IT-security of industrial systems

Seminar: IT-security

Kilian Lütkemeyer

inf7269@fh-wedel.de

FH Wedel – University of Applied Science

WS 2012/2013

Advisor: Prof. Dr. Gerd Beuster
gb@fh-wedel.de

Contents

This article was created during a seminar on IT security in the FH Wedel, considering different aspects of IT security. This particular article describes the security in industrial IT systems, possible vulnerabilities and the attack vectors for such systems.

Starting with a brief overview of the development and the change of importance of security within industrial IT-systems, this article will provide a critical viewing of various statistics for acquisition of incidents in industrial IT-systems.

A discussion of the most common structures of industrial IT-systems and their most important components with their specific requirements allows a more detailed view of possible attack vectors on those systems, leading to a final example of an attack on an industrial IT system: A analysis of the STUXNET worm.

Part I.

IT-security of industrial systems

1. Introduction

With growing affluence the wages raised and in most branches of industry many steps were mechinized. Starting with single steps to be executed by simple machines controlled and observed by humans, the industrial processes became more and more automated by using sensors, relays, cam timers, drum sequencers and some dedicated closed-loop controllers.

By combining some tasks and creating possibilities for connection and communication of the controlling units for these small steps, the first industrial control networks were built. These systems were very specific and relied on proprietary hardware and without any common standard. The logic relies on the right wire-connection of the relays and the other components. Changing the logic requires a physical change of the wire connection, which requires a lot of time to maintenance.

To keep the system maintainable and flexible enough to perform small adjustments without requiring electricians to rewire all components, the logic was more and more implemented in software, running on digital computers, which were connected to the sensors and actuators. The first controlling computers were general purpose computer systems requiring protecting the computers' hardware from the plant floor conditions. But it was not practical to provide a room with conditions required by the controlling hardware and connection all sensors and actuators to the hardware within it. Therefore a new type of computers was created: The industrial purpose computer.

Industrial purpose computers were very robust against a greater range of temperature, variations in power supply and vibrations or other mechanical movements. For an easy connection of all the other industrial components they provide an easy bit-based in- and output. The first controllers replaced the relay logic, so they were designed to be programmed in a ladder-logic, which resembles a schematic diagram of the old replaced relay logic. Later they changed to be programmed with instruction lists executed on a stacked based machine.

Despite the different ways of programming, controllers were replacing the classic industrial IT structure of hardware implementing processing logic more and more, so the industrial logic was transferred into a software representation.

Nowadays all developed countries rely on a highly optimized industry. The IT-system, managing and controlling these complex industrial processes, have to be protected from attacks and failure, because a damage of the IT-system would stop the whole process and possibly even create a danger to workers within the industrial facilities. An attack on critical facilities of the infrastructure (power plants, gas stations etc.) could stop economy in some parts and cause a loss of billions of dollars.

The last years, industrial IT-systems became bigger, more complex and they control more parts of the industrial processes. The security of those systems was not considered in detail for long times. They seem to be secure just by having a proprietary system, so it would require a lot of internal knowledge to attack such a system. But being a very crucial target, professional attackers may invest in such a knowledge just to damage a single facility by hacking into their IT-system.

When the slammer worm infected a nuclear power plant in 2003, even the last expert of IT-security was convinced, that industrial facilities became networked with common business application networks, that attacks on them became more and more likely, even without physical access to the system and without detailed knowledge on every system component.

In conclusion the security of industrial IT-systems is more important than ever before and after years of ignoring the danger of attacks, the current security measure concepts developed for other types of IT-systems (like application-server-systems, personal computers, etc.) have to be inspected for making the industrial IT-systems more secure and safe too.

2. Incidents in industrial IT-systems

To find the weak points in a any security system, it is very important to analyze all attacks and other security relevant incidents of industrial IT systems. It is also important in order to prevent some incidents to evaluate reports and analyse incidents in other industrial facilities using similar IT-systems and -structures.

To provide the required information for such an analysis, there were some attempts to create a international database for statistical analysis of incidents with industrial IT-systems and industrial systems in general.

2.1. The Industrial Security Incident Database

In 2001 Eric Byres, Justin Lowe and David Leversage from British Columbia Institute of Technologies (BCIT) developed the Industrial Security Incident Database (ISID) for tracking incidents in industrial control systems. In their academic research project they tried to analyze trends and patterns of such incidents. After about five years of collecting and analyzing incidents the BCIT and the database developers discontinued the project.

2.1.1. Database structure

The database defines a simple structure for tracking and provides a simple tool for incident registration. In addition to a descriptive name, the date, the company branch, a brief description of the incident and the impact on the company, more specific information of the incident is reported by categorize the type of incident (e.g. accident, virus, etc.), the entry point into the system (e.g. internet, network, control panel etc.), the type of hardware and software which was impacted and as far as known also information on the perpetrators and their motivation.

Additional information on measures to prevent the reoccurring of the incident can also be registered within the database, which might be one of the most important but sensitive information within a record. On the one hand information on countermeasures may help to enhance the security level of other industrial facilities and their IT systems, but they may also provide additional information for attackers to get access to the described system.

To ensure the quality of the records within this database all incidents contains references to confirm the information and can be rated according to their reliability on a scale from one (confirmed) to four (hoax or urban legend).

2.1.2. Type and extend of data generation

The data was collected by the researchers from public known incidents and by private reporting from member companies. In exchange for reporting their incidents, the member companies got read access to the database to use the recorded information on other company incidents to close possible vulnerabilities before causing a new incident.

By 2004 41 incidents were logged in the ISID with eleven of them still pending on detailed investigation. Seven incidents were flagged as urban legend and removed from study data. In his report “The Myths and Facts behind Cyber Security Risks for Industrial Control Systems” [Byres and Lowe, 2004] Byres analyzes the 34 remaining plausible incidents:

Regarding the amount of incidents per year (figure 2.1) a massive increase of reported incidents since 2001 is detectable. Byres concludes the raise must be based partially on the general raise of maleware in the internet. But most of the raise is based on the

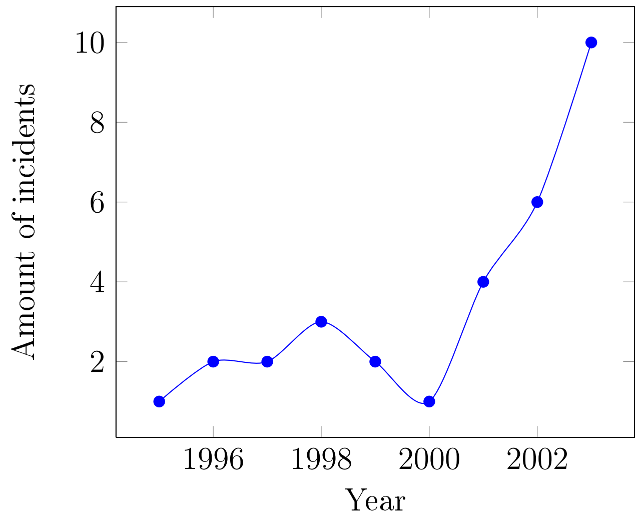

Figure 2.1.: Amount of incidents per year (1995-2003) according to the ISID

start ISID itself and the growing group of member companies reporting their incidents to the database. Older entries in the database are all public known and investigated incidents. Smaller incidents often caused no public stir, so getting knowledge that there was an incident or even detailed information on this incident is not very likely.

One of the problems of the ISID was the small set of incidents reported to them, regarding the estimated number of unknown cases which is about ten times higher according to Byres ([Byres and Lowe, 2004]) . It is possible, that companies fear a damage of their reputation reporting incidents to a database, where every other member company can see the incidents.

2.2. Repository of Industrial Security Incidents

Two years after closing the ISID research project Eric Byres started a collaboration with Mark Fabro on the Repository of Industrial Security Incidents (RISI) with a goal of making RISI available to the entire industrial automation community.

In this repository detailed information on security incidents in industrial systems are collected and analyzed. Even supporting or member companies do not have direct access to the information stored in there. Based on statistical analysis the information will be processed for multiple reports handling types of systems or specific incidentlat types.

In July 2009 finally the Security Incidents Organization^{TM} was founded to operate the RISI. This non-profit corporation was established to be a self-sustaining organisation focused on performing research in the public interest and making the results of that research available to the public on a nondiscriminatory basis.

2.2.1. Interaction of RISI and companies

Compared with the ISID, RISI also depends on companies and their submit of incident records, but the raw incident information is not public to the group of participating companies. Only the group of senior technical advisors has access to this information for creating their analysis and reports.

Another possibility for companies to participate in the project is becoming a advisory panel member, which has to provide guidance of the Security Incidents Organization^{TM} regarding their policies, products and services. They do a kind of community work recruiting new member companies and updating press releases. Nevertheless advisory panel member do not have any access on incident records nor on reports, which have not been published.

In conclusion the RISI project is a development of the ISID provided by a self-sustaining organisation. It is more likely companies will send them information on incidents due to the closed incident database. The reports, which can be bought there, provide many information on current incident situation in industrial systems, which provides important know-how for the security of control systems.

3. Logic controllers

A logic controller is one of the most important components in every industrial IT-system. It is optimized for the general requirements of controlling industrial facilities based on sensor information.

The task of a controller is to control some actuators based on information given by sensors and the application logic implemented in the software on the controller. Therefore an infinitive loop is executed on it, in which the application is reading sensor information, calculating the actions to perform based upon the input and setting the required output values for the actuators.

3.1. Requirements for controlling industrial processes

The most important requirement for controlling any industrial process is the safeness of the controlling process. The controller has to react in a deterministic way on every input. This includes also bringing the system into a safe state on invalid input, which requires a stable controller system, which will not freeze or crash. With a valid input, the controller must run in a standalone manner without any failure for long times (month/years). Many desktop operation systems are not capable of running such a long time without becoming slow or unstable by littering the memory over time.

Another crucial requirement is the ability to provide hard realtime execution of the controlling logic. This requirement assures, the controller will determinate the right reaction on some given input information from the sensors, in an exactly defined time frame. To fullfill this requirement a special firmware and operation system is required on the controller. Most desktop operation systems cannot provide the realtime ability for any application based on the system of their scheduler, and their target to make less limitations on usage than realtime systems. A realtime system will automatically (based on the kind of scheduling) require some limits for every task running on it and for their total amount.

These two requirements fit apply to every industrial controller. They have to be detailed exactly for every new project, where a controller should be used. The time that is allowed to be required for executing the application must be determined and defined when planning the structure of the system based on the other components.

Another obvious requirement is the amount of input and output connections. They depend on the task of the controller, too, and on the sensors and actuators types, as for example a single touch sensor may require less or at least a different connection type than a sensor for pressure or temperature.

If the communication of multiple controllers is required, additional connections are required, as for example an ethernet or firewire connector. Handling the communication of different controllers is very difficult in some cases, caused by the realtime requirement. To ensure the controller does not have to wait for answers, the protocols should be handled in a seperate task running on the controller and not interacting with the realtime application task on it.

Therefore the complexity of a industrial IT-system increases dramatically on connecting multiple controllers and letting them communicate. But nowadays this complexity is required in multiple parts in the industry for handling the huge amount of actuators in the sophisticated manufacturing process, regulated by the IT-system.

3.2. General and single purpose controllers

In general there are two kinds of controllers, which are used for different tasks: The general purpose controller, which is based on hardware modules and where the application logic is fully build in software, which runs on an operations system designed for the hardware module and the single purpose controller, which is a controller, that was totally developed for one single purpose, including hardware- and software-development.

General purpose controllers for industrial systems are mostly designed in modules: One base controller module on which the operation system and the application logic is implemented and executed on. Additional extension packs for all kind of input and output connections can be added and connected to this base module to provide the required functionality.

Even if the controller could be build up by combining different modules, there will always be components within a single module, which are not required for the controller's task. These components make a general purpose controller more expensive in production than an optimized single purpose controller, which only contains all required components. On the other hand, the untapped components allow to extend the controller's functionality without making any modification to the hardware.

Nowadays most general purpose controllers are programmable logic controllers (*see chapter 4. PLC: Programmable logic controller*). They can be bought in many variations from industrial IT suppliers like for example SIEMENS.

Single purpose controllers are more optimized on performing their task than general purpose controllers. The general concept of implementing the application logic in software is still persued here, but some operations may be implemented in hardware directly to ensure a more performant execution and the firmware running the application logic is more optimized for the task providing only the required functions and having less system overhead in the operation system.

Typically, compared to a general purpose controller providing the same functionality, single purpose controllers are less expensive in production because they were build only with the hardware components actually needed to fullfill their application. This benefit is paid for by significant higher costs for the development of such controllers.

Another downside of single purpose controllers is the difficulty of changing the logic of the controller. Even small changes may not be possible or only be possible with huge concessions on execution speed due to the limitations of hardware.

Due to the cost structure of the two general kinds of controllers, the single purpose controller is used where a huge amount of controllers is produced, all computing exactly the same task and where no changes of application logic are expected. The costs for development are outweighted by the savings in production costs at high numbers. Examples might be highly optimized controllers of the automative industry which are built into every new car, where the controllers are expected not to change their general behaviour for the automobiles lifecycle.

Another reason to choose a single purpose controller instead of a general purpose one, even if the amount of required controllers is small, is the requirement of very high speed realtime. Based on the small system footprint and overhead and the possibility to transfer some of the calculations into hardware, single purpose controllers can guarantee the requirement of much less time per loop execution.

Most industrial IT-systems are expected to change in time and mostly realtime conditions providing guaranteed times within some $10ms$ are sufficient. Therefore the most common type of controller is a general purpose controller. In nearly every branch of industry racks full of them can be found controlling the actuators within the industrial facilities.

4. PLC: Programmable logic controller

A programmable logic controller is general purpose controller. It is designed extremely modular providing extension modules for more input and output arrangements and for communication with other modules.

The instructions to execute are typically stored on a battery-backed-up or non-volatile memory. Therefore compared to hardware implemented control systems changing some parts of the controlling flow has become much easier.

4.1. The operation system

As a consequence of the different requirements for the control software on PLCs compared to regular application computers or servers, there have to be specialised operating systems for running such software. One of the most significant difference is the requirement of realtime execution of the custom program instructions, which is not supported by most common personal computer or server operation system.

In contrast to common option, that realtime execution means a very fast code execution, a realtime condition only defines a fixed time range in which the code must be executed, which could be a really long time. But in the industrial context this is typically the time after which an actuator must react on a specific event measured by sensors.

Figure 4.1 shows a simplified flow chart of the execution of a common PLC operation system. After some startup routines the PLC continues to run an infintive loop: The operation loop (*see section 4.2. The execution loop*).

To ensure the realtime capability of the operation, the custom controller program itself must also fit the realtime requirements, which is important to mind when developing these programs. Typically the IDEs for developing PLC programs provide algorithms to check the requirements the program must fulfill.

The first PLC initialation is only executed once each time the PLC starts the execution. This routine loads the required libraries for code execution and run some checks for PLC hardware and software integrity, including a memory checkup. Which libraries are loaded into the memory depends on the custom PLC program code and the settings defined on transfer of the program on the PLC.

This first initialisation may take several seconds which is no problems for the realtime ability of the PLC, because it only runs once on startup and is not included in

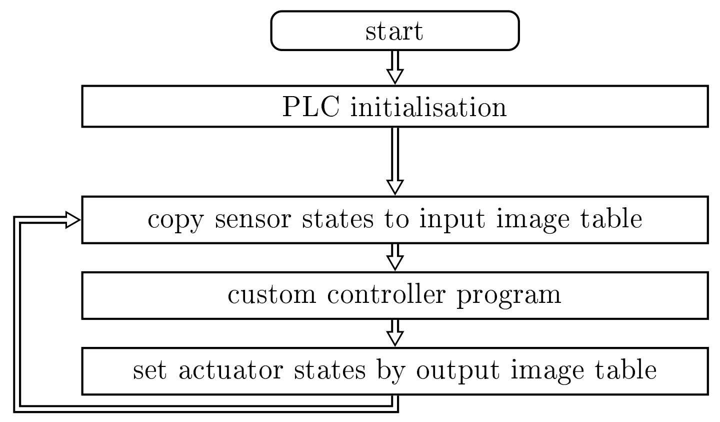
start
PLC initialisation
copy sensor states to input image table
custom controller program
set actuator states by output image table

Figure 4.1.: Simplified execution flow of operations systems for PLCs

the operation loop. Except by hardware or PLC operation system errors or a manual PLC shutdown, the PLC execution flow will never end, which is normally necessary in industrial context.

4.2. The execution loop

Within the operation system the execution loop is one of the most important parts. The maximal time required to complete one loop cycle is proportional to the minimal time a controller can define as time required to react on a change of sensor information. So the first thing to look at for raising the frequency of IO-Interactions is the minimisation of the control loop and the removal of all not required code within.

Nevertheless there are some additional steps within the execution loop, which may not handle any task of the industrial progress. They are required for a permanently safe code execution and guarantee the high demands of reliability of the controller.

4.2.1. IO-Integrity

The simplified model shows the operation loop, which is seperated in three steps (see figure 4.1): Copying the sensor states to the input image table, executing the custom application program and setting the actuators by copying the output image table to the physical output. With this setup, the physical in- and output are seperated from the custom application program, which is required to ensure the integrity of input information.

As an example, we assume we have a special sensor, which provides a two-bit value, and we also have a two-bit output. The algorithm (see algorithm 4.2.1) starts by checking if a very complex function f_1 with the first (in_1) and the second signal (in_2) as input returns true. If this happens the first output (out_1) will be set to 1 and if another complex function f_2 with the second parameter as argument returns true, too, the second output (out_2) will also be set to 1.

Obviously the program should only set out_2 to 1, if the both functions f_1 and f_2 return *true* with the given parameter.

```
if $f_1$(getInput($in_1$), getInput($in_2$)) then
    setOutput($out_1$, 1) if $f_2$(getInput($in_2$)) then
        setOutput($out_2$, 1)
    else

    end
else

end
```

Algorithm 4.2.1: Example algorithm of custom application program in the PLC

With direct access of the program to the physical in- and output this example code might create unexpected results: After getting the input values for the first function call, the input may change, so the second function might be based on different sensor information the the first call.

A similar situation exists for the output: The two bits of the output targets the same actuator and defines an encoded command. Setting one bit after the other, which will typically happen here, as the functions may take some time due to their complexity, will propably show a wrong command at the output for some short time, causing a wrong behaviour of the whole system.

So to avoid this problem an input and output buffer is created, which is synchronized with the physical in- and output before or after the custom application is executed by the operating system. The program code can now only access the input and output image tables and the integrity of the communication states can be garantueed during program execution.

4.2.2. Runtime checkup

Especially in industrial systems, where the controllers are permanently running with a typically high utilization, it is likely some parts of the hardware will break down sometimes. Those malfunctions can create further damage if they are not recognized immediately, as it may happen on memory errors for example.

The startup integrity check would usually detect such malfunctions and report them instead of entering the execution loop to prevent further damage, but once entered the execution loop, the startup integrity check will not be executed anymore until reboot. To enable the PLC to detect errors during the execution loop, an additional checkup can be added at the first position of the execution loop, as shown in figure 4.2.

To keep the time for a single execution of the loop as small as possible, not the whole checkup will be done at once. Each time the new additional checkup will be called, a

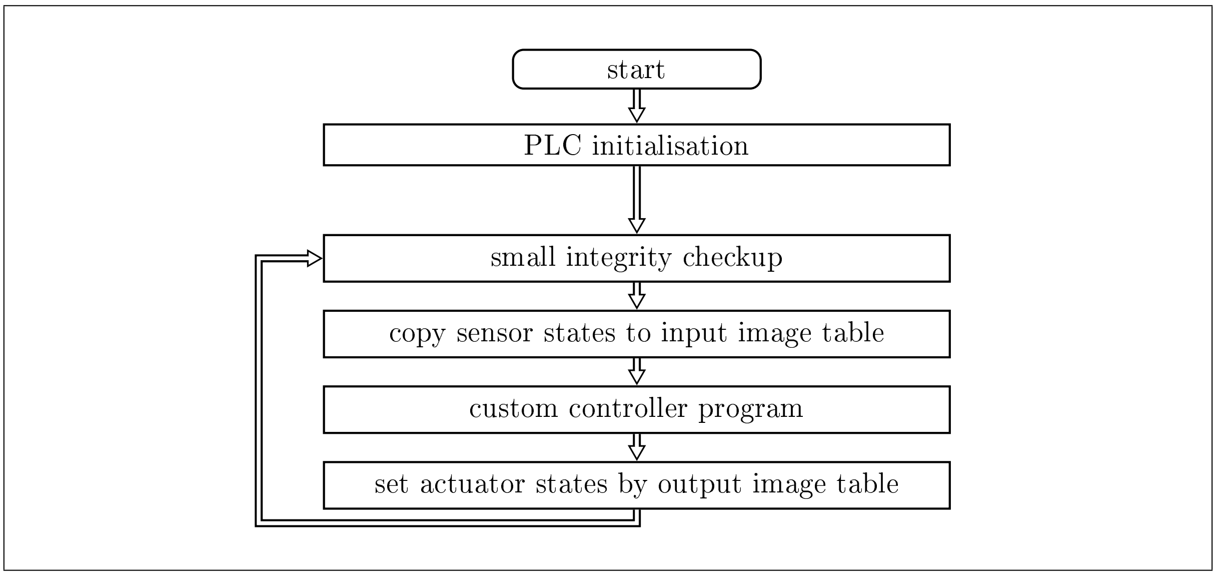
start
PLC initialisation
small integrity checkup
copy sensor states to input image table
custom controller program
set actuator states by output image table

Figure 4.2.: Execution flow with additional integrity checkup

little part of a defined checkup process is fullfilled. Therefore the whole checkup will be done after some execution cycles.

4.3. External communication

Depending on the complexity and structure of the whole IT system in the industrial facility, and the task for the controllers, it might be necessary for the controllers to communicate with each others. This requirement causes some problems considering the realtime capability of PLCs.

Some communication protocols and methods do not match realtime conditions or require a specific frequence of execution to guarrantee their own realtime execution. Typically this frequence will not be the same frequence the PLC needs for its main task.
One option to solve this problem of different requirements is to find the greatest common divisor of the required execution time cycle and to run the communication protocol instructions and the custom application code in the same execution loop. This solution fulfills all requirements but will in most cases end in extrem high processing requirements for the PLC, especially with more then two protocols and different required execution periods.

Regarding multiple controllers it is getting even more complicated: Assuming two controllers are running with the same duration per period, it is unlikely the both share exactly the same tact, so even between those two controllers communication requires synchronizing mechanisms to provide a safe exchange of information.
To make things more complex, it is very likely for different controllers to work at different frequences. A controller for an industrial laser for example controls a very quick reacting system and must therefore work at a high frequence to react very fast

on changes of conditions, but a controller for the cooling system, which has to cool the laser among other things and reacts lazy compared to the laser system, can work with a significant lower frequency.

To solve this problem PLCs became capable of handling multiple tasks in parallel. The communication is seperated from the application flow and outsourced in specific libraries, running their communication protocols in different tasks. The libraries provide buffers, which the application can access to use the library for communication.

Each library and the custom application can now be considered as seperate unit, defined by its own execution loop with its own execution period as shown in figure 4.3

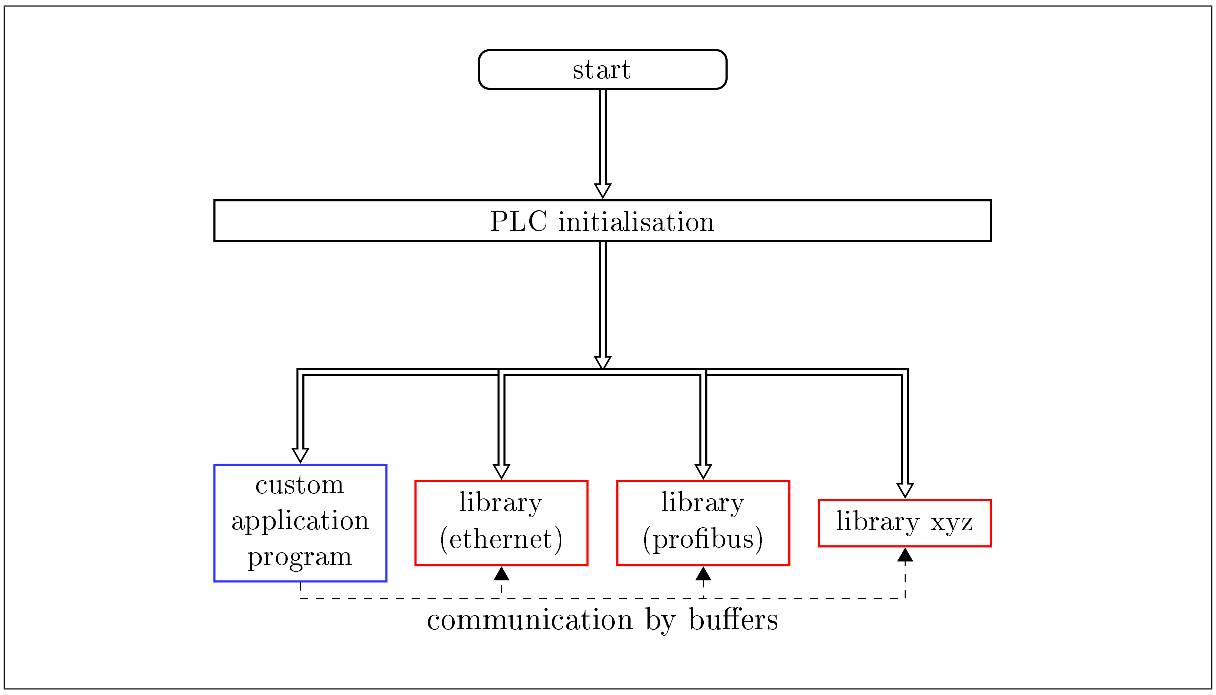

Figure 4.3.: Execution flow of multiple tasks on a single PLC

After splitting up the program flow to serveral tasks running seperated and communicating with synchronized and safe buffers and methods, the problem of providing a realtime condition can now be handled with regular realtime schedule algorithms, executing several tasks on one or more processors fulfilling all the required time periods by intellegent scheduling.

5. The structure of industrial IT-systems

A single controller is good for managing a single and simple task: Controlling a small amount of actuators based on information from sensors. All actuators of the PLC should act as a single unit building a single task.

To build up a whole industrial process by creating one task would be beyond the possibilities of a controller. In most cases, extremly different requirements for the small tasks within an industrial process and their critical period times, a single controller would require too much resources to fulfill the task permanent and safe.

5.1. PLC-systems

A PLC-system is a simple and small network of multiple PLCs. In theory they can communicate with any protocol, but due to standardization and maintainability they use often protocols like Profibus or Ethernet, but for some tasks it is favorable to use a specific proprietary protocol optimized for the specific task having a smaller protocol overhead in communication.

Within a PLC-system each PLC can act as sensor or as actuator for another PLC. A typical example for PLC systems is controlling an array of self-contained working systems, all performing the same action. Every system itself is controlled by a PLC which communicates with a master PLC regulating the supply of all the subsystems.

The master PLC has not to calculate the need of each subsystem, because the subsystem's PLC indicates to the master PLC there is a need of supply. This ensures a integrity of information between the master PLC and each subsystem.

5.1.1. Human-machine-interfaces in PLC-systems

In addition common PLC-systems have human-machine-interfaces. These interfaces are used as display for current sensor information or a graphical visualisation of the system's state.

In general, these interfaces in PLC-systems provide no option for workers to directly interact or manipulate the progress, but in some cases a PLC working as interface for a human to control the industrial process is connected with human input devices, acting as sensors for the PLC. In this case the PLC can be used for small adjustments of the industrial process. Nevertheless acts a PLC-system autonomous without the requirement of human interaction.

5.2. SCADA-systems

Supervision control and data aquisation (in short SCADA) systems represent complex and easily adjustable control systems, managing a full industrial process. Within the SCADA-system, the SCADA-server is the main component.

The whole industrial process is mapped on the server, providing several options for manually defining limits or recommended values. The actuators in SCADA-systems are still controlled directly by PLCs, but each PLC or PLC networt is connected to the SCADA networt to retrieve information and commands from the SCADA server.

5.2.1. Logging and data aquisation

One of the advantages of SCADA systems compared to complex PLC systems is the possibility of logging system events or collecting information on the systems state for a given period to enable detailed analyses of the process and search for optimization.

This aquisation of data of the system is also very important for dynamic regulation of recommended values, based on complex statical calculations which cannot be accomplished on PLCs due to their duration of execution.

Usually SCADA-servers are connected with application servers of the business application network to provide their information on resources and production to common business software, like ERP software. This can provide more possibilities for automatisation especially in resource planing.

6. Summary of the present situation

The amount of attacks and incidents in industrial IT-systems has grown the last years. The infection of the Davis-Besse nuclear power plant [Byres and Lowe, 2004] was a wakening call for industrial IT security and at least since then security measures are a sensitive topic in industrial facilities. With the announcement of the STUXNET-worm, which will be analysed more detailed in the next part of this article, the importance of industrial IT security was brought to public.

6.1. New era of maleware

Most experts on IT security agreed on one fact: "Stuxnet is not only a new virus or worm but it's a new era of malware." [Thabet, 2011a, p. 2] With STUXNET, worms reach a new level of harm done to their victims. Providing the first rootkit for industrial PLCs, industrial controllers got in crosshairs of attackers. Controlling the PLCs actually means controlling the actuators, which is one of the biggest dangers for every industrial facility.

Disguising maleware is not new for IT systems in any way, but keeping a powerful weapon like STUXNET unrevealed for years, shows the importance of methods and procedures to verify the security and integrity of IT-systems of any kind, including those in industrial facilities.

6.2. Worms and viruses as weapons

STUXNET shows the power of worms and that they can be used as weapons, but using software as weapons comes with a high risk of getting attacked as well. The power of cyber weapons requires a new code of honour and a careful and advised deployment:

On the one side it is very difficult to ensure only a target is affected by the attack. Usually the targets are protected and not directly connected to the internet, forcing the attacker to use a method of spreading. The method of spreading must work in a way aggressive enough to breach all security measures protecting the target, but not affect any non-related systems, which may have similar structures as the target.

The spreading progress should also work either in background avoiding the attack to be observed or work with such a rate, the attack can spread enough before counter measures can have an impact. This makes the spreading itself very complicated and a careful development and planing as important as the development of the malicious code.

On the other hand there is always the risk of using the attacker's code which may be disassembled, reproduced and manipulated to start a new attack with similar effects against the original attacker. The software can be reproduced based on the codes and be manipulated without big requirements. Common weapon differ in this point: Even if the structure and technique of a weapon is known, wthe reproduction of this weapon requires specific materials or facilities.

With STUXNET providing a PLC rootkit, the development of new worms targeting specific industrial facilities based on SIEMENS STEP-7 controllers became relative easy. So before using a cyberweapon the attacker must be sure, his own security measures are effective against the used worm or virus, to minimize the risk of getting beaten at their own game.

Part II.

STUXNET – a cyber weapon against the Iranian nuclear program

7. The evolution of STUXNET

STUXNET is a very complex malware and maybe one of the first cyber weapons against industrial IT-systems. It was designed to sabotage the Iranian nuclear weapon program. The STUXNET-worm was discovered in June 2010 first, but afterwards more variants of the worm were found (see table 7.1).

It was targeted at the development system of the Iranian nuclear facilities to manipulate the STEP-7 program, which is the IDE used to develope the programs for the SIEMENS PLCs used within the uranium enrichment facilities. The manipulated program generates malicious PLC programs, which causes damages within the facilities.

STUXNET succeeded in working unrevealed and slowing down the Iranian nuclear program for years. The full damage caused by STUXNET can only be estimated, due to the highly sensitive target.

Date	Version	Description
November 3rd, 2005	0.500	Registration of C&C server
November 17th, 2007	0.500	Date submitted to a public scanning device
June 22nd, 2009	1.001	Main binary compile timestamp
July 4th, 2009	0.500	Infection stop date
March 1st, 2010	1.100	Main binary compile timestamp
April 14th, 2010	1.101	Main binary compile timestamp
June 24th, 2012	1.x	Infection stop date

Table 7.1.: Evolution of stuxnet versions [McDonald et al., 2013]

7.1. Parts of STUXNET

The latest version of STUXNET can be divided in three component packages, providing complete different tasks within the whole STUXNET project. Each of these components was required to achieve the general target of slowing down the program without revealing the STUXNET worm.

7.1.1. The dropper

The dropper was used to spread the worm within the network to infect all required system for the final attack. To prevent the worm from spreading all over the world, which might have revealed STUXNET much earlier, the worm spreaded via LAN, STEP-7-project files and removable devices, but analysed the system after infection and removes itself, if it recognized, that it infected a non-target system.

An additional mechanism for updating was integrated in order to distribute new versions even faster and more reliable. This updater even may download updates on PCs, which are attached to an intranet, but not to the internet.

7.1.2. The backdoor compiler

STUXNET uses a kind of backdoor compiler: It manipulates multiple driver files for writing STEP-7 assembly on Siemens PLCs by using a man in the middle attack. This backdoor allows STUXNET to manipulate the assembly written on the PLC without a chance for the developer to notice.

The injected code generator creates the STEP-7 assembly blocks, when writing the code on the PLCs. A very specific code manipulation will be executed ensuring only specifc PLCs in the industrial control system became infected.

In order to make it even more difficult for developers to reveal the infection, STUXNETs man-in-the-middle component removes his own manipulation from the code, when loading the infected PLC code back into the IDE, so even checking the code send to the PLC by retrieving it afterwards will not reveal the worm.

7.1.3. PLC payload

The PLC payload is the final payload causing the damage. With the use of a PLC rootkit which is part of STUXNET, some centrifuge and auxiliary valves are manipulated to damage the cascaded centrifuges. The harm to the system is proceeded in eight steps, damaging the centrifuges for a long time, without revealing a obvious reason for the destruction.

8. Attack scenario

The way the STUXNET attack really happend is not completly provable, but based on the functional range of STUXNET a possible and reasonable attack scenario can be reconstructed.

8.1. The infection of the system

The PLCs are typically programmed from Windows computers not connected to the internet and in most cases not even to the internal network. As each PLC is configured in a unique manner, the attackers needed schematics of the industrial control system. These schematics were propably retrieved by earlier versions of STUXNET.

After retrieving the schematics, it was possible to setup a mirrored environment for testing and debugging of the STUXNET code, which propably needs up to six month for a team of five to ten developers and multiple individuals for managing, testing and quality assurance.

In addition two drivers required for STUXNET were signed with two digital certificates which are stolen from the companies. Due to their physical proximity it is possible, that they were stolen by someone having phyiscal access to computers or networks with direct access on the certificates.

To actually infect the target facility, the worm has to be brought in the facility. This may have occoured by a third-party, like a contractor who may have access to the facility. A removeable drive may be used for the first infection.

Once STUXNET had infected a PC within the facility, it starts to spread to other computers on the LAN using a zero-day exploit. In addition Step-7 projects, which are development projects for PLCs were infected, because it is likely they will once be opened on the development PCs for the target PLCs. For a more aggressive spreading another exploit was used for spreading by removable devices.

STUXNET tried to communicate with a Command and Control-Server (C&C Server) for getting new commands and updates. Because it was assumed, that the key computers are not connected to the internet, the information of the C&C server was provided within the network by Peer-to-Peer connections of the infected PCs.

Once STUXNET found a suitable PC running the STEP-7 development environment, it manipulated all code transfered on PLCs to sabotage the industrial process.

8.2. STUXNET 0.5: The eight steps of system manipulation

In STUXNET 0.5, the centrifuges of the infected industrial system were damaged slowly in only eight steps of permanent manipulation of the executed code on the controller.

8.2.1. Step 1: Identification of a steady state

The code verifies the system has reached the steady state for the attack. Therefore all states of centrifuge and auxiliary valves are logged and must noch change within 300 snapshots. All activated cascades must have run for at least the last three days and one must have been operation for at least 35 days or all cascades at least 297 days in total.

This step is required to ensure, the manipulation does not start working immedietely after starting the system or during some change of the system configuration. In both cases all changes and non-expected values from sensors and valves would be very suspicious.

8.2.2. Step 2: Recording I/O snapshots for replay

Manipulating valve control values will create unusual sensor information which will be suspicious and warning to technicans observing the system. To avoid these different sensor informations later, 21 I/O snapshots were taken for a replay during the manipulation.

8.2.3. Step 3: Attack centrifuge valves

The normal operation pressure is measured and stored for later replay.

For each stage of centrifuges except the feed stage, multiple centrifuge valves will be closed. The feed stage remains completly open, while the centrifuges at product end and tails are completly closed.

If not all centrifuges of a stage are closed, the valves to close are chosen randomly.

8.2.4. Step 4: Secondy pressure reading

One random stage opens the first two centrifuges valves and the pressure will be read at stage 1. Typically this pressure should be relativly low. This step might by omitted, if the pressure measurement at the beginning of step 3 was not obtained proberly. In this case hard coded default values are used as regular operation pressure.

8.2.5. Step 5: Wait for pressure change

The program remains in this step until a specific defined change of pressure is measured or a time limit is reached.

8.2.6. Step 6: Attack auxiliary valves

In this step all auxiliary valves are opened, expect the valves 17, 18 and 20. The code waits for at least 2 minutes and 53 seconds before continuing with step 7.

8.2.7. Step 7: Wait and show replay

The code remains in this state for 6 minutes and 58 seconds showing the previously recorded sensor values, so technicans will not suspect any unexpected behaviour of the system. In addition setting new control values during this period is not possible.

8.2.8. Step 8: Reset

Reset all data and return to step 1.

8.3. STUXNET 1.x: Difference in centrifuge manipulation

It is not really proved, whether the program of STUXNET 0.5 worked as expected and damaged the centrifuges in the wished manner. Nevertheless the attackers changed tactics in the new version STUXNET 1.0:

By changing the speed of the centrifuges it was much easier to damage them without creating attention.

Part III.

appendix

List of Figures

List of Tables

Bibliography

[Benjamin et al., 1998] Benjamin, R., Gladman, B., and Randell, B. (1998). Protecting IT systems from cyber crime.

[Byres and Hoffman, 2003] Byres, E. and Hoffman, D. D. (2003). The myths and facts behind cyber security risks for industrial control systems.

[Byres and Lowe, 2004] Byres, E. and Lowe, J. (2004). The myths and facts behind cyber security risks for industrial control systems.

[Falco et al., 2002] Falco, J., Stouffer, K., Wavering, A., and Proctor, F. (2002). IT security for industrial control systems.

[Falliere and Liam O Murchu, 2011] Falliere, N. and Liam O Murchu, a. E. C. (2011). W32.stuxnet dossier. Technical report, Symantec Security Response.

[Joyal, 1996] Joyal, P. M. (1996). Industrial espionage today and information wars of tomorrow.

[McDonald et al., 2013] McDonald, G., Murchu, L. O., Doherty, S., and Chien, E. (2013). Stuxnet 0.5: The missing link. Technical report, Symantec Security Response.

[Naedele, 2005] Naedele, M. (2005). Standardizing industrial it security - a first look at the iec approach. In Editor, F. and Editor, S., editors, *Emerging Technologies and Factory Automation, 2005. ETFA 2005. 10th IEEE Conference on*, pages 7 pp. – 863.

[Summary, 2011] Summary, E. (2011). What is the siemens pcs 7 industrial control systems – a primer... 3.

[Thabet, 2011a] Thabet, A. (2011a). Stuxnet malware analysis paper. Freelancer Malware Researcher Report.

[Thabet, 2011b] Thabet, A. (2011b). Stuxnet malware analysis paper. Freelancer Malware Researcher Report.

[Wellenreuter and Zastrow, 2011] Wellenreuter, G. and Zastrow, D. (2011). *Automatisieren mit SPS - Theorie und Praxis.* Vieweg Verlag Wiesbaden, 5th edition.