Peach Improvement on Profinet-DCP for Industrial Control System Vulnerability Detection

Dianbo Zhang[1, a], Jianfei Wang[2, b] Hua Zhang[1, c]

[1] State Key Laboratory of Networking and Switching Technology

Beijing University of Posts and Telecommunications;

[2]ZheJiang Xinlan Network Media Limited Company.

[a]dianbo123@126.com , [b]hz14131@163.com , [c]zhanghua_288@bupt.edu.cn

Keywords: Peach, Fuzzing, Profinet-DCP, PLC, Industrial Safety.

Abstract. With the development of ICS, PLC and SCADA systems are interconnected with Ethernet and directly connected to internet, which greatly improve the efficiency of data sharing and introduced in security threats at the same time. Once crack fault occurrence of critical infrastructure will result in casualties and great economic loss. Peach Fuzzer is an advanced and extensible fuzzing platform and is restricted to those with TCP/UDP-based protocols on Windows Platform, the PN-DCP would not be supported without publisher to send PDU correctly. So it is urgent to develop an additional publisher for PN-DCP. In this paper, we propose a novel Peach improvement on Profinet-DCP for industrial control system vulnerability detection. We analyze the importance of vulnerability detecting for PN-DCP with Peach Fuzzer. Then, introducing the Peach Framework, the hierarchy of Profinet-DCP and the PitFile of Profinet-DCP. We also evaluate our approach through experiments, the results can fully satisfy the requirement of vulnerability detecting of PN-DCP on Peach platform.

Introduction

Many of our critical infrastructures are controlled by cyber-physical systems responsible for monitoring and controlling various processes [1]. The supervisory control and data acquisition (SCADA) systems are responsible for a wide range of industrial processes e.g. manufacturing, power generation, refining, as well as infrastructure, e.g. water management, oil and gas pipelines, wind farms, and facilities， e.g. airports, space stations, buildings, etc. The importance of monitoring and control, which heavily relies on such cyber-physical systems, is paramount for world economies in various industrial sectors [2].

Stuxnet [3] was firstly released in the summer of 2010, which aimed for highly specialized industrial systems in critical high-security infrastructures. Stuxnet is one of the most sophisticated worms that hit cyber world. In injection phase when it finds its actual target i-e Siemens WinCC control and monitoring system connected to programmable logic controllers (PLC) it starts functioning and deviates them from their normal behavior .Common predictions are that this worm can target either Bushehr Nuclear power plant or Natanz nuclear facility in Iran [4]. Therefore, more and more attention has been made on industrial security problems.

Regarding the importance of industrial control system (ICS), the absence of security vulnerability will cause serious consequences. Peach Fuzzer [5] is a test tool that attempts to discover security vulnerabilities by sending random input to an application. It is widely used to test for security bugs in input validation as well as in the application logic. However, Peach cannot fuzz the protocols based on layer 2 protocols such as PROFINET Discovery and basic Configuration Protocol (PN-DCP) [6].

Regarding this problem, we propose a novel Peach improvement on Profinet-DCP for industrial control system vulnerability detection. We analyze the importance of vulnerability detecting for PN-DCP with Peach Fuzzer. Then, introducing the Peach Framework, the hierarchy of Profinet-DCP and the PitFile of Profinet-DCP. We also evaluate our approach through experiments, the results can fully satisfy the requirement of vulnerability detecting of PN-DCP on Peach platform.

The rest of this paper is organized as follows: in Section II, we briefly describe the Peach Framework and the hierarchy of Profinet-DCP; in Section III, we present the PitFile of Profinet-DCP and our new approach for Peach improvement; in Section IV, we demonstrate the experimental results by Pcap packet and analysis our data, and we summarize this paper in section V.

Related Works

Peach Fuzzer Overview.

In this section, we introduce Peach Fuzzer. Security Fuzz Testing (Fuzzing) finds flaws and vulnerabilities in technology solutions through data mutation. Attackers use fuzzing to discover system vulnerabilities that can lead to zero-day attacks. Companies wishing to eliminate external fuzzing use specialized software to fuzz their own products, thereby revealing and correcting vulnerabilities during the software and hardware development lifecycles.

What happens during Fuzzing?

① Unexpected data is sent to a system repeatedly in order to cause unintended consequences (like a crash, or unstable state).

② These unintended consequences are observed and relevant data is captured.

③ The captured data is analyzed to discover potential vulnerabilities.

Peach Fuzzer is available in four editions: the Community Edition, the Professional Edition, the Consulting Edition and the Enterprise Edition. In this paper, our improvement is based on the Community Edition which is open source edition.

Peach Fuzzer Community edition is an advanced and extensible fuzzing platform. This software has been developed to enable security consultants, product testers and enterprise quality assurance teams to find vulnerabilities in software using automated generative and mutational methods.

The primary capabilities of the Peach Fuzzer ecosystem are: intelligent fuzzing, robust support, ease of use and scalability.

The Peach Fuzzing Platform was designed to speed up the development of fuzzers for both security researchers, security teams, consultants and companies. Peach accomplishes this by separating out modeling of the data and state systems being fuzzed and the actual fuzzing engine. Peach also provides a robust agent/monitoring system to allow for monitoring a fuzzing run, detecting faults (bugs), etc. All the major components of Peach are pluggable and extensible allowing for infinite flexibility. A typical Peach network fuzzing scenario is shown in Fig. 1.

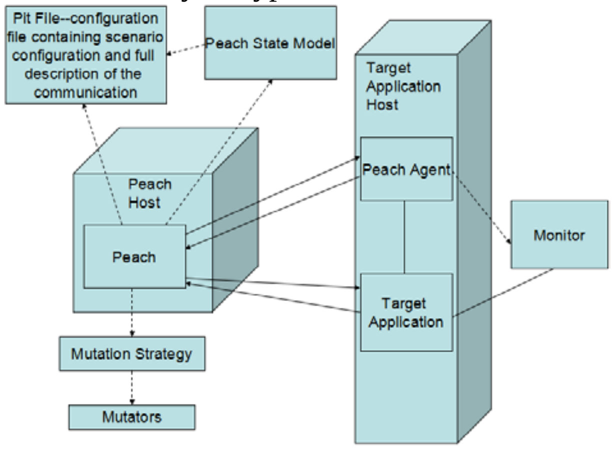


Fig. 1 Peach Fuzzing Scenario

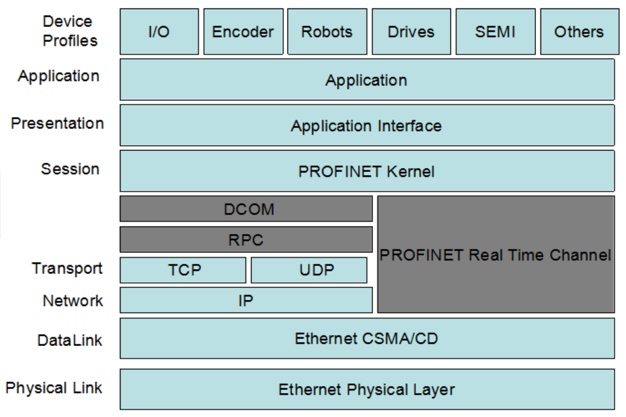


Fig. 2 Hierarchy of PROFINET protocol

In a usual scenario Peach acts as a client and communicates with a single remote application. It alters requests specified in the configureation and tries to generate an input for the remote application, which would cause a crash or other kind of undesired behavior. It uses a so called Peach Agent to maintain control over the target application.

Introduction of PN-DCP.

In this section we explain the Discovery and Configuration Protocol which we use to discover the Profinet networks. Profinet is a standard for Industrial Ethernet based on Industrial Ethernet according to IEEE 802.xx [7]. The Discovery and Basic Configuration Protocol (DCP) is a protocol definition within the PROFINET context. It is a Data Link Layer based protocol to configure station names and IP addresses. It is restricted to one subnet and mainly used in small and medium applications without an installed DHCP server [8]. The hierarchy of protocol is shown in Fig. 2.

Improvement on Peach Fuzzer Framework for Profinet-DCP

Peach Fuzzer is arguably the most established, freely available fuzzer out there. It has tons of built in functionality to support a huge range of features. But the Peach Fuzzer cannot fuzz the protocols based on Ethernet protocols, such as Profinet-DCP[9]. We just need a Profinet publisher to solve this problem. In this section we'll show how to write and compile a Profinet publisher so we can spend all our CPU cycles fuzzing the stuff that matters.

Publishers are located within the Peach.Core/Publishers directory of the source package. Basically every publisher inherits from the Publisher class (Peach.Core/Publisher.cs) and should override a few key functions that are tied back to the corresponding Action Types referenced in the Peach PitFile. The following table provides a summary of those functions (all are of type protected virtual void unless otherwise noted and descriptions are from the Publisher.cs source).

Table 1 The function of Publisher

Function	***Description***
OnStart( )	Be called once per fuzzing "Session", not on every iteration, when the publisher is started.
OnStop( )	Be called once per fuzzing "Session", not on every iteration, when the publisher is stopped
OnOpen( )	Open or connect to a resource. Be called automatically if not called specifically.
OnClose( )	Close a resource. Be called automatically when state model exists, or when needed.
OnAccept( )	Accept an incoming connection.
OnInput( )	Read data
OnOutput( )	Send data
Protected virtual variant OnCall( )	Call a method on the Publishers resource
OnSetProperty( )	Set a property on the Publishers resource
Protected virtual Variant OnGetProperty ( )	Get value of a property exposed by Publishers resource

Depending on the purpose of the Publisher, some of the above functions are more important than others. For our improvement, we are mainly concerned with modifying the output of data right before its sent, then we would override OnOutput( ).

The visual component of the Peach Fuzzer is programmed using the C# programming language on .NET Framework 4. This section presents an implementation of Profinet Publisher by means of the SharpPcap library [10].

First up we'll start out by making a copy of the TcpPublisher which extends the BufferedStreamPublisher class. We'll set the name for Profinet Publisher that will be referenced in the PitFile by replacing "Tcp" with "Profinet" on line 23:

```
[Publisher("Profinet", true)]
```

And name the class of Profinet publisher by replacing "TcpPublisher" with "ProfinetPublisher" on line 27:

```
public class ProfinetPublisher : BufferedStreamPublisher
```

And line 34: `public ProfinetPublisher(Dictionary<string, Variant> args)`

We have Profinet Publisher all done. Granted, its really a waste at this point since its exactly the same thing as the TcpPublisher, but nonetheless it's still Profinet Publisher.

We'll need to do in Profinet Publisher is override the OnOutput function so that we can modify the data before its sent. So we'll add a new line after line 53 and insert:

```
protected override void OnOutput(byte[] buffer, int offset, int count) {   }
```

Then comes the program body, we need to define a device with the SendPacket's function. Since the limitation about the data based on MAC layer, we first need to put in some intelligence that ensures the max length of data is less than 1514 Bytes [11].

Table 2 The Program body

```
var devices = CaptureDeviceList.Instance;
             ICaptureDevice device = devices[1];
            device.Open();
             byte[] bufferPart = null;
             if (buffer.Length >= 1514)
            {
                bufferPart = new byte[1514];
               Array.Copy(buffer, bufferPart, 1514);
             }
           if (bufferPart == null)
           {
              device.SendPacket(buffer, count);
           }
           else
           {
              device.SendPacket(bufferPart);
           }
        device.Close();
```

Table 3 The <Test> Tag of PitFile

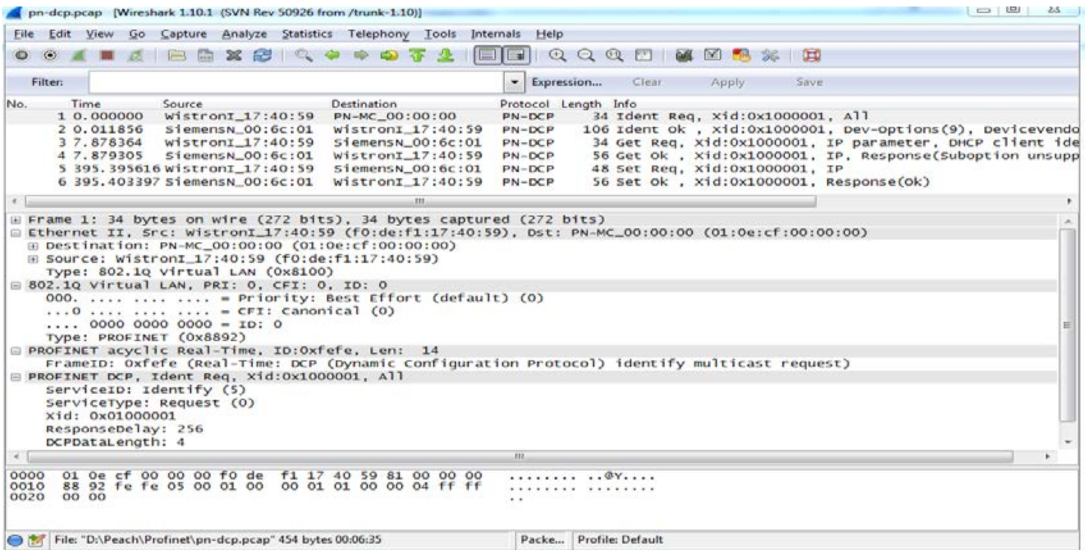
```
<Test name="Default">
        <Agent ref="Local" />
        <StateModel ref="TheStateModel"/>
        <Publisher class="ProfinetPublisher" />
        <Strategy class="Sequential"/>
        <Logger class="Filesystem">
          <Param name="Path" value="Logs" />
        </Logger>
 </Test>
```

This can be an controversial move and it is an important note to make about Profinet Publisher. Our intention is to fuzz the PN-DCP and as part of that fuzzing, we should be trying really long strings and other values. By implementing this limitation we are effectively limiting our test cases. It might be worthwhile just to forget about an accurate value in the buffer length as it might lead to more vulnerabilities.

After modifying the OnOutput function, the Profinet Publisher is Written.On to testing. The next thing we need to do is call Profinet Publisher from a PitFile via the <Publisher> tag within our Test definition. The <Test> tag of PitFile is shown in Table 3.

The full Peach PitFile for this project can be found in section IV.

We'll run the instances of our fuzz case and use Wireshark [12] to inspect output on the wire. As shown in Fig. 3, we can see the PN-DCP Type, FrameID, Service and included within the data is the content of PitFile's DataModel.



pn-dcp.pcap [Wireshark 1.10.1 (SVN Rev 50926 from /trunk-1.10)]
File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help
Filter:
Expression... Clear Apply Save
Packe...
Profile: Default

```
No.     Time       Source               Destination          Protocol Length Info
      1 0.000000   WistronI_17:40:59    PN-MC_00:00:00       PN-DCP      34 Ident Req, Xid:0x1000001, All
      2 0.011856   SiemensN_00:6c:01    WistronI_17:40:59    PN-DCP     106 Ident Ok , Xid:0x1000001, Dev-Options(9), DeviceVendo
      3 7.878364   WistronI_17:40:59    SiemensN_00:6c:01    PN-DCP      34 Get Req, Xid:0x1000001, IP parameter, DHCP client ide
      4 7.879305   SiemensN_00:6c:01    WistronI_17:40:59    PN-DCP      56 Get Ok , Xid:0x1000001, IP, Response(Suboption unsupp
      5 395.395616 WistronI_17:40:59    SiemensN_00:6c:01    PN-DCP      48 Set Req, Xid:0x1000001, IP
      6 395.403397 SiemensN_00:6c:01    WistronI_17:40:59    PN-DCP      56 Set Ok , Xid:0x1000001, Response(Ok)

⊞ Frame 1: 34 bytes on wire (272 bits), 34 bytes captured (272 bits)
⊟ Ethernet II, Src: WistronI_17:40:59 (f0:de:f1:17:40:59), Dst: PN-MC_00:00:00 (01:0e:cf:00:00:00)
  ⊞ Destination: PN-MC_00:00:00 (01:0e:cf:00:00:00)
  ⊞ Source: WistronI_17:40:59 (f0:de:f1:17:40:59)
    Type: 802.1Q Virtual LAN (0x8100)
⊟ 802.1Q Virtual LAN, PRI: 0, CFI: 0, ID: 0
    000. .... .... .... = Priority: Best Effort (default) (0)
    ...0 .... .... .... = CFI: Canonical (0)
    .... 0000 0000 0000 = ID: 0
    Type: PROFINET (0x8892)
⊟ PROFINET acyclic Real-Time, ID:0xfefe, Len:  14
    FrameID: 0xfefe (Real-Time: DCP (Dynamic Configuration Protocol) identify multicast request)
⊟ PROFINET DCP, Ident Req, Xid:0x1000001, All
    ServiceID: Identify (5)
    ServiceType: Request (0)
    Xid: 0x01000001
    ResponseDelay: 256
    DCPDataLength: 4

0000  01 0e cf 00 00 00 f0 de  f1 17 40 59 81 00 00 00   ........ ..@Y....
0010  88 92 fe fe 05 00 01 00  00 01 01 00 00 04 ff ff   ........ ........
0020  00 00                                              ..

File: "D:\Peach\Profinet\pn-dcp.pcap" 454 bytes 00:06:35
```

Fig. 3 PCAP Packets of PN-DCP

Experiment and Analysis

To prove the validity of the improvement mentioned in the above section, we write the PitFile based GetRequest service with XML, the generation based mutation mechanism is employed for the test case generating. The Profinet-DCP client is implemented by Microsoft to generate legal PN-DCP packets for testing. To extend the fuzzing field, the PLC and WinCC is tested together [13, 14, 15].

Experiment Design.

The topology of the experiment is designed like Fig. 4, which composed of a PLC(the target), a Wincc (the target) and the Profinet-DCP Fuzzer for testing. Peach Fuzzer requires the creation of Peach PitFiles that define the structure, type information, and relationships in the data to be fuzzed. The PitFile item can be described as a four Tuples, includes (Data Modeling, State Modeling, Agents and Monitors, Test Configuration) to test the devices of Profinet-DCP[16]. The example of the PitFile of PN-DCP.GetRequest is shown in Table 4.

Table 4 The PitFile of PN-DCP.GetRequest

```
<DataModel name="ProfinetDCPGetRequest">
    <Blob name="DestinationMACAddress" value="00 1c 06
01 1a 06" valueType="hex" mutable="false" />
    ......
    <Number              size="16"              signed="false"
name="DCPDataLength" valueType="hex" endian="big">
        <Relation type="size" of="DCPData" />
    </Number>
    <Block name="DCPData" minOccurs="1" >
     <Blob name="OptionIP" value="01" valueType="hex"  />
    ......
    </Block >
</DataModel>
<StateModelname="TheStateModel"initialState="TheState">
   <State name="TheState">
    <Action type="output">
     <DataModel ref="ProfinetDCPGetRequest" />
    </Action>
   </State>
</StateModel>
<Test name="Default">
        ......
        <Publisher class="ProfinetPublisher" />
        ......
 </Test>
```

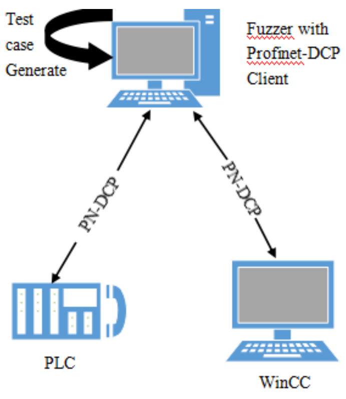
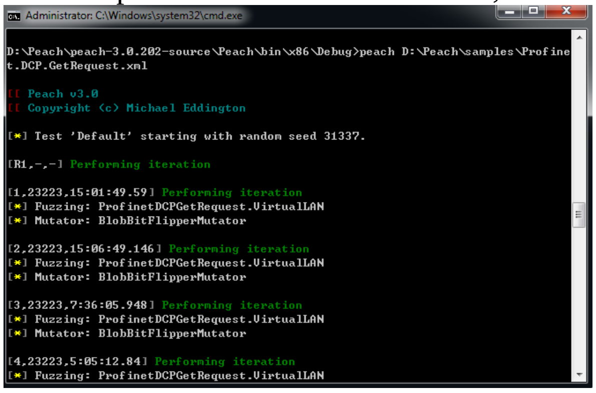


Fig. 4 Topology of Experiment

Result Analysis.

During the experiment process, 23223 Profinet-DCP GetRequest’s test cases and 23223 log items are produced; the console output can be described like Fig. 5. If the PitFile is right, the console return Finished and the log file will be updated with the Fuzz testing. Three service PitFiles are tested by the Peach Fuzzer, partially described as follows and there different Profinet-DCP including Identify, GetRequest and SetRequest are supported totally. The result of the experiment proves that the Profinet Publisher mentioned can satisfy the requirement of the Profinet-DCP testing.

We set up the environment and fuzz it, the result is shown in Table 5.



```
D:\Peach\peach-3.0.202-source\Peach\bin\x86\Debug>peach D:\Peach\samples\Profine
t.DCP.GetRequest.xml

[[ Peach v3.0
[[ Copyright (c) Michael Eddington

[*] Test 'Default' starting with random seed 31337.

[R1,-,-] Performing iteration

[1,23223,15:01:49.59] Performing iteration
[*] Fuzzing: ProfinetDCPGetRequest.VirtualLAN
[*] Mutator: BlobBitFlipperMutator

[2,23223,15:06:49.146] Performing iteration
[*] Fuzzing: ProfinetDCPGetRequest.VirtualLAN
[*] Mutator: BlobBitFlipperMutator

[3,23223,7:36:05.948] Performing iteration
[*] Fuzzing: ProfinetDCPGetRequest.VirtualLAN
[*] Mutator: BlobBitFlipperMutator

[4,23223,5:05:12.84] Performing iteration
[*] Fuzzing: ProfinetDCPGetRequest.VirtualLAN
```

Fig. 5 Console output of the Profinet-DCP fuzzer

Table 5 Test Results

PitFile	Cases	Target	Consequence
Identify	22074	WinCC	Normal
GetRequest	23223		Buffer Overflow
SetRequest	48981		Buffer Overflow

Conclusions

With the specific of industrial Ethernet protocols such as Profinet-DCP, the Peach Fuzzer framework cannot directly applied to them on Windows Platform, and some improvements are seriously needed. In this paper, we proposed a fuzzing approach for Profinet-DCP and described the structure and the workflow. We also evaluated our approach for WinCC fuzzing through experiment, and the experiment results proved that the Peach Fuzzer designed can satisfy the requirement of vulnerability detecting of Profinet-DCP.

Acknowledgement

This work is supported by NSFC (Grant Nos. 61300181, 61202434), the Fundamental Research Funds for the Central Universities (Grant No. 2015RC23).

References

[1] S. Karnouskos, Cyber-Physical Systems in the SmartGrid, IEEE Int. Conf. on Ind. Informatics, July 2011, vol.20, no.23, pp. 26-29

[2] S. Karnouskos, Stuxnet worm impact on industrial cyber-physical system security, Annual Conference on IEEE Ind. Electronics Society, 7-10 Nov. 2011, pp.4490, 4494

[3] Information on https://secure.wikimedia.org/wikipedia/en/wiki/Stuxnet#cite_note-5

[4] R. Masood., U. Um-e-Ghazia and Z. Anwar, SWAM: Stuxnet Worm Analysis in Metasploit, IEEE Conf. on Frontiers of Inf. Technology, Dec 2011, pp. 142-147

[5] Information on http://peachfuzzer.com, accessed 2015.

[6] Information on http://wiki.wireshark.org/PROFINET/ DCP, accessed 2015.

[7] Information on http://w3app.siemens.com/mcms/infocenter/documentencenter/sc/ic/Documents u20Brochures/E20001-A24-M116-X-7600.pdf

[8] Veysel Harun ŞAHİN, İbrahim ÖZCELİK, Topology discovery of PROFINET networks using Wireshark, IEEE Electronics, Computer and Computation, Nov. 2013,vol.88, no.91, pp. 7-9

[9] Han X.,Wen Q.Y.and Zhang Z., A mutation-based fuzz testing approach for network protocol vulnerability detection, IEEE Computer Science and Network Technology, 29-31 Dec. 2012,, pp.1018,1022

[10] Yang C.W., Xu J., and Vyatkin, V., Towards implementation of IEC 61850 GOOSE messaging in event-driven IEC 61499 environment, IEEE Emerging Technology and Factory Automation, 16-19 Sept. 2014, pp.1, 4

[11] Shi K., Yang, O., Shu Y.T.; Lin S.,Wang J.S. and Luo J.R., A Distributed MAC Layer Congestion Control Method to Achieve High Network Performance for EAST Experiments,IEEE Trans. Nuclear Science, Oct. 2013, vol.60, no.5, pp.3758, 3763

[12] Information on http://wireshark.org, accessed 2015.

[13] Jing C.M., Yin X.,Wang Z.L. and Wu J.P., A Formal Approach to Robustness Testing of Network Protocol with Time Constraints,IEEE Young Computer Scientists, 2008: 2168 - 2174

[14]Bi M.G., Huang X., and Huang R., Detection of software vulnerabilities based on Fuzzing, Cyberspace Technology (CCT 2013), Int. Conf. On, 23-23Nov. 2013, pp.192, 195

[15] Gu S.J., Li W.H. and Zhao X., "Brute Force Vulnerability Testing Technology Based on Data Mutation," IEEE Vehicular Technology Conf. (VTC Fall), 5-8 Sept. 2011, pp.1,6

[16]Wang H., Wen Q.Y. , Zhang Z., Improvement of Peach Platform to Support GUI-Based Protocol State Modeling, IEEE Cyber, Physical and Social Computing, 20-23 Aug. 2013, pp.1094, 1097