Research on Memory Access Vulnerability Analysis Technique in SCADA Protocol Implementation

FANG LAN, WANG CHUNLEI, HE RONGHUI
Department of Network Research
Beijing Institute of System Engineering
10th North Anxiang Road, Chaoyang District, Beijing
CHINA
flan721@yahoo.cn, hrh7706@gmail.com, wcl08@mails.tsinghua.edu.cn

Abstract—SCADA systems play key roles in monitor and control of the critical infrastructures, the vulnerabilities existed in them may destroy the controlled critical infrastructures. This paper proposes an analysis method of memory access vulnerability in SCADA protocol implementation. Firstly, the memory taintedness model of SCADA protocol implementation is formally defined. Based on this model, the detection algorithm of memory access vulnerability is proposed. Finally, the model and algorithm are validated through the experiment of vulnerability analysis.

Key words: SCADA Protocol Implementation; Memory Access Vulnerability; Dynamic Analysis

1 Introduction
SCADA (Supervisory Control and Data Acquisition) refers to the automatic industrial process control system based on computer and network. It is widely used in national critical infrastructures for supervisory and control, such as water treatment and distribution, oil and gas pipelines, electrical power transmission and distribution, transportation services, and so on.

In the past, SCADA systems usually ran on the dedicated networks with proprietary protocols and used vendor-specific hardware and software, which were isolated from the public network. To improve the efficiency and raise the degree of automation, open protocols and PCs with common operation systems are widely used in SCADA systems in recent years. More and more SCADA systems connect to the commercial networks or Internet to enable the information sharing across the enterprise, which cause increasingly cyber threats focusing on SCADA systems.

Since the implementation foundations of SCADA systems are computer and network techniques, they also have vulnerabilities in network communication, software implementation and security management etc, like normal IT systems.

In SCADA systems, the acquisition and exchange of field data and the sending of control instructions mainly depend on SCADA protocols and specifications. Since SCADA systems initially established by various manufacturers and adopted private protocols, software and hardware platforms, there is no unified standard protocol in various SCADA systems. Over the years, the domain of industrial control has moved into accepting common open standard protocols. The currently popular SCADA protocols are Ethernet/IP (Industrial Protocol), DeviceNet, ControlNet, Profibus, Modbus-TCP, and DNP3 etc. Furthermore, OPC Foundation adapts and creates OPC (OLE for Process Control) [1] specifications in order to facilitate interoperability of process control products and solve the problem of control data acquisition, and provide the unique interface for the development of industrial automation software. OPC specifications have been widely accepted in SCADA systems and become the important technical standards for industrial process control and data exchange.

Since the protocols and specifications mentioned above were developed for process control and data acquisition and focused on the real-time and reliability, they were not well considered in security mechanism. Various manufacturers also have various faults in the implementation of their products and led to the existence of vulnerabilities.

US-CERT CSSP (Control System Security Program) [2] has released 23 software vulnerabilities for SCADA system since 2006, of which 48% of the vulnerabilities associated with the memory access. Thus, memory access-related vulnerabilities (such as buffer overflows, etc.) are the most important security problems that SCADA systems faced with.
Many analysis methods and research directions have been proposed and investigated for analyzing the vulnerabilities existed in SCADA systems. Zhivich [3] has proposed the developer environment for automated buffer overflow test (DEADBOLT) that performs vulnerability analysis of SCADA software written in C and C++. Mora [4] and Franz [5] employed fuzzing technique to test implementations of OPC and Inter Control Center Protocol (ICCP) respectively. However, the fuzzing technique cannot obtain the internals of target execution such as memory and CPU register information. Therefore it is insufficient to uncover the vulnerabilities in implementations of industrial control systems. The blackPeer proposed by Byres et al. [6] is a tool that uses attributed grammars to generate and execute meaningful sequences of PDU’s, which are sent to the implementation of a defined SCADA protocol under analysis. This tool automatically generates test cases for industrial control protocols, and interprets the behaviors they exhibit during the analysis. Bellettini et al. [7][8] proposed an analysis method for finding memory corruption vulnerabilities in Modbus-TCP protocol binaries. This method obtains memory access addresses by tracing the propagation of memory access taintedness, and performs further analysis through a decision tree and a finite state machine model.

In this paper, we propose an analysis method for memory access vulnerability in SCADA protocol implementation. The rest of the paper is organized as follows: Section 2 explains the memory taintedness model of SCADA protocol implementation. Section 3 describes the detection algorithm of memory access vulnerability in SCADA protocol implementation. Section 4 validates the model and algorithm through the experiment of vulnerability analysis. Section 5 concludes the paper and points out the following works.

2 Memory Taintedness Model

At present, most of the attack methods are the control-data attacks, which hijack the control flow of the target program. For a successful invasion, the external attacker would need to provide specific values, and assigned to the desired control variable.

Based on that feature, Suh et al [9] and Crandall et al [10] proposed the usage of taintedness tracking and inspection for preventing attacks.

Inspired by the dynamic taintedness analysis method proposed in [11], in this paper, we propose a memory access vulnerability analysis method for implementation of industrial control protocols. Fig. 1 shows the process of dynamic taintedness analysis method for implementation of industrial control protocol. The upper of the diagram shows the normal course of the attack, and the data transmission and conversion during the attack, while the lower of the diagram shows the dynamics taintedness tracking and inspection mechanism. The analysis is based on the premise that, if a data can be obtained from the taintedness data through mathematical derivation or simply copy, then the data value are tainted.

![Dynamic taintedness analysis of industrial control protocol implementation](image_url)
Analyzing the vulnerabilities of the SCADA protocol implementation using dynamic taintedness analysis method, the key point is how to accurately mark data and track their transmission process. According to the characteristics of SCADA protocol and dynamic taintedness analysis, we can use graph theory to represent the dynamic memory taintedness model. In this section, we first give the definition of dynamic taintedness graph and related primitive operations. And then, section 3 describes how the dynamic taintedness model is applied to the analysis of SCADA protocol implementation vulnerabilities.

**Definition 1.** The memory dynamic taintedness graph is a binary group, \( G = (V(G), E(G)) \), where:
\[
V(G) = \{v_1, v_2, v_3, ..., v_n\}
\]
refers to the set of vertices, where each vertex represents a memory object type. Memory object type, \( MemObjectType = \{input, heap, stack\} \), where \( input \) represents the input data object, \( heap \) represents the memory heap objects (e.g., buffer), \( stack \) represents an object on the stack (e.g., variable).

\[
E(G) = \{e_1, e_2, e_3, ..., e_m\}
\]
refers to the set of directed edges, uniquely determined by the ordered pairs of vertices and the type of edges. \( EdgeType \) is the type of the directed edges, which represents the type of taintedness data spread between nodes in memory, such as data replication between buffers, data derived between variables, etc.

\( EdgeType = \{CopyEdge, ComputeEdge\} \), where

- \( CopyEdge \) indicates the copy of directed data, with the same range of bytes read and write; while
- \( ComputeEdge \) indicates other read/write edges which the bytes write range is derived from the read range and perhaps without the same size.

Using dynamic memory taintedness graph, it can be accurately determined how the conditional variable is dependent on the input. In order to facilitate the construction and processing of the dynamic memory taintedness graph, and to design appropriate vulnerability detection algorithm, we define the following primitives:

- \( LabelTaint(in) \) : Creates the source taintedness node based on the input values \( input \).
- \( PointTo(p) \) : Dynamic parses the pointer \( p \), and returns the reference node it points to.
- \( InEdges(o) \) and \( OutEdges(o) \) : Returns the set of edges that enters and out of the node \( o \).
- \( WriteRange(e) \) and \( ReadRange(e) \) : Returns the range of read/write bytes of the directed edges \( e \).
- \( SourceNode(e) \) : Returns the source node of the directed edge \( e \) (i.e., the parent node or nodes located in the tail of the arrow).

- \( IsTaintSource(o) \) : If the node \( o \) is introduced by calling \( LabelTaint() \), then returns \( true \).
- \( TransformRange(i, o) \) : Determines the offset of byte range \( i \) according to the write range of edges \( e \), and then converts the value and returns the new byte range according to the read range of edges \( e \).

Note that the operation \( TransformRange() \) should process \( CopyEdge \) and \( ComputeEdge \) differently. As for \( CopyEdge \), when converting the taintedness range to read range, the size of the taintedness range can be kept accurately; While for \( ComputeEdge \) the expansion to the entire read range is needed.

In the memory dynamic taintedness graph, the key point of tracking memory taintedness is how to determine the range of input bytes of the conditional variable, in this paper it is implemented by depth-first search of the memory dynamic taintedness graph. The search process is based on the read/write bytes range of the directed edges and the process of \( CopyEdge \) and \( ComputeEdge \) differently. The TraceMemTaint algorithm is shown in algorithm 1. The main idea of the algorithm is to check the input edges of the node that corresponds to the byte range being tracked, and then recursively process each overlapping part of the tracking range.

---

**Algorithm 1. TraceMemTaint(p, n)**

```plaintext
// Generation of the track of tracing memory taintedness.
Input : Pointer of the memory node \( p \), byte range \( n \)
Output : Track of the memory taintedness \( T \)
1. foreach \( e \in InEdges(\text{PointTo}(p)) \) do
2. \( i \leftarrow WriteRange(e) \cap \langle p, n \rangle ; \)
3. if \( i \neq 0 \) then
4. \( p, m \leftarrow TransformRange(i, e); \)
5. if \( IsTaintSource(\text{SourceNode}(e)) \) then
6. \( T \leftarrow \{p, m\}; \)
7. Output \( T \);
8. else
9. TraceMemTaint(p, m);
10. endif
11. endif
12. endfor
```

Alg.1 TraceMemTaint algorithm
3 Detection of Vulnerabilities
To construct the Dynamic memory taintedness graph, firstly the set of external input should be determined according to the specification and implementation of the SCADA protocol. Then, the set of important nodes are marked on the dynamic memory taintedness graph (e.g., control the critical field devices to do a sensitive actions, etc.) according to the protocol type and functional features.

To track the spread of memory taintedness of the important nodes, suppose $Gr$ be the dynamic memory taintedness graph of the SCADA protocol implementation $P$, $InputSet$ refers to the set of external input, $INodeSet$ refers to the set of the important nodes, If the value of important node $Node$ is derived from the value of the external input $Input$, or affected by the value of $Input$, then the node is potentially vulnerable, and the track of the memory taintedness represents the trigger mechanism or exploitation patterns. Algorithm 2 can be used to detect the potential vulnerabilities of the SCADA protocol implementation and the set of the memory taintedness tracks.

Algorithm 2. CheckIcpVuls

// Detection of the potential vulnerabilities of the implementation of SCADA protocol and the tracks of memory taintedness.
Input : Dynamic memory taintedness graph $Gr$, the set of the external input $InputSet$, the set of the pointer of the important nodes $INodeSet$

Output : Potential vulnerabilities and the set of the memory taintedness tracks

1. $VulSet := \emptyset$
2. foreach $node \in INodeSet$ do
3. \hspace{1em} $r := TraceTaint(node, p, node.n)$
4. \hspace{1em} $VulSet := VulSet \cup \{(node, r)\}$
5. endforeach
6. Output $VulSet$

Alg.2 CheckIcpVuls algorithm

After obtaining the potential vulnerabilities and the set of its taintedness tracks $VulSet$, by further analyzing the illegal exploitations of taintedness data for each node in $VulSet$, such as whether using taintedness data as a jump target (return address, function pointer or program pointer offset, etc.), or as an important system call parameters, the vulnerabilities of the protocol implementation can be located more accurately and the trigger mechanism be found automatically.

4 Experimental Results
4.1 Experimental Description
In order to further elaborate the above proposed vulnerability analysis method for the implementation of SCADA protocol, we analyze the vulnerabilities of Gesytec’s Easylyn OPC Server, the experimental environment is shown in Fig. 2.

![Fig.2 The experimental environment for vulnerability analysis](image)

According to vulnerability note VU # 205073 [12] (CVE-2007-4473 [13]) released by US-CERT, there are memory access related vulnerabilities in Gesytec’s Easylyn OPC server. This experiment was carried out with Windows XP environment and Easylyn OPC server version 2.30.32.

4.2 Experimental Analysis
The vulnerability existed in this OPC server is mainly due to the implementation of OPCDA specification failed to verify the input handle. OPC Server handle (handle) is a 32-bit identifier, generated by the OPC server, and it uniquely identifies an OPC server, group or data item. It uses the 32-bit memory address pointer of an OPC server, group or data item as its handle in the implementation of the OPC server. In the operation of read, write, or delete an object, the handle is used as an input parameter of client function call request. Therefore, the failure to verify the handle parameters properly leads to the emergence of vulnerabilities in related functions, including:

- HRESULTIOPCServer::AddGroup()
- HRESULTIOPCServer::RemoveGroup()
Through experimental analysis of the Easylon OPC server, we have successfully located the vulnerability and identified the associated trigger mechanisms. The detailed experimental procedure is as follows:

1) Opens the Easylon OPC server with Ollydbg in debug mode;
2) Connects to Easylon OPC server with an OPC client, reads and writes item1 normally;
3) Removes item1 from Easylon OPC server;
4) Connects to Easylon OPC server with the OPC client again, reads and writes item1, which is equivalent to the client requesting a data item that does not exist, and also means the access of the memory area that does not exist. At that time, Easylon OPC server generated a read/write exception and Ollydbg generated an exception interrupt:

Access violation: Read[029C948B]
The valid memory region is [029C4000-029C9000] when the exception occurs. As can be seen, 029C948B outside of the valid area of memory;

5) Ignores this exception and continue to debug, after several steps, the program crashed. By capturing the error report, the program crash occurred as offset: 00018591, it definitely is the location of the memory access violation mentioned above.

Through this experiment we verified the existence of memory access vulnerability in Easylon OPC server v2.30.32: When the OPC client accesses the OPC server handles that does not exist, the server will crashed due to abnormal memory read and write.

Since the occurrence of this vulnerability type is related to the memory access, thus we can analyze it with the model and algorithm proposed in this paper. According to the memory taintedness model described in section 2, the dynamic memory taintedness of this vulnerability can be described as Fig. 3.

![Fig. 3 The memory dynamic taintedness of Easylon OPC server vulnerability](image)

In Fig.3, rectangular node represents memory buffer or variable. Oval node represents external input, that is, sources of memory taintedness. Directed edges between nodes represent the track of memory taintedness. Through the detection algorithm it can be found that there are no any mechanism for checking or exception handling from the input (OPC handles) to the specific memory region (OPC data item), therefore there exists a path between the source of memory taintedness and specific memory region, which means the existence of memory access vulnerability.

There are two methods for triggering the vulnerability. One is the client calling the removeGroup() function, which performs the delete operation to the server group object, and then the server will call free() function to release the memory allocated for the object. Since the server does not check the validity of the input parameters strictly, assigned specific value to the input parameter hServerGroup, the server will release memory address unallocated and result in server crash.

The second is the client calling the Write() function, performing a write operation to the data items on the OPC server, To assign specific values to the input parameters, such as handle phServer and data item pItemValues. As the server did not check the input parameters strictly, it can write any value in any address of the server and lead to execute arbitrary code.

5 Conclusions
This paper proposes an analysis method of memory access vulnerability in SCADA protocol implementation, including the memory taintedness model of SCADA protocol implementation and the detection algorithm of memory access vulnerability based on this model. Through the experiment of vulnerability analysis for Easylon OPC server, the model and algorithm are validated.

As this study continues, researches will be dedicated on improving the accuracy of memory access vulnerability detection algorithm, development of prototype system and extensively experimental analysis.

References:


