Companion Assisted Software Based Remote Attestation in SCADA Networks

William A. Johnson[1], Sheikh Ghafoor[1], Stacy Prowell[2]
[1] Department of Computer Science, Tennessee Tech University, Cookeville, United States
[2] National Security Sciences, Oak Ridge National Lab, Oak Ridge, United States
wajohnson43@tntech.edu
sghafoor@tntech.edu
prowellsj@ORNL.gov

Abstract: Critical infrastructure such as power generation and water distribution systems have become a priority target in cyber warfare because of their recent computerization and introduction to the internet. As a result, Supervisory Control and Data Acquisition (SCADA) system security has become a hot topic in academic and industrial research. Among these topics, Remote Attestation is a security method intended to detect the presence of fileless malware in remote devices as they continue to operate. This allows for the detection of malware in the absence of long-term storage artifacts before symptoms of compromise begin to appear. In general, a trusted device (the verifier) makes a request for evidence of innocence from the untrusted device (the prover). In software-based schemes, the verifier can then measure the delay between its request and the prover's response. If this delay is greater than the known computational time of the evidence gathering algorithm performed by the prover, then evidence may have been forged. Multi-hop networks often introduce too much network jitter to allow accurate measurement of prover response time, which limits the effectiveness of software based Remote Attestation in a real-world setting. In this work, we introduce a companion device that the verifier can trust to perform a subset of attestation, thereby removing any network jitter. This device is a Field Programmable Gate Array (FPGA) that is physically connected to the prover. We provide a communication protocol between the verifier, prover, and companion. To evaluate our scheme, we simulate it in a common SCADA network environment under normal and heavy traffic loads. Our simulations are performed in the discrete event network simulator NS-3, and we perform statistical analysis over our results to show that our scheme allows for tight timing constraints to be placed on the prover such that the verifier can more easily determine the validity of the evidence that it receives.

Keywords: Remote Attestation, SCADA Security, Fileless Malware

1. Introduction

Supervisory Control and Data Acquisitions (SCADA) networks that control critical infrastructure such as power grid, manufacturing, and water distribution systems have become high value target in cyber warfare because of their recent computerization. Farwell & Rohozinski (2011) discuss the dangers and implications of the Stuxnet worm that targeted the Iranian nuclear program. Ten et al. (2018) cover the complexity of the Ukrainian power distribution attack. Both attacks offer examples of what can happen to a SCADA system that is compromised without detection.

Traditional forensic methods are often able to detect evidence of malware on a computing system by powering the system down and investigating the device's persistent storage for malware artifacts. They can fail because of two major issues in SCADA systems. The first is that many SCADA devices cannot be powered off regularly because they have real time, mission critical tasks. The second is that fileless malware does not leave artifacts in persistent storage because it exists completely in volatile memory (RAM). As a result, SCADA system operators need a more robust way to check a SCADA device for fileless malware.

Surveys published by Abera et al (2016b), Arias et al. (2018) and Steiner & Lupu (2016) over Remote Attestation define it as a security service that allows a trusted device (the *verifier*) to check a remote device (the *prover*) for malware. In general, the service begins when the verifier issues a challenge to the prover. The challenge should incorporate some randomness and is usually a request for system state information. When the prover receives the request, it checks the challenge's validity and then computes a response. The response is sent back to the verifier and is then checked for evidence of malware.

If a device is compromised, malware must be assumed to be in possession of all resources available to the device, including secret keys. As a result, Remote Attestation schemes need a root of trust in the prover to know that any evidence provided is not forged. According to Abera et al (2016b), Arias et al (2018) and Steiner & Lupu (2016), Remote Attestation schemes can be categorized into one of three roots of trust. (1) Hardware based methods provide a security co-design that ensures system state information will be accurately recorded. (2) Software based designs typically compare the prover's delay with the expected computational time of the response calculation algorithm to determine if malware has interfered. (3) Hybrid schemes typically utilize pre-existing security co-processors like the Intel Software Guard Extension (SGX) or the ARM Trustzone.

Hardware based schemes offer excellent security for SCADA systems, but may not apply to existing devices because they expect to be implemented in future manufacturing. Similarly, current hybrid schemes can offer protection to specific devices, but no current hybrid scheme can be applied. Under current hybrid schemes, SCADA system operators would be constrained to a small group of devices and may be forced replace several devices. Software based schemes are applicable to any device because they rely on timing constraints and not hardware. This leads to a variety of security vulnerabilities, however. For example, if the prover and verifier are multiple hops away from each other in the network (which is often the case) the delay between them may vary regularly depending on the network traffic. Because of a longer delay under heavy traffic conditions, the verifier may infer that that the prover is compromised even though the prover may be innocent. To complicate the issue, malware may be able to predict a challenge, and either remove itself before system state information is collected or precompute a forged system state. According to Steiner & Lupu (2016), tighter timing constraints must be placed on the prover's response time, and randomness must be introduced into the challenge to prevent precomputation and Time of Check Time of Use (TOCTOU) attacks.

The primary contribution of this work is the introduction of a hybrid root of trust that combines the timing components of a software based approach with the assistance of a companion device connected directly to the prover. Our approach is deployable on existing SCADA hardware of any manufacturer or family and provides stronger security guarantees than existing hybrid and software remote attestation schemes. Our companion device allows for the inclusion of cryptographic methods that defend against common attacks such as Denial of Service (DoS) and replay attacks. The timing component mitigates pre-computation attacks, and verifier control over the protocol allows randomness to defend against other known attacks in remote attestation schemes. Our scheme is designed to be implemented with a low power companion device like the tinyFPGA that is powered by the prover. The low power consumption of the companion helps to mitigate attacks that affect the prover's power supply. We evaluate our scheme using the NS3 network simulator and give statistical analysis of our results.

The remainder of this paper is organized as follows: Section 2 discusses previous works. Section 3 gives a detailed overview of our proposed scheme. Section 4 covers the details of our experimentation. Section 5 gives an analysis of our results, and Section 6 concludes the paper.

2. Previous work

Remote Attestation for embedded devices like those found in SCADA networks has been heavily studied in recent years. Early examples like those discussed by Seshadri et al. (2004) and AbuHmed et al. (2009) often used timing information. Known information such as the maximum computational time of the response generation algorithm is compared to the delay of the prover's response. If the prover's response is later than expected, verifiers may be able to assume that the response is forged. Schemes that utilize timing information are often referred to as software based.

Li et al. (2015) and Steiner & Lupu (2016) offer compelling evidence that software-based schemes have a variety of weaknesses. They discuss common attacks such as pre-computation, memory substitution, and TOCTOU. Pre-computation attacks are performed by predicting a challenge, and forging evidence in advance. Memory substitution attacks are performed by scanning a different, honest section of memory. TOCTOU attacks are performed by predicting the time of a challenge, removing malware in advance, and re-installing it after the challenge is complete. A common theme of these attacks is predictability. If the guilty prover can predict either the contents or timing of the challenge, then it can subvert detection.

Another major pitfall of software-based schemes is their reliance on strict timing information. Network jitter can introduce a great deal of variance in the delay of a response. Unforeseen events like downloading large software updates, data streaming, and increased hosts can all contribute to increased delays that have nothing to do with the honesty or guilt of a prover. Attempts have been made to reduce the effects of network jitter. Authors that target swarms of devices such as Kuang et al. (2019) and Ammar et al. (2018) are able to leverage the swarm devices' closeness to each other and their similar hardware and software components to reduce the effects of network jitter by allowing swarm devices to attest each other. It may be possible to reduce the effects of network jitter for wireless sensors by estimating network conditions and sending several small challenges from the verifier to the prover. Steiner & Lupu (2019) argue that Round-Trip Time (RTT) estimates allow them to send several challenges from the verifier with a controlled time out. The prover should be able to respond to at least one challenge before the time out of that challenge. The authors show the success of this approach with statistical analysis and argue that they are still able to include randomness in their scheme by forcing each challenge to include a different nonce. Neither of these two approaches are universally applicable, however. Not all embedded devices act as links to other devices, so they are not close enough to each other to apply a swarm solution. A group of small challenges may be able to reduce the effects of network jitter on the prover's response, but a malicious prover that receives several challenges can use the extra time between the first and last challenge to perform a TOCTOU attack.

As a result, other roots of trust have been proposed in remote attestation. Hardware based schemes like ATRIUM by Zeitouni et al. (2017), C-FLAT by Abera et al. (2016a), and OAT by Sun et al. (2020) propose hardware co-designs to be implemented in future embedded systems. These schemes can offer a greater amount of security because they have access to processor resources such as caches and busses, and they can leverage memory that is not accessible to the processor of the prover. This greater security does not come without a cost, however. Their solutions are intended to be applied to future embedded devices, so they are unable to protect any devices that have already been manufactured. Hardware co-designs are by nature unable to update. If attacks like Spectre discussed by Koruyeh et al. (2018) or Meltdown discussed by Lipp et al. (2018) were discovered in a hardware co-design, then devices with the design could be rendered obsolete.

Hybrid schemes designed to bridge the gap between hardware and software have become popular in recent years. Typically, these schemes rely on security co-processors that already exist on modern computing devices like those discussed by Maene et al. (2017). They can leverage access to protected memory that is not accessible to the prover, which allows them to utilize cryptographic keys and other secret information. Hybrid schemes have similar shortcomings as hardware schemes, however. They are generally able to update in the case that a vulnerability is found, but they are reliant on specific hardware features. These features are not consistent across all security co-processors. For instance, the operation and architecture of the Intel SGX are quite different from those of ARM Trustzone. If a vulnerability is found in the security co-processor of a specific device, the device can be rendered obsolete and the scheme that relies on it may not be easily ported to a new security co-processor.

These issues motivate us to design a hybrid approach that combines the timing components of a software based approach with an additional companion device. Such a scheme could solve the problem of network jitter and protect against known attacks with randomness. We choose this approach because it is easily updated, is not reliant on any specific hardware, and would be broadly applicable over a variety of devices while offering a more robust defence against fileless malware.

3. Companion assisted attestation

Our approach to reduce network jitter and allow for tighter timing constraints on the prover response is to introduce a new entity into our network: the companion. This device is an FPGA that communicates with the prover via a dedicated line like USB. The verifier can communicate with the companion through the prover. A detailed overview of our protocol is shown in Figure 1.

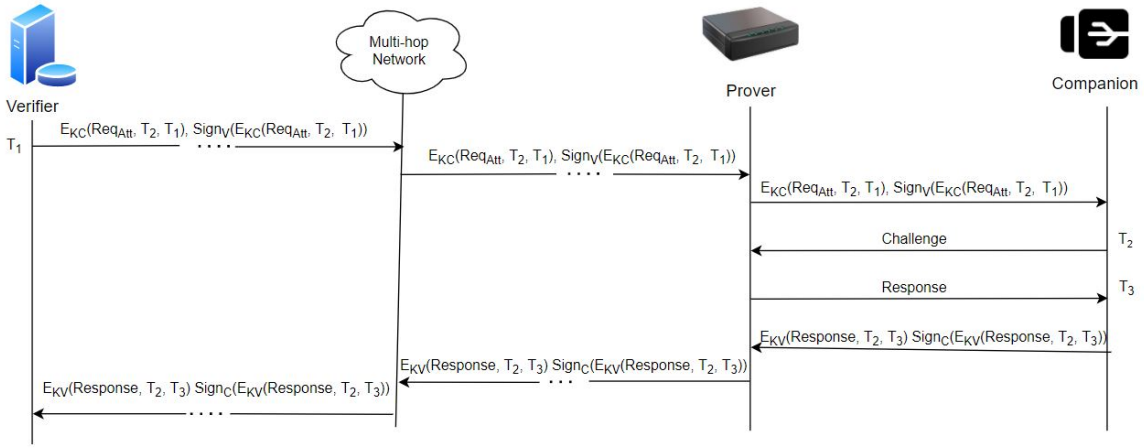

Figure 1: Proposed companion assisted software-based remote attestation protocol

Our protocol begins when the verifier sends a request to the companion for an attestation challenge at a specified, random time (T_2). This request is sent with a timestamp (T_1), and it is encrypted with the companion's public key (K_C). A signature ($Sign_V$) is also generated over the message. In this way we prevent denial of service attacks as well as replay attacks. Once the prover receives the message, it forwards it on to the companion immediately. If it does not, additional delays may be incurred, and the verifier will suspect it of compromise. When the companion receives the message, it checks its authenticity and waits until the appropriate time to issue the challenge. When the challenge is issued, the exact time is recorded, and the companion awaits the prover's response. When it arrives, the companion bundles the response, the time requested (T_2), and the time received (T_3). It then encrypts this information with the verifier's public key (K_V), and signs it ($Sign_C$). This message is then sent to the prover who immediately passes it to the verifier. The verifier is then able to examine the response given by the prover within the context of the timing information collected by the companion.

The introduction of the companion device provides a variety of benefits. First, attestation occurs over a single hop on a dedicated line. This eliminates network delays that allow enough ambiguity for compromised prover to forge evidence. Second, the verifier controls the randomness and timing of the challenge, and neither are known to the prover prior to the challenge. This renders precomputation, memory copy, compression, and TOCTOU attacks ineffective. Third, the companion device is an FPGA, so it is harder to compromise than a device with a multi-purpose processor, and it is reprogrammable in the case that some aspect of our protocol is found to be vulnerable to attack. Finally, the companion is only reachable through the prover device by authorized entities via public-private key interactions.

4. Experimental setup

We simulated our proposed protocol using the discrete network simulator NS3. NS3 is open source, and natively supports simulations of TCP/IP and UDP communications. It allows users to define nodes and the links between them. It also allows users to build and install applications onto nodes that define their network behaviour. NS3 natively supports TCP/IP and it allows for development of additional protocols.

4.1 Testing environment

We defined a two-layer SCADA network as discussed by Barbosa, Ramin, & Pras (2012) to be our test network. Our top layer, the IT infrastructure, was made up of 4 entities: two host devices, a single server, and a router. The second layer, the production infrastructure, was defined to be 5 devices: three Programmable Logic Controllers (PLCs), a server, and a human machine interface. We chose this number of devices so that we could model a much larger network by balancing network traffic between different entities. PLC_2 acted as our prover device and was connected directly to the companion device. Our test network can be seen in Figure 2.

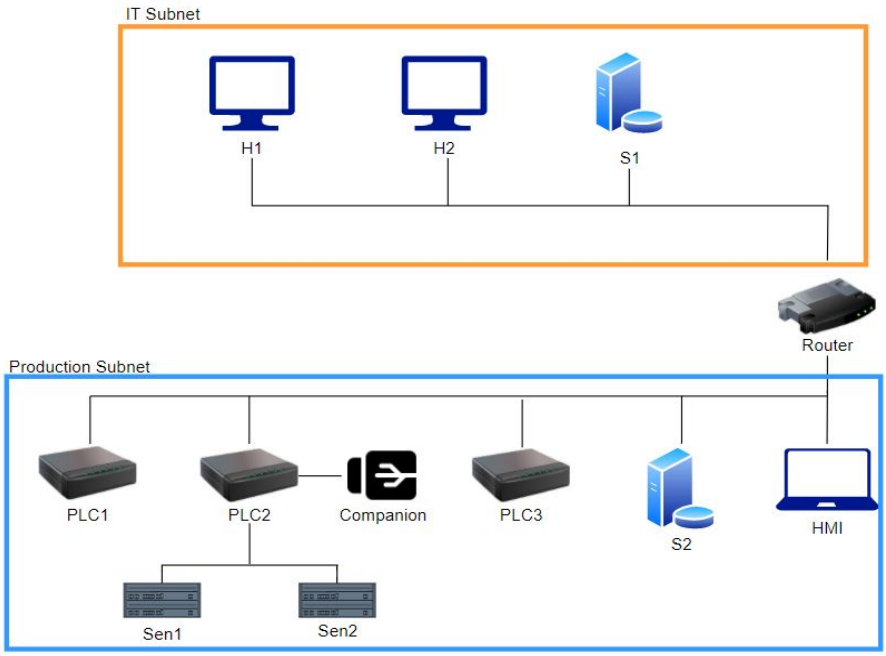

Figure 2: Test network architecture

The connections between the IT network and the router, the production network and the router, and PLC2 to its sensors are all simple ns3 Carrier Sense Multiple Access (CSMA) busses. These busses model an ethernet connection, although they do not provide simulations for the physical layer. The data rate of each connection was 700 Mbps with a 2 msec delay. The connection between PLC2 and the companion is a point-to-point 400 Mbps connection with a 2 msec delay intended to model a USB2 connection.

To simulate network traffic, we used the tcp_echo_client and tcp_echo_server applications native to NS3. The client simply issues a TCP echo request of programmable size, and the server responds with the same packet. To simulate Modbus traffic, we used the source code modbus_tcp_client and modbus_tcp_server provided by Sahraei (2013) in their study of integrating Modbus with NS3. We defined traffic flows to balance overall network traffic evenly across the network. Our goal was to show the overall effects of traffic. The total traffic generated in our IT subnetwork accounts for 1/3 of all traffic in the network. Similarly, the production network accounts for another 1/3. The communication between S_1 and S_2 accounts for the final 1/3. Each flow's overall traffic was controlled by the delay and packet size of the client's request. We added randomization to each client by calculating a uniform random variable with upper bound (delay + delay/2) and lower bound (delay – delay/2). In all tcp_echo_client/server cases, we set the packet size to 100. The delays are given in Table 1 with each flow.

Table 1: Flow information with protocol and request delay , where T is the total traffic in bytes per second

Client	Server	Protocol	Delay
H_1	S_1	tcp_echo_client/server	600 / T sec
H_2	S_1	tcp_echo_client/server	600 / T sec
PLC_1	S_2	tcp_echo_client/server	900 / T sec
PLC_2	S_2	tcp_echo_client/server	33.34 msec
PLC_3	S_2	tcp_echo_client/server	900 / T sec
HMI	S_2	tcp_echo_client/server	600 / T sec
PLC_2	Sen_1	modbus_tcp_client/server	100 msec
PLC_2	Sen_2	modbus_tcp_client/server	100 msec

We did not allow PLC_2 traffic between the corporate networks and the sensors to vary with the overall amount of traffic. This is because PLC_2 acts as the prover in our case, and it is intended to model a single device's traffic. The other devices in the network were designed to generate the traffic of a much larger network and are acting as several devices.

4.2 Testing cases

To test our proposed protocol, we studied several variables. We implemented our companion attestation protocol as well as a direct attestation, in which the verifier sends a challenge directly to the prover. To show the effects of traffic on both protocols, we defined three traffic cases: a medium, a high, and a burst. The medium case was 15 MBps. The high case was 66 MBps. The burst case 100 MBps. These values were taken from traffic analysis done by Maglaras & Jiang (2014). We also chose to vary the location of the verifier to see the effects of distance over a busy network. The first location is S_1 and the second is S_2. Finally, we implemented a prover application that could model both honest and dishonest delay behaviour. We chose to use the prover behaviour characteristics studied by Gardner, Garera, and Rubin (2007) shown in Table 2. They implemented an honest device and a compromised device that would perform a memory copy attack and studied the delay characteristics of each. We chose these values to represent our honest and guilty prover because the authors perform their attack on a computationally constrained device that is somewhat similar to what might be available in a SCADA network.

Table 2: Honest and guilty prover delay behaviour

Prover Type	Mean	Standard Deviation
Honest	1873.23 s	0.110
Guilty	1876.19 s	0.000232

Our verifier application sends a single 100-byte packet to a programmable destination and awaits a response. The companion application awaits a message from the verifier, and then forwards it to the prover. It then awaits a response from the prover to forward back to the verifier. Table 3 shows the variables we studied in this work.

Table 3: Overview of the variables tested in our scheme. A test was performed for each combinations of variables seen here.

Variable	Values
Protocol	Direct Attest, Companion Attest
Prover Honesty	Honest, Guilty
Verifier Location	S_1, S_2
Traffic	Medium, High, Burst

5. Results and analysis

For each combination of variables discussed in Table 3, the random seed of NS3 was set to 1. We measured the prover's response delay over 100 runs, each with a different NS3 run number. Figure 3 gives the resulting graphs of delays for attestation from S1, and Figure 4 gives the graphs for attestation from S2.

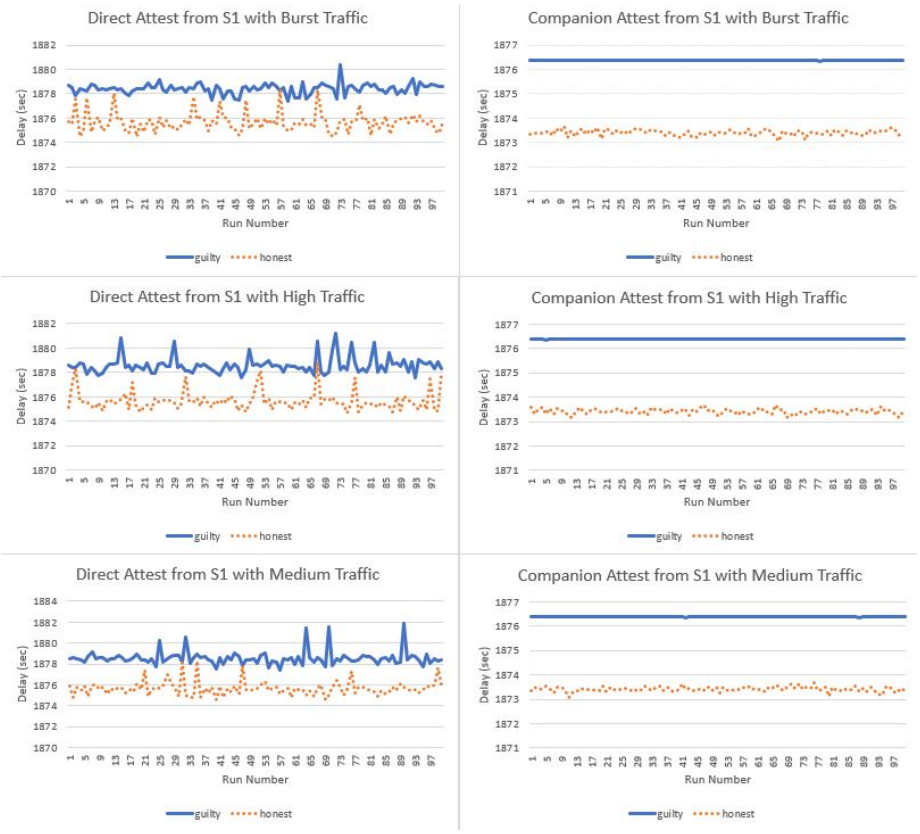

Figure 3: Graphs of prover response delay with verifier at S_1 over NS3 run numbers from 1 to 100.

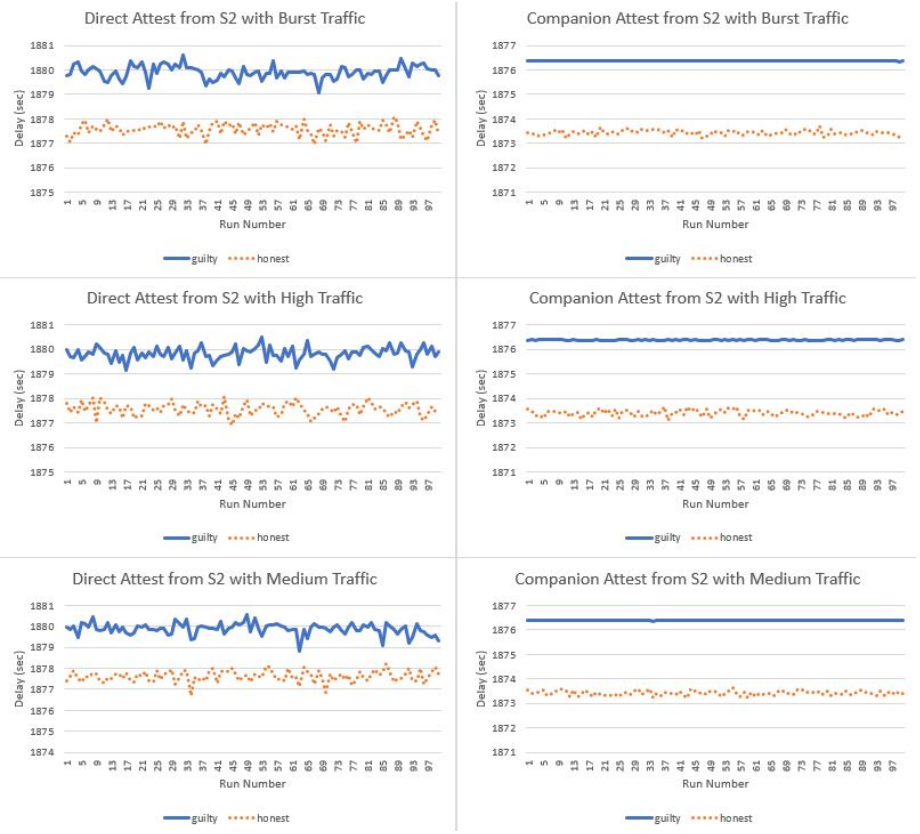

Figure 4: Graphs of prover response delay with verifier at S_2 over NS3 run numbers from 1 to 100.

In each case, the left column of graphs gives the direct attest, while the right gives the companion attest. The rows indicate different traffic levels from medium to burst. The solid line in each graph represents the resulting delays when the prover is guilty. The dotted line in each graph represents the resulting delays when the prover is innocent.

A variety of observations can be made from these graphs. In the direct attestation case, the prover delay is sensitive to traffic changes as well as distance between the prover and verifier. This allows for a greater degree of ambiguity between the guilty prover case and the honest prover case. When attestation challenges are made from S_1, the honest prover and the guilty prover trendlines overlap, indicating that false positives may be more common. Conversely, companion attestation has a much weaker reaction to increased traffic in the network and a greater distance between the verifier and the prover. Visual inspection of all companion assisted attestation graphs indicate a clear difference between honest and guilty behaviour.

To confirm the visual inspection of our results, we first performed an Anderson Darling Normality Test as discussed by Razali & Wah (2011). This test allows us to determine whether the distribution of prover responses under different circumstances is normal. Table 3 shows the results as well as the mean and standard deviation for each test.

Table 3: Anderson Darling Normality Test p-values, means and standard deviations

				P-Value	Mean	Std. Dev.
Direct	S1	Burst	Guilty	<0.0005	1878.409	0.432
			Honest	<0.0005	1875.677	0.804
		High	Guilty	<0.0005	1878.573	0.673
			Honest	<0.0005	1875.676	0.796
		Medium	Guilty	<0.0005	1878.562	0.698
			Honest	<0.0005	1875.657	0.679
	S2	Burst	Guilty	0.670819	1879.905	0.269
			Honest	0.286916	1877.573	0.252
		High	Guilty	0.183584	1879.829	0.261
			Honest	0.310729	1877.558	0.257
		Medium	Guilty	0.006581	1879.882	0.279
			Honest	0.359948	1877.627	0.264
Companion	S1	Burst	Guilty	0.037053	1876.385	0.00281
			Honest	0.517268	1873.403	0.114
		High	Guilty	<0.0005	1876.386	0.00300
			Honest	0.272961	1873.422	0.115
		Medium	Guilty	0.000577	1876.386	0.00299
			Honest	0.345178	1873.413	0.101
	S2	Burst	Guilty	0.000840	1876.385	0.00304
			Honest	0.590245	1873.435	0.101
		High	Guilty	0.001228	1876.386	0.00288
			Honest	0.761777	1873.405	0.126
		Medium	Guilty	0.0171116	1876.385	0.00299
			Honest	0.970431	1873.419	0.094

Using an alpha level of 0.05, we can conclude that any sample with a p-value above 0.05 comes from a normal distribution. This indicates that all samples that come from a simulation with an honest prover and a companion assisted attest are from a normal distribution. Furthermore, the mean and standard deviation are very close to the true prover behaviour. From this we can conclude that our scheme closely captures the behaviour of an honest prover.

In the case of the direct attestation from S_1, results are poor. Sample means differ from prover behaviour by whole seconds, and standard deviations are substantially larger than prover behaviour. It is of note that attesting from S2 performs somewhat better. Our results show that each sample from the burst and high traffic

cases as well as the honest medium case come from a normal distribution. Their means and standard deviations are substantially different from the prover behaviour, however.

6. Conclusion and future work

In this work, we introduce a new device, the companion, into a software based remote attestation scheme. With the addition of this device, we can show promise that network jitter might be greatly reduced, and that a software root of trust might be more feasible in real world settings. We simulated a 2-layer SCADA network with both Modbus and TCP traffic in NS3 to evaluate our scheme. We defined 4 testing variables to give a more complete picture of how our scheme compares against a direct software based remote attestation protocol. Using visual inspection and statistical analysis, we can show that our protocol closely captures the behaviour of an honest prover device. Using our protocol, security designers and SCADA administrators may be able to leverage a software-based root of trust in a SCADA network.

In the future, we will likely investigate the effects of a more realistic, larger SCADA network on our protocol as well as additional guilty prover behaviour models. We will also likely implement our protocol using a real-world FPGA as the companion device to research the hardware and power requirements of our scheme. Finally, we will continue to research methods to improve software-based remote attestation schemes.

7. Acknowledgements

We would like to thank the CyberCorps Scholarship for Service as well as the Tennessee Tech University Cybersecurity Education Research and Outreach Center (CEROC) for the vital support they gave for this project in these difficult times.

This manuscript has been authored by UT-Battelle, LLC, under Contract No. DE-AC0500OR22725 with the U.S. Department of Energy. The United States Government retains and the publisher, by accepting the article for publication, acknowledges that the United States Government retains a nonexclusive, paid-up, irrevocable, worldwide license to publish or reproduce the published form of this manuscript, or allow others to do so, for the United States Government purposes. The Department of Energy will provide public access to these results of federally sponsored research in accordance with the DOE Public Access Plan. (http://energy.gov/downloads/doe-public-access-plan). The authors are with the Oak Ridge National Laboratory, Oak Ridge, TN 37831 USA.

8. References

Abera, T., Asokan, N., Davi, L., Ekberg, J.E., Nyman, T., Paverd, A., Sadeghi, A.R. and Tsudik, G., 2016a, October. C-FLAT: control-flow attestation for embedded systems software. In Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security (pp. 743-754).

Abera, T., Asokan, N., Davi, L., Koushanfar, F., Paverd, A., Sadeghi, A.R. and Tsudik, G., 2016b, June. Things, trouble, trust: on building trust in IoT systems. In Proceedings of the 53rd Annual Design Automation Conference (pp. 1-6).

AbuHmed, T., Nyamaa, N. and Nyang, D., 2009, November. Software-based remote code attestation in wireless sensor network. In GLOBECOM 2009-2009 IEEE Global Telecommunications Conference (pp. 1-8). IEEE.

Ammar, M., Washha, M., Ramabhadran, G.S. and Crispo, B., 2018, December. slimiot: Scalable lightweight attestation protocol for the internet of things. In 2018 IEEE Conference on Dependable and Secure Computing (DSC) (pp. 1-8). IEEE.

Arias, O., Rahman, F., Tehranipoor, M. and Jin, Y., 2018, March. Device attestation: Past, present, and future. In 2018 Design, Automation & Test in Europe Conference & Exhibition (2018) (pp. 473-478). IEEE.

Barbosa, R.R., Sadre, R. and Pras, A., 2012, March. Difficulties in modeling SCADA traffic: a comparative analysis. In International Conference on Passive and Active Network Measurement (pp. 126-135). Springer, Berlin, Heidelberg.

Farwell, James P & Rohozinski, Rafal, 2011. Stuxnet and the Future of Cyber War. Survival (London), 53(1), pp.23–40.

Gardner, R.W., Garera, S. and Rubin, A.D., 2007. On the Difficulty of Validating Voting Machine Software with Software. EVT, 7, pp.11-11.

Koruyeh, E.M., Khasawneh, K.N., Song, C. and Abu-Ghazaleh, N., 2018. Spectre returns! speculation attacks using the return stack buffer. In 12th {USENIX} Workshop on Offensive Technologies ({WOOT} 18).

Kuang, B., Fu, A., Yu, S., Yang, G., Su, M. and Zhang, Y., 2019. ESDRA: An efficient and secure distributed remote attestation scheme for IoT swarms. IEEE Internet of Things Journal, 6(5), pp.8372-8383.
Li, Y., Cheng, Y., Gligor, V. and Perrig, A., 2015, March. Establishing software-only root of trust on embedded systems: facts and fiction. In Cambridge International Workshop on Security Protocols (pp. 50-68). Springer, Cham.
Lipp, M., Schwarz, M., Gruss, D., Prescher, T., Haas, W., Fogh, A., Horn, J., Mangard, S., Kocher, P., Genkin, D. and Yarom, Y., 2018. Meltdown: Reading kernel memory from user space. In 27th {USENIX} Security Symposium ({USENIX} Security 18) (pp. 973-990).
Maene, P., Götzfried, J., De Clercq, R., Müller, T., Freiling, F. and Verbauwhede, I., 2017. Hardware-based trusted computing architectures for isolation and attestation. IEEE Transactions on Computers, 67(3), pp.361-374.
Maglaras, L.A. and Jiang, J., 2014, August. Intrusion detection in SCADA systems using machine learning techniques. In 2014 Science and Information Conference (pp. 626-631). IEEE.
Razali, N.M. and Wah, Y.B., 2011. Power comparisons of shapiro-wilk, kolmogorov-smirnov, lilliefors and anderson-darling tests. Journal of statistical modeling and analytics, 2(1), pp.21-33.
Sahraei, M.R., 2013. 'Ns-Modbus: Integration of Modbus with ns-3 network simulator', MS thesis, Islamic Azad University, Iran.
Seshadri, A., Perrig, A., Van Doorn, L. and Khosla, P., 2004, May. SWATT: Software-based attestation for embedded devices. In IEEE Symposium on Security and Privacy, 2004. Proceedings. 2004 (pp. 272-282). IEEE.
Steiner, R.V. and Lupu, E., 2016. Attestation in wireless sensor networks: A survey. ACM Computing Surveys (CSUR), 49(3), pp.1-31.
Steiner, R.V. and Lupu, E., 2019. Towards more practical software-based attestation. Computer Networks, 149, pp.43-55.
Sun, Z., Feng, B., Lu, L. and Jha, S., 2020, May. OAT: Attesting operation integrity of embedded devices. In 2020 IEEE Symposium on Security and Privacy (SP) (pp. 1433-1449). IEEE.
Ten, Chee-Wooi et al., 2018. Impact Assessment of Hypothesized Cyberattacks on Interconnected Bulk Power Systems. IEEE Transactions on Smart Grid, 9(5), pp.4405–4425.
Zeitouni, S., Dessouky, G., Arias, O., Sullivan, D., Ibrahim, A., Jin, Y. and Sadeghi, A.R., 2017, November. Atrium: Runtime attestation resilient under memory attacks. In 2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD) (pp. 384-391). IEEE.