Secure Remote Access to Industrial Control Systems with Mobile Devices

Laurens Lemaire, Jan Vossaert, Vincent Naessens
KU Leuven, Belgium

Abstract

Industrial control systems are increasingly often connected to corporate networks and the internet, allowing users to remotely control and monitor them. This greatly improves the usability, but it also introduces additional security concerns.

This work investigates the security considerations that must be taken into account when remotely accessing these control systems. Four different remote access architectures are proposed and compared with each other with regards to security. One of these is implemented in practice and thoroughly tested. The implemented architecture includes a secure connection, as well as user, server, and device authentication. For our case study we consider an industrial generator, a stand-alone power generator that produces energy from burning rapeseed oil. An embedded device is connected to this generator using a serial connection. A Wi-Fi dongle is attached to allow remote access.

1. Introduction

Nowadays a lot of electronic devices are connected to the internet. The term "Internet of Things (IoT)" is commonly used to refer to this trend. Industry is not staying behind in this regard. Increasingly often, machines and installations are controllable over the internet. Current industrial control systems are supplied with Ethernet ports or wireless support. This improves the ease of use of said systems.

A problem with this evolution is the additional security concerns that come with it. In the past if an attacker wanted to sabotage an isolated control system, he had to obtain physical access to the machine. But now, if it is possible to remotely connect to a device, it is also possible to remotely attack it. Connecting these systems to the internet also attracts new kinds of attackers, for example script kiddies who just want to test their powers as hackers [1], [2].

In recent years, industrial control systems (ICS) have increasingly often been targeted by cyber-attacks. Especially in the energy sector [3]. Due to the nature of these systems, attacks can have catastrophic consequences. Previous attacks on ICS illustrate this. Famous examples are the Maroochy Shire sewage spill in Australia [4], and the Stuxnet worm in Iran [5], [6]. The former caused 800.000 litres of raw sewage to spill into local parks and rivers, the latter was used to sabotage the fuel enrichment plant of Natanz in Iran [7].

Typical IT security solutions are often not applicable to industrial control systems. Because of their critical nature, additional or different requirements such as high determinism and response times are introduced. Reliability of the network is more important than in most IT applications. For these reasons, applying patches to fix vulnerabilities is not always possible, especially if they require a reboot of the system [8]. Patch management is often an important aspect of maintaining ICS.

It is clear that a decent security analysis is required to obtain a system that is resistant against as many attacks as possible whilst still being easy to use.

Contribution. In this article we provide a security analysis of remote access to a specific industrial control system, the iGenerator at our university campus. It should be possible for mobile devices such as smartphones or tablets to connect to this generator remotely. Users will be able to read parameters and control the generator. Four architectures for establishing a remote connection are investigated and compared in terms of security. One of these has been implemented and further evaluated.

Outline. The structure of this article is as follows. Section 2 contains some background information about the case study and the used set-up. It also suggests alternative set-ups and discusses which attacks we wish to defend against. Section 3 introduces some technologies and tools that are used in our architectures. Section 4 lists the four remote access architectures. In Section 5 we discuss the implementation of the chosen architecture. Section 6 evaluates the security of the implementation. Finally, Section 7 concludes the article and contains future work.

2. Background

In this section we explain the case study and the set-up that is used for our architectures in detail. We also provide some alternative set-ups that the user may consider, and discuss possible attacks that the architectures should protect against.

2.1. Case study

A company rents out electricity generators to building sites, festivals, etc. They adopt the idea to make it possible to remotely control these generators using tablets and smartphones. These mobile devices are delivered to the customer at the same time as the generators. The communication between these mobile devices and the generator must be secure, this article investigates some ways to achieve this.

The architectures are tested on the iGenerator at our university campus. This generator already contains a control system that can be controlled through a serial cable or touch panel. This is the InteliLite NT MRS 11. In order to make remote access possible, an embedded board is connected to the MRS using this serial connection. A Wi-Fi dongle is then attached to this embedded device. The embedded board can communicate with the generator control system using the Modbus protocol.

The board we decided to use is the Freescale SABRE Lite development board. It contains a quad core ARM Cortex A9 processor and has 1 GB RAM, these specs are suitable for our needs.

Using their mobile devices, users will be able to check and modify parameters of the generator. It is also possible to switch the generator on or off remotely. Users will have to authenticate themselves in order to perform these actions. There are two types of users, the first type can only read the parameters, whereas the second type has more privileges and can take all actions.

We also consider two types of attackers: An internal attacker who is trying to obtain full access so he can shut down the generator, and an external attacker who has no privileges.

2.2. Alternative set-ups

In this section we will briefly go over some alternative set-ups and their advantages or disadvantages over the chosen method.

The **VoCore** is a small chip that has a Wi-Fi module added to it [9]. The VoCore removes the need for a Wi-Fi dongle, and is a lot cheaper and smaller than our embedded board, however the specs are quite a bit worse. The processor runs at 360 MHz and the chip has 32 MB RAM. If these specs are sufficient for your application, the VoCore is a nice alternative.

Other possibilities for secure remote access include **firewall software** such as pfSense [10]. Firewall rules can be used to restrict access to certain devices in a network. However, our case study only contains one device. For our case it is more useful to be able to restrict certain commands, based on the user permissions (role-based access control), and using our embedded board gives us this option.

Industrial routers can also be used in certain set-ups. These allow users to establish VPN connections into remote networks. Examples include the Phoenix mGuard. For our small case study, VPN was overkill. In addition, the generator and the embedded board do not require a power source, whereas the router would need some workarounds or a power source to be used.

A final alternative is a **cloud solution**, for example Siemens Sinema Remote Connect [11], or eWON. Here the user logs in to the machine using a secure connection through the cloud. It requires a router to be connected to the control system.

2.3. Possible attacks

In order to be able to secure the system against attackers, we must first consider which attacks we wish to defend against. The main objective of an attacker will be to control the generator without being authorized to do so. There are several ways to achieve this goal, for example stealing passwords from legitimate users, or hijacking their sessions.

Here are some attacks we want to defend against:

- Eavesdropping. It should not be possible for an attacker to find out the content of messages, otherwise he/she could learn the passwords of users.
- Replay attack. A replay attack occurs when an attacker saves a sent package and sends it again later.
- Tampering. An attacker should not be able to change the content of a message without being noticed.
- Denial of Service (DoS). A DoS attack overloads the network with messages to cause a drop in performance or stop communications altogether.
- Dropping packets. When packets go missing, the sessions should end or the communication should be halted until the missing packet has arrived.

Each architecture will contain a database with user passwords. There are some security concerns relating to this database as well. It should be required of users to create strong passwords such that brute force attacks and dictionary attacks are not successful. Social engineering is another possible attack to obtain passwords, unfortunately this is quite hard to defend against. Social engineering occurs when a user is somehow tricked into disclosing their login information to the attacker. This happens mostly when we are considering the internal adversary. If an architecture offers additional protection against social engineering, this is a plus.

From the above, we can identify security requirements on the server side, and the communication link between client and server. The client side is the hard part to secure. In order to make this easier the following security requirements are assumed:

- Only the client and the company renting out the generator know the passwords.
- When the generator is moved to a new client, all passwords are changed.
- User profiles are created that tie different privileges to different accounts.

In Section 6 we will evaluate how the suggested architectures fare against the above attacks.

3. Used technologies

Here we give some brief information about the technologies and protocols that are used or analysed in our architectures and implementation.

3.1. Modbus

Modbus is an open source communication protocol used for serial connections. It is easy to deploy and maintain and has become a standard communication protocol. The iGenerator is controlled using Modbus commands.

A Modbus frame is composed of an ADU (Application Data Unit), which has a PDU (Protocol Data Unit) enclosed. The PDU contains a function code and a data field. The function code allows the recipient to know what kind of command needs to be done, in other words what to do with the data. The ADU contains the address of the receiver, the PDU, and a checksum. The checksum ensures the data has not been tampered with. CRC (Cyclic Redundancy Check) is used for this purpose [12]. If the checksum fails, the data has to be sent again.

3.2. Jetty

Jetty is an open source Java webserver and Servlet container. It can be used in one of two ways: As a standalone application or directly integrated in code. In our implementation we will take the latter approach. The Jetty server is located on the embedded device.

Jetty was founded in 1995 and has been moved to the Eclipse foundation as of Jetty version 9.

3.3. Shibboleth

Shibboleth is a single sign-on system for the internet. The project started in 2000 to address resource sharing problems between organizations that employ different authentication and authorization infrastructures. In 2008, Shibboleth 2.0, the current version, was released.

Single sign-on means the user only has to log in once to gain access to several services. The advantage for the user is that he/she only has to remember one password and does not have to log in multiple times to access certain services. For the administrators, it is easier to provide security as there is only one point of entrance [13].

The procedure works as follows:

1. The user contacts the desired service. When a session is already open, the service is returned. Otherwise, the user is sent to the Service Provider (SP).
2. The SP creates an authentication request and forwards it to the Identity Provider (IdP).
3. The IdP authenticates the user and notifies the SP.
4. The SP validates the response from the IdP and creates a session. The necessary user details for establishing the session are included in the IdP response.
5. The user can now gain access to the service.

Shibboleth is not part of the implementation in this article, but it forms the basis of some of the architectures in the next Section.

4. Remote access architectures

Four different architectures are proposed, each dealing with the security requirements differently. We present the architectures below and discuss their differences as well as their strengths and weaknesses.

4.1. Architecture 1: basic security

The first architecture is quite basic, yet often used in installations nowadays. The server and the mobile device authenticate themselves to each other in order to set up a secure connection before they start communicating.

A symbolic representation of architecture 1 is show in Figure 1. When the mobile device indicates it wants to set up a connection with the server, the server sends its certificate over. The mobile device confirms if this is the correct certificate and when this is the case a secure channel is set up. The user sends his login and password over this channel to authenticate himself to the server. The server then looks in its database to check these details. In case the wrong details are sent, the user will be able to try again up to 3 times before the secure connection is terminated. When the user successfully logs in, data exchange can commence.

Figure 1. Symbolic view of architecture 1

4.2. Architecture 2: device authentication

In the second architecture, there will be three different instances of authentication: The server and

the user will have to authenticate themselves to each other as in the previous example, and this time the

mobile device will also need to be authenticated. When any of these authentications fail, there is no mutual trust and the communications should cease.

Figure 2 shows how this happens in practice. First, a secure channel is set up between the server and the mobile device. This channel is set up using a handshake after both parties have authenticated themselves to each other. Certificates are used for this authentication. The mobile device knows to trust the certificate of the server. The server has a list with the certificates of all the mobile devices it trusts. When setting up the channel, both parties send over their certificates and if these are trusted by the other party, secure communication is initiated. The user then authenticates himself to the server using a login and password combination.

This architecture has been implemented in practice, the details of this can be found in the next Section.

4.3. Architecture 3: authentication server

The third architecture focuses more on the user authentication. It is largely based on the Shibboleth model which is discussed in the previous Section. A separate authentication server is used in this architecture.

Figure 3 shows how this architecture works. To start things off, the mobile device will initiate contact with the generator server. If there is no active session, the server will send a challenge back to the user. The user then forwards this challenge together with his authentication details to the authentication server. This server checks whether the user is authorized to utilize the generator services. When this is the case, the authentication server will sign the challenge and send this together with the user permissions and the current time to the generator server. This message is encrypted to prevent eavesdroppers from finding its contents. After the generator server has verified the signature, a session will be established and data can be exchanged. Sessions expire after one hour or when the user logs off.

Figure 2. Symbolic view of architecture 2

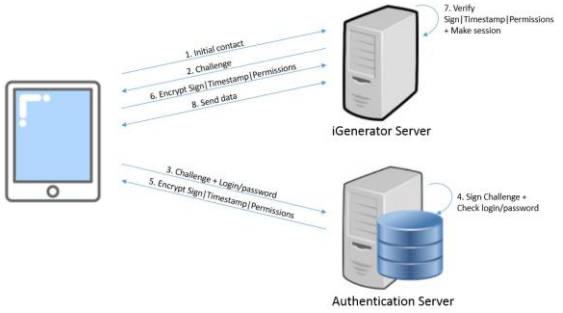

Figure 3. Symbolic view of architecture 3

4.4. Architecture 4: combining architectures 2 and 3

The final architecture combines the previous two, using a separate authentication server and implementing device authentication.

As in architecture 3, the user will first initiate contact with the generator server, which will issue a challenge to be sent to the authentication server. The mobile device certificate is now added to the authentication details that are sent to this server. The authentication server contains a set of trusted certificates in addition to the database used for user authentication. If both the user details and the mobile certificate check out, a token with the appropriate permissions is generated and forwarded to the generator. Afterwards a session is established and communication can commence.

4.5. Comparison of architectures

Here we will compare the strengths and weaknesses of the different architectures. A security analysis will happen in Section 6.

Architecture 1 is rather basic, but often used in industrial settings nowadays. The main advantage of this architecture is the ease of implementation. There is no need to configure an extra server as the password database is located on the generator server. There are several downsides. The user will have to find out the IP address of the generator server before use. A second disadvantage is that the administrator will have to go to the generator each time he wants to update the password database. A server at a centralized location would be easier to access. Thirdly, the server is running on an embedded device with limited memory and processor capacity. This could potentially turn into a bottleneck, slowing down communications. Finally, having the server on

Table 1. A comparison between the architectures

	Arch 1	Arch 2	Arch 3	Arch 4
Easy implementation	X	X		
Easy maintenance			X	X
Certificate storage		X		X
Potential bottleneck	X	X		
IP address available				
Server authentication	X	X	X	X
User authentication	X	X	X	X
Device authentication		X		X

location means that attackers can access it more easily.

These downsides are addressed in the other architectures. The second architecture adds device authentication into the mix. It has largely the same strengths and weaknesses as the first architecture, the difference being some added security. The generator server will have to keep a list of allowed mobile certificates which further limits the memory of the embedded device.

The third architecture introduces an authentication server. This has the added advantage that administration of passwords is easier to do. The problem of the limited resources of the embedded device is also solved by having a separate server. The attacker can also no longer access the database. A downside that persists is that the user has to figure out the IP server of the generator prior to use.

Architecture four combines the strengths of the third architecture with the added security of the second one. Like architecture three, it requires an additional server to be set up. However, this means it is easier to configure the database, and harder for attackers to tamper with it. As in architecture two, it uses three forms of authentication: server, device, and user authentication.

Table 1 summarizes the comparison between the architectures. It also shows which forms of authentication are supported by the different architectures.

A disadvantage for all architectures is the fact that the user has to find out the IP address of the generator server before use. A possible solution to this is the addition of a location server. Each time the generator moves, it automatically updates the IP address at the location server. The user can then request the IP from this server before he attempts to contact the generator. Another solution would be to use fixed IP SIM cards. This is touched upon further in Section 7.1.

5. Implementation

This Section is divided in two parts: First we discuss how the connection between the mobile devices and the generator is obtained, then we discuss security separately.

As mentioned in the previous Section, the decision was made to implement the second architecture, as this means we can investigate all three forms of authentication. In addition, our case study only consisted of one control system, so setting up central database management seemed overkill.

5.1. Controlling the generator remotely

Controlling the generator can be subdivided in three big tasks: Setting up a serial connection between the embedded device and the generator, setting up a Jetty server on the embedded device, and creating an application for use on the mobile devices.

5.1.1. Serial connection. An initial attempt to contact the generator using a C# program failed. It turns out the stock program was using a proprietary protocol to communicate with the generator. In order to use Modbus, an additional password had to be given. After contacting the supplier and obtaining this password, the C# program worked.

After confirming we could reach the generator over the serial connection, the real work could start. As the embedded device is running Linux, the decision was made to program in Java. This caused another issue, as Java does not contain a standard library for setting up serial connections, unlike C\# and C. We were unable to find a third-party library suitable for our needs, and hence decided to start using the Java Native Interface (JNI). JNI is a framework that ensures java code can call methods from native applications and libraries. These applications and libraries are written in a different programming language, for instance C, C++ or assembler. Using this method, we wrote a C program that sets up a serial connection independent of the operating system used.

5.1.2. Jetty server. There is ample documentation available for setting up Jetty servers. Initially we made a standard Jetty server without any security concerns. In the next Section we will modify this server to be able to handle SSL connections. Three objects are required for setting up a Jetty server:

- A *Server* object. This is the main object to which the next two will be connected.
- A *ServerConnector* object. This contains the port through which one can reach the server.
- A *ContextHandler* object. When the server is contacted, the handler in this object will deal with the request. This can range from returning some values to making some changes etc. A context can be passed to the handler, allowing the creation of separate handlers for specific links.

Now all that needs to happen is *server.start()*, followed by *server.join()*. This second method ensures operations can only continue when the server is fully started.

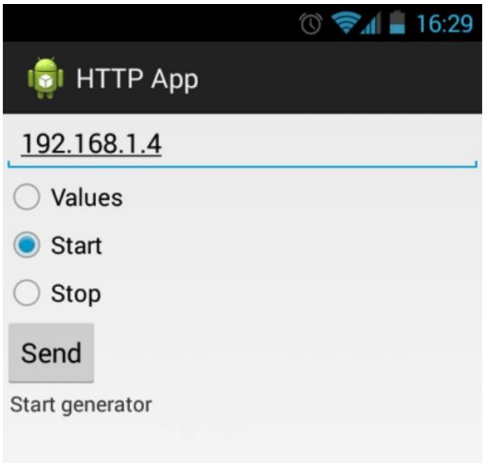

Figure 4. The Android application

5.1.3. Android application. A simple Android application has been written to remotely control the generator. Figure 4 shows a screenshot of the application. At the top, the user must enter the IP address of the generator. As generators could change location fairly frequently, this is not an ideal solution. Adding a location server will make the application easier to use.

The user can choose what to do by using the radio buttons. He can start and stop the generator, provided he has the permissions to do so, and ask for some parameter values. The 'Send' button sends the command to the Jetty server, which converts it into a Modbus command.

5.2. Securing the implementation

The second architecture contains three forms of authentication, which are explained below. Server authentication is ensured by setting up an SSL connection. The user then authenticates himself by sending his login and password over the secure connection. Device authentication was also added and is implemented by using certificates.

5.2.1. Setting up the SSL connection. To set up an SSL connection we first have to configure the server to be reachable through SSL. Then the mobile application has to be configured to send commands to the SSL port on the server.

Server: To make the Jetty server reachable through SSL, a *HttpConfiguration* object is created with a *SecureRequestCustomizer* attached to it. This combination ensures the server can interpret incoming SSL packets. Now a *SSLContextFactory* object is made. This is required to make a *ServerConnector*. Some attributes are passed with the *SSLContextFactory*:

- *KeyStorePath*: The path to the keystore location. This keystore contains the private key and the certificate of the server.
- *KeyStorePassword*: The password required to gain access to the keystore.
- *KeyManagerPassword*: The password to use the keys in the keystore.

Now the *ServerConnector* is initialised and a port number is attached. The server is then reachable over two port numbers: 9999 for a normal http connection, 9998 for SSL. The http connection is just for testing purposes and will be removed at the end.

Mobile: The certificates used in this implementation are x509 certificates signed by our own Certificate Authority (CA). To ensure browsers can connect to our server we create a *NullHostNameVerifier* class which is coupled to the *HttpsURLConnection* class. This will allow all certificates to be trusted. In a real system the certificates would be signed by a real, trusted CA.

To initialise the connection, a URL object of the form https://IP-address:port/ is created. The *openConnection()* method is invoked on this URL, returning an object of class *HttpsURLConnection*. An *SSLSocketFactory* is added to this connection, containing a *KeyStore*. In this keystore the certificate of the server is located, that is how the application knows the server is trustworthy.

5.2.2. User authentication. For user authentication, the server has a database of passwords. These are hashed using the SHA-256 algorithm together with the username and the permissions of the user. After the secure connection has been established, the user is allowed to log in using his username and password. These are sent to the server over the secure channel and the server checks with its database whether the user should receive certain permissions.

5.2.3. Device authentication. To incorporate device authentication, the SSL connection above needs to be extended. At the server side, the *SSLContextFactory* gets some new attributes to allow device authentication:

- *TrustStorePath*: The path to the truststore location. This truststore contains the certificates of all trusted mobile devices.
- *TrustStorePassword*: The password required to gain access to the truststore.
- *NeedClientAuth*: When this parameter is set to true, the server will expect a certificate from the client when setting up an SSL connection. If the client cannot deliver a certificate, communications will end.

Similarly, a *KeyStore* is created on the mobile device, containing the certificate of the device and the private key. This keystore is used to create a *KeyManagerFactory* object. The *KeyManager* is added to the *SSLContext* together with the *TrustManager*. This way, the application knows when an SSL connection is set up which certificates

he should trust and which ones he should send to the server.

6. Evaluation

In this Section we evaluate the security of the four suggested approaches and compare them to each other. We can identify three areas of concern when it comes to security of the architecture: server side security, client side security, and the security of the communication channel.

All architectures set up a secure connection using SSL, so the security of the communication channel is similar across the three. However, architectures 2 and 4 implement device authentication, adding an extra authentication factor and making the mutual authentication a little stronger.

Server side security concerns the storage of the password database, containing login details, permissions of users, and sometimes a trust store of device certificates. All four architectures save the contents of their databases hashed using SHA-256. In the first two architectures, the databases is found on the embedded device. If an attacker has physical access to the generator, he could steal the device and attempt to break the database. In the third and fourth architectures, the details are safely stored on a separate server.

Client side security is mostly about passwords. When the generator is moved, all passwords should be replaced. This is possible in all architectures. Users should not tell anyone their passwords, of course the architectures cannot prevent this from happening. However, adding device authentication means architectures 2 and 4 are slightly better protected against stolen passwords as the attacker also has to steal the device in order to control the generator. The architecture should also support having multiple user roles with different permissions, all four architectures support this.

The second and third architecture both have certain benefits when it comes to security, it is clear that both of them are better than architecture 1, but compared to each other they are largely equal. As discussed in Section 4.5., architecture three requires an extra server to be set up, but makes database management easier, this could influence the administrator's decision on which architecture to opt for. Architecture four combines the benefits of architectures 2 and 3 without introducing additional weaknesses, hence it is the favoured solution.

We can also evaluate the architectures against the possible attacks in Section 2.3. Since all architectures use a secure SSL connection, the attacks on the communication channel are defended against. As we have already pointed out, architectures 2 and 4 add extra defence against social engineering attacks by using device authentication, and architectures 3 and 4 can make it harder to attack the password database as it is harder to obtain.

An important security aspect that this evaluation ignores is the security of the device itself. Is it possible that a compromised device, used over a secure connection, can still compromise the entire system? To what extent? How could an attacker abuse this? These are questions that will be answered in future work.

7. Conclusion

The aim of this article was to investigate how to control industrial control systems remotely in a secure way. Some security demands and possible attacks were listed, after which four possible architectures were proposed. One of these has been implemented in practice as a case study. The four architectures were compared to each other and their strengths and weaknesses with regards to security were analysed.

7.1. Future work

The evaluation showed that the fourth architecture is the most secure, future work will focus on implementing and thoroughly testing this architecture.

An additional goal is to switch from a Wi-Fi implementation to a 3G or 4G one to improve accessibility.

A recurring issue was the fact that the user had to know the IP address of the generator when he wanted to control it. This could be resolved by adding a location server to the architecture that keeps track of the current IP address of the generator. Then when the user wants to contact the generator, the application would automatically reach out to this location server to request the current IP.

Another solution to this problem would be to use static IP SIM cards. This solution requires a 3G router to be connected to the iGenerator. When implemented, it allows the IP address to be hard coded inside the application and removes the need for an additional server.

Finally, more research should be done to determine how the security of the mobile device can affect the security of the set-up as a whole.

8. Acknowledgements

Research funded by a PhD grant of the Agency for Innovation by Science and Technology in Flanders (IWT).

9. References

[1] D.-J. Kang, J.-J Lee, S.-J. Kim, and J.-H. Park, "Analysis on Cyber Threats to SCADA Systems", *Transmission & Distribution Conference & Exposition: Asia and Pacific*, IEEE, 2009, pp. 1-10.

[2] B. Zhu, A. Joseph, and S. Sastry, "A Taxonomy of Cyber Attacks on SCADA Systems", *Internet of Things (iThings/CPSCom), 2011 International Conference on and 4th International Conference on Cyber, Physical and Social Computing*, IEEE, 2011, pp. 380-388.

[3] M. B. Line, A. Zand, G. Stringhini, and R. Kemmerer, "Targeted Attacks against Industrial Control Systems: Is the Power Industry Prepared?", *Proceedings of the 2nd Workshop on Smart Energy Grid Security*, ACM, 2014, pp. 13-22.

[4] M. Abrams and J. Weiss, "Malicious Control System Cyber Security Attack Case Study – Maroochy Water Services, Australia", 2008.

[5] A. Matrosov, E. Rodionov, D. Harley, and J. Malcho, "Stuxnet under the Microscope", 2011.

[6] N. Falliere, L. Murchu, and E. Chien, "W32.Stuxnet Dossier", 2011.

[7] R. Langner, "To Kill a Centrifuge: A Technical Analysis of what Stuxnet's Creators Tried to Achieve", 2013.

[8] S. Tom, D. Christiansen, and D. Berrett, "Recommended Practice for Patch Management of Control Systems", 2008.

[9] "VoCore – A Coin-sized Linux Computer with Wifi", http://vocore.io/, 2016.

[10] "Getting Started with pfSense Software", https://www.pfsense.org/getting-started/, 2016.

[11] Siemens, "Industrial Remote Communication Sinema Remote Connect", 2015.

[12] Z. Xiaoxiang, "Modbus Protocol and Programing", *Electronic Engineer*, vol. 7, p. 016, 2005.

[13] R. Morgan, S. Cantor, S. Carmody, W. Hoehn, and K. Klingenstein, "Federated Security: The Shibboleth Approach", *Educause Quarterly*, vol. 27, no. 4, pp. 12-17, 2004.

