Software Vulnerability Design and Approaches for Securing SCADA Control Systems

Giovanni A. Cagalaban, Jae-gu Song, Sungmo Jung, Seok soo Kim
Department of Multimedia Engineering, Hannam University
gcagalaban@yahoo.com, bhas9@paran.com, sungmoj@gmail.com, sskim@hnu.ac.kr

Abstract

Despite growing awareness of security issues especially in SCADA networks, there exist little or scarce information about SCADA vulnerabilities and attacks. Where security has been a consideration, there has been no clear methodology to assess the security impacts brought about by attacks. Worst, there have been no or very little security tools that have been released publicly.

This research aims to addresses the issues regarding security and vulnerability testing. Software program was created to simulate the vulnerability testing and carry out assessment methodologies to test existing SCADA software design and implementation. This paper also describes the application of a well know security testing approach known as the software implemented fault injection as well as building the model for security vulnerabilities identification and analysis. Impact analysis was also performed to provide a better understanding of the attacks. Sufficient security measures are needed to block all possible intrusion points into the SCADA system to significantly reduce the chances of a successful attack.

Keywords: control systems, vulnerability model, fault injection, impact analysis

1. Introduction

Many infrastructures and industries use computer-based systems, commonly known as to remotely control sensitive processes and physical functions previously controlled manually by its operators. These systems, commonly known as Supervisory Control and Data Acquisition (SCADA), allow a physical system such as water utility to collect data from sensors and control equipment located at remote sites [1].

However, SCADA networks were developed with little attention being paid to security. As a result, many SCADA networks may be susceptible to attacks and misuses. Furthermore, studies indicated that some water utilities may have spent little time and money securing their SCADA systems.

There exist still little or scarce information about SCADA vulnerabilities and attacks, despite the growing awareness of security issues in industrial networks. Regarding the case of information technology security, most owners and operators are often unwilling to release attack or incident data. Yet, these sensitive data are not public repositories of advisories and vulnerabilities in industrial devices unlike information technology products and protocols. Even though some vulnerability testing and research are being conducted in this area, very little has been released publicly and no so-called SCADA security tools have been released to the public.

To address the limitations, this research aims to create a software vulnerability testing and simulation program to perform vulnerability assessment methodologies to test the existing SCADA software design and implementation. The program provides features to sniff network packets and transmit them onto the network. Further, it will employ a well-known security technique in the testing of vulnerability of a software system referred to as fault injection. Specifically, this approach will be injecting faults based on software techniques is called software implemented fault injection (SWIFI). This approach has been based on years of researches and experiences in vulnerability analysis of software systems [2].

2. Related Study

To improve the security of the nation's critical infrastructure, researches have been conducted to identify and resolve vulnerabilities of SCADA systems to ensure reliability of its operations. Ongoing research and assessment activities have revealed an effective methodology for identifying vulnerabilities and developing assessment methods to secure SCADA systems.

Software tools used to determine known vulnerabilities in traditional IT systems have been widely available. The market for these vulnerability scanners has been significant and products such as Nessus, FoundScan and Internet Security Scanner (ISS) have been popular with IT administrators trying to locate unpatched computers on their networks.

Several test tools that have had success in locating new vulnerabilities in network devices based on grammar and fuzzy techniques are also been developed in academic researches. Considerable work has been done by the PROTOS project group [3] and by Tal, Knight and Dean [4]. Each considers the syntax-based generation of protocol data units that translates into a single test packet to be sent to the device under test. Their methods have proven effective in finding vulnerabilities [3][4] however, they only allow for the construction of simple single-packet test cases.

Despite efforts to improve and provide guidance to help ensure program activities address real control system security issues, still there are very little security tools that have been released publicly.

3. System Methodology

In this research, a simple SCADA device test bed was set up for vulnerability testing and assessment. A prototypical SCADA master and slave programs were used to simulate the serial communication between a SCADA master station and Remote terminal Units (RTUs) or slaves. This will allow assessment of vulnerabilities and security configurations of SCADA software used in industries.

The testbed consists of one MTU that communicates with several RTUs. The system used SCADA software for process monitoring and control. RTUs are running a Modbus communication driver to communicate and exchange data with the MTU. Attacks were simulated using direct access to the infrastructure. Figure 1 shows the prototypical SCADA testbed.

Many types of vulnerabilities exist, and computer security researchers have created taxonomies of them [5]. Security vulnerabilities in software systems range from local implementation errors to much higher design-level mistakes. Vulnerabilities typically fall into two categories. They may be bugs at the implementation level and flaws at the design level [6]. Design-level vulnerabilities are the hardest defect category to handle, but they're also the most prevalent and critical issues to deal with.

A SCADA system uses Modbus communication as represented by RS232, RS422 and RS485 communication standard. In this research, Modbus protocol was used since it lack inherent security that any moderately skilled hacker would be able to carry out a large variety of attacks if system access can be achieved.

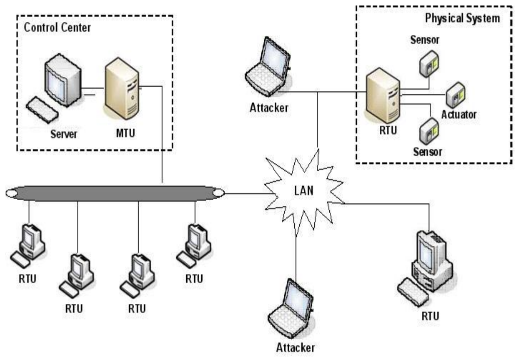


Figure 1. Prototypical SCADA Testbed

This research uses RS485 and RS232 standards to establish and examine the communication between SCADA Master and RTUs. In Modbus communication, there are two options possible: the installation of interface devices (PCI or PCMCIA type) and use of an RS485 communication converter [7] connected with an RS232 standard interface connected to a computer.

RS232 communication device was used as a medium for serial transmission of data between SCADA master and slave. A man-in-the-middle physical configuration is shown in Figure 2 wherein the SCADA master and slave is connected with an intruder that eavesdrops on the network traffic. The man-in-the-middle computer serves as an intruder to perform sniffing and fault injection through the use of a developed program. The goal of this attack was to analyze the communication link between the SCADA communication port and the RTU and develop a means to send software-injected faults to change state in the either the master or slave operation.

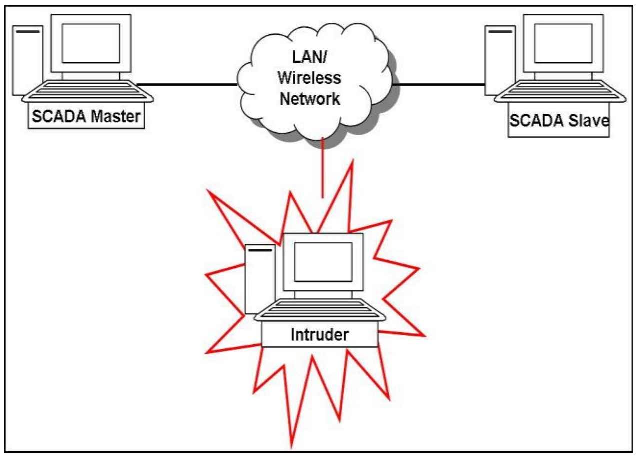


Figure 2. Sniffing attack

In this approach, sniffing activities include faults that are injected by a program into the system thereby changing its current status. Changing the status of the system environment during testing would provide assessment on how it responds and whether there will be security vulnerabilities that can be detected. If not, then the whole system is considered secure and reliable.

The use of software implemented fault injection technique has several advantages. One advantage is that it is hard to detect and analyze certain vulnerabilities and the ability to detect them depends on the personnel's knowledge of the system. Fault injection using software techniques provides a way of emulating the system vulnerability without having to be concerned with how they could occur in actual simulation.

Another advantage is that unlike other approaches where they are considerably difficult to implement and quantify results, software implemented fault injection technique provides a capability of implementing and automating the testing procedure.

3.1 Vulnerability Model

In order to determine the security vulnerabilities, it is necessary to inject faults through software that manifest themselves as security defects in systems at the application software level. One thing to note is that those faults should emulate the real system faults appropriately. The elements to be considered are internal elements and external elements. Internal elements refer to those elements that are part of the program's code and data such as variables and stack. While external elements are elements that are external to a program's code and data such as file, media, other party, and network.

One factor to consider in making and implementing a secure system is the nature of having shared elements. A program is not alone in accessing and changing these internal and external elements. Other factors, such as other users, may access and change the whole elements as well.

To provide high confidence in the validity of the security vulnerabilities caused by faults to be injected, the approach described here models the software system at a high level. Software implemented fault injection at this level emulate what a "real" attack scenarios are being done.

3.2 Building the Vulnerability Model

A faults can be injected using software techniques can affect an application in two different approaches. One scenario is when an application receives inputs from its external elements which then inherited by the medium of internal elements of the program. Figure 3 shows an indirect way in which the faults being injected are not handled by the program. Solid lines represent input from the external elements to the program while dashed lines refer to the fault via internal elements.

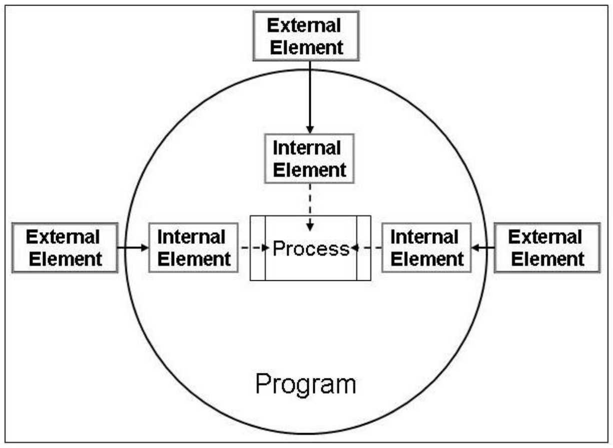


Figure 3. Indirect way of fault injection

Another scenario is shown in Figure 4. This shows a direct way in which the faults injected are not handled by the program. Security vulnerability occurs when faults are the direct cause of security vulnerability and the medium for the faults are the internal element directly.

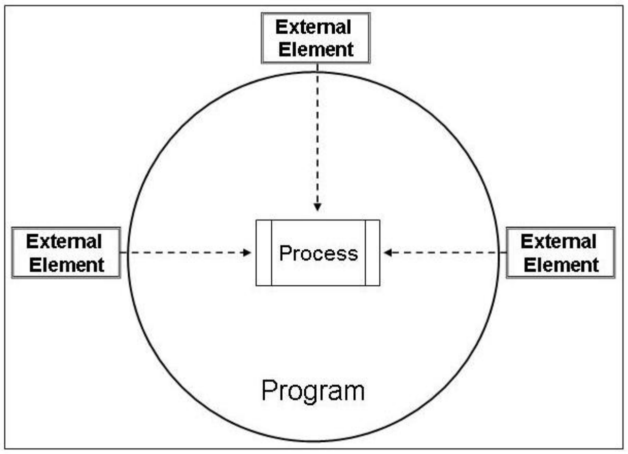


Figure 4. Direct way of fault injection

3. 3 Software implemented fault injection

To implement fault injection, the software approach created can be outlined in the following steps. The program was written in C language, a language which supports serial interface and communication. The steps are shown in order below:

1. Set **counter** and **size** to 0.
2. For each test case, do step 3 to 9.
3. For each interaction point in the execution trace, decide if the program asks for an input. If there is no input, only inject direct environment faults; if there is an input; inject both direct and indirect way of injecting faults.
4. Decide the object where faults will be injected.
5. For each fault in the list, inject it before the interaction point for the direct environment faults; inject each fault after the interaction point for the indirect environment faults since in this case, we want to change the value the internal entity receives from the input.
6. Increase **size** by 1.
7. Detect if security policy is violated. If violated, increase **counter** by 1.
8. Calculate interaction coverage. If the test adequacy criterion for interaction is satisfied then stop else repeat steps 3-9 until the adequacy criteria for interaction coverage is achieved.
9. Divide **counter** by **size** yielding to obtain the vulnerability assessment score for the application program.

To illustrate the steps, consider a portion of software implemented fault injection. A refinement of code mutation is shown below which adds code, rather than modifies existing code. This is usually done through the use of functions which are simple functions which take an existing value and change it via some logic into another value,

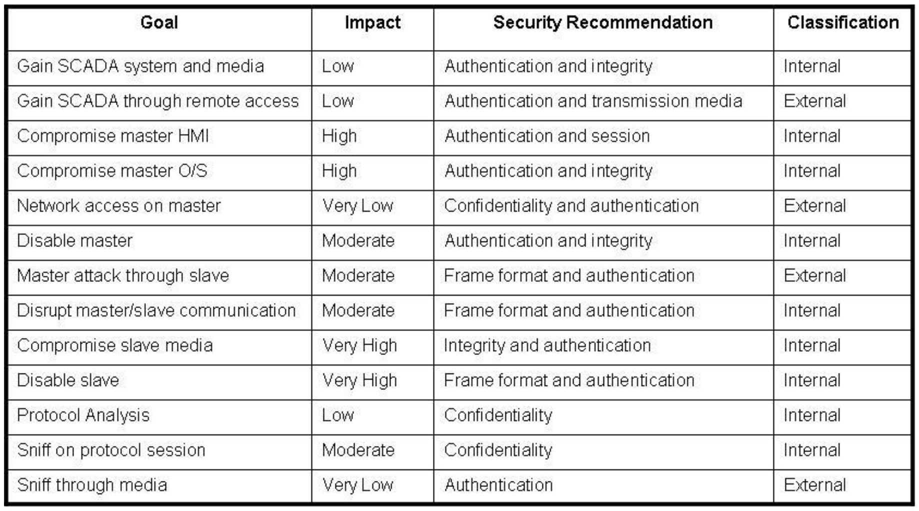
```
int funcFault(int value) {
  return value + 26;
 }
 int main(int argc, char * argv[]) {
  int a = funcFault(aFunction(atoi(argv[1])));
  if (a > 26) {
    change the value
  } else {
    call function B
  }
}
```

In this case funcFault is the function and it is applied to the return value of the function that has been called introducing a fault into the system.

4. Impact Analysis

To assess the possible security vulnerabilities, some method of assessing and rating the risk of any vulnerability is needed. The impact in this case is an expression of the likelihood that a defined threat will exploit a specific set of vulnerability of a particular attractive target to cause a given set of consequences. Sniffing activities which include software injected faults seems to only have little cost or apprehension concerns.

The purpose of the of the analysis is to determine the values associated with the goal of attack to give a better understanding which also reflect the classification of the faults to compromise the whole system. These also indicate where security recommendations are required. Figure 5 shows the results of the assessment of effects of sniffing activities and compromising the SCADA system.



Goal	Impact	Security Recommendation	Classification
Gain SCADA system and media	Low	Authentication and integrity	Internal
Gain SCADA through remote access	Low	Authentication and transmission media	External
Compromise master HMI	High	Authentication and session	Internal
Compromise master O/S	High	Authentication and integrity	Internal
Network access on master	Very Low	Confidentiality and authentication	External
Disable master	Moderate	Authentication and integrity	Internal
Master attack through slave	Moderate	Frame format and authentication	External
Disrupt master/slave communication	Moderate	Frame format and authentication	Internal
Compromise slave media	Very High	Integrity and authentication	Internal
Disable slave	Very High	Frame format and authentication	Internal
Protocol Analysis	Low	Confidentiality	Internal
Sniff on protocol session	Moderate	Confidentiality	Internal
Sniff through media	Very Low	Authentication	External

Figure 5. Assessment results

Impact analysis of the sniffing activities including fault injection indicates that the avenues of attack depend on the ability of the attacker to gain SCADA access and identify the existing protocol. If sufficient security measures are put in place to block all possible intrusion points into the SCADA system, then the chances of a successful attack are greatly reduced. Unfortunately, in this research the predominant security effort in most SCADA facilities tends focus on attacks via the Internet or through the business network. This leaves open attacks from other intrusion points such as remote field stations, the SCADA transmission infrastructure, or wireless control network connections.

5. Conclusion

Based on the impact analysis, sniffing attacks including software implemented fault injection are dependent on the ability of the attacker to gain network access and locate existing protocol devices. If sufficient security measures are in place to block every possible intrusion point, then the chance of successful attack is extremely low. Since there is virtually no security inherent in a protocol such as Modbus-based SCADA or industrial control systems, any moderately skilled hacker would be able to carry out a large variety of attacks if system access can be achieved.

The study indicated that the use of these vulnerability models in SCADA communications can significantly reduce the vulnerability of these critical systems to malicious cyber attacks, potentially avoiding the serious consequences of such attacks. The results of the study also indicate that the software implemented fault injection can be a very useful tool for modeling threats and vulnerabilities in a wide variety of systems especially SCADA control systems. However, the approach is not without its limitations. Lightweight approaches to threat modeling are useful for protocol designers, vendors, and users in an area that needs more exploration.

Future studies are needed to have more formal approaches that better aggregate subordinate values and dynamically reflect specific parameters in carrying out sniffing attacks to the SCADA system [8].

References

[1] Technical Information Bulletin 04-1, Supervisory Control and Data Acquisition (SCADA) Systems, NCS TIB 04-1, Oct. 2004

[2]. Krsul, I., Software Vulnerability Analysis. PhD thesis, Purdue University, Department of Computer Sciences, West Lafayette, Indiana, 2000.

[3] Kaksonen, R., Laasko M., and Takanen A., "Vulnerability analysis of software through syntax testing," University of Oulu, Finland, Tech. Rep. (2000).

[4]. Tal, O., Knight S., and Dean T., "Syntax-based vulnerability testing of frame-based network protocols," Privacy, Security and Trust (2004).

[5]. Byres, E.J., Hoffman, D., & Kube, N. (2006). On Shaky Ground – A Study of Security Vulnerabilities in Control Protocols. Proceedings of the 5th American Nuclear Society International Topical Meeting on Nuclear Plant Implementation, Controls, and Human Machine Interface Technology, American Nuclear Society, Albuquerque, NM, November, 2006.

[6]. James R. Davidson. Vendor System Vulnerability Testing Test Plan, January 2005 http://www.inl.gov/technicalpublications/Documents/3484441.pdf

[7] LCS-485 Converter (USB to Serial 2p RS432/485), http://kunhocom21.co.kr/product/productView.php?nProdCode=61019&service_id=pcdn

[8] Franz. M; “Flexible Threat Modeling.” www.io.com/~mdfranz/papers/unpub-may04-flexible-threatmodeling.pdf

Authors

Giovanni Cagalaban received a B.S degree in Computer Science from University of the Phillippines in the Visayas Miag-ao, Philippines, 2000 and M.S. degree in Computer Science from Western Visayas College of Science and Technology, Philippines, 2007. And currently, on the Integrate course in Multimedia Engineering from Hannam University. His research interests include Multimedia system, SCADA security, sensor network.

Jae-gu Song received a B.S. degree in Multimedia from Hannam University, Korea, 2006 and M.S. degree in Multimedia from Hannam University, Korea, 2008. And currently, on the Ph.D. course in Multimedia Engineering from Hannam University. His research interests include multimedia network system, ubiquitous system, medical information system, network security.

Sungmo Jung received the B.S. degree in Department of Multimedia from Hannam University, Daejeon, Korea in 2008. Now, he is working on the Master’s degree in Multimedia Engineering from Hannam University. His research interests include Software Engineering, Embedded database systems and Sensor network.

Seoksoo Kim received a B.S. degree in computer engineering from Kyungnam University, Korea, 1989, and M.S. degree in Information engineering from Sungkyun-kwan University, Korea, 1991 and Ph D. degree in Information engineering from Sungkyun-kwan University, Korea, 2002. In 2003 he joined the faculty of Hannam University, Korea where he is currently a professor in Department of Multimedia Engineering. His research interests include Multimedia Communication systems, Distance learning, Multimedia Authoring, Telemedicine, Multimedia Programming, Computer Networking. Information Security. He is a Member of KCA, KICS, KIMICS, KIPS, KMS, and DCS.