, Kaspersky Lab, Moscow, Russia, 2018.

[26] G. Klein, P. Derrin and K. Elphinstone, Experience report: seL4: Formally verifying a high-performance microkernel, *Proceedings of the Fourteenth ACM SIGPLAN International Conference on Functional Programming*, pp. 91–96, 2009.

[27] G. Klein, K. Elphinstone, G. Heiser, J. Andronick, D. Cock, P. Derrin, D. Elkaduwe, K. Engelhardt, R. Kolanski, M. Norrish, T. Sewell, H. Tuch and S. Winwood, seL4: Formal verification of an OS kernel, *Proceedings of the Twenty-Second ACM Symposium on Operating Systems Principles*, pp. 207–220, 2009.

[28] D. Kuhn and J. Dray, Formal specification and verification of control software for cryptographic equipment, *Proceedings of the Sixth Annual Computer Security Applications Conference*, pp. 32–43, 1990.

[29] I. Kuz, Y. Liu, I. Gorton and G. Heiser, CAmkES: A component model for secure microkernel-based embedded systems, *Journal of Systems and Software*, vol. 80(5), pp. 687–699, 2007.

[30] L. Lamport, The temporal logic of actions, *ACM Transactions on Programming Languages and Systems*, vol. 16(3), pp. 872–923, 1994.

[31] L. Lamport, *Specifying Systems: The TLA+ Language and Tools for Hardware and Software Engineers*, Addison-Wesley, Boston, Massachusetts, 2002.

[32] L. Lamport, The TLA Home Page (`lamport.azurewebsites.net/tla/tla.html`), December 6, 2018.

[33] H. Mackenzie, SCADA security basics: Why industrial networks are different than IT networks, *Tofino Security Blog*, October 31, 2012.

[34] L. Martin-Liras, M. Prada, J. Fuertes, A. Moran, S. Alonso and M. Dominguez, Comparative analysis of the security of configuration protocols for industrial control devices, *International Journal of Critical Infrastructure Protection*, vol. 19, pp. 4–15, 2017.

[35] Modbus Organization, Modbus over Serial Line: Specification and Implementation Guide, V1.02, Hopkinton, Massachusetts (`www.modbus.org/docs/Modbus_over_serial_line_V1_02.pdf`), 2006.

[36] National Institute of Standards and Technology, Framework for Improving Critical Infrastructure Cybersecurity, Version 1.1, Gaithersburg, Maryland, 2018.

[37] C. Newcombe, T. Rath, F. Zhang, B. Munteanu, M. Brooker and M. Deardeuff, Use of Formal Methods at Amazon Web Services, Amazon, Seattle, Washington (`lamport.azurewebsites.net/tla/formal-methods-amazon.pdf`), 2014.

[38] B. Obama, Presidential Policy Directive 21: Critical Infrastructure Security and Resilience (PPD-21), The White House, Washington, DC, February 12, 2013.

[39] M. Permann, K. Lee, J. Hammer and K. Rohde, Mitigations for security vulnerabilities found in control system networks, presented at the *Sixteenth Annual Joint ISA POWID/EPRI Controls and Instrumentation Conference*, 2006.

[40] J. Rushby, Design and verification of secure systems, *Proceedings of the Eighth ACM Symposium on Operating Systems Principles*, pp. 12–21, 1981.

[41] K. Scarfone and P. Mell, Guide to Intrusion Detection and Prevention Systems (IDPS), NIST Special Publication 800-94, National Institute of Standards and Technology, Gaithersburg, Maryland, 2007.

[42] U. Shamir, Analyzing a New Variant of BlackEnergy 3: Likely Insider-Based Execution, SentinelOne, Mountain View, California, 2016.

[43] K. Stouffer, J. Falco and K. Scarfone, Guide to Industrial Control Systems (ICS) Security, NIST Special Publication 800-82, National Institute of Standards and Technology, Gaithersburg, Maryland, 2011.

[44] J. Sullivan and D. Kamensky, How cyber-attacks in Ukraine show the vulnerability of the U.S. power grid, *The Electricity Journal*, vol. 30(3), pp. 30–35, 2017.

[45] United Nations Security Council Counter-Terrorism Committee Executive Directorate (CTED) and United Nations Office of Counter-Terrorism, The Protection of Critical Infrastructure against Terrorist Attacks: Compendium of Good Practices, Geneva, Switzerland, 2018.

[46] D. Wagner, Infrastructure under attack, *Risk Management*, vol. 63(8), pp. 28–33, 2016.