Testing UPnP Internet Gateway Devices with Faulty Packets

Jangbok Kim[1], Minsik Kim[1], Kyunghee Choi[1], Kihyun Chung[2], Daniel Hoffman[3], Kevin Yoo[3]
[1] Graduate School of Information and Communication, Ajou University,
Suwon, 442-749, South Korea
[2]School of Electrics Engineering, Ajou University,
Suwon, 442-749, South Korea
[3]Department of Computer Science, PO Box 3055 STN CSC Victoria,
BC Canada V8N 4C9

Abstract: - UPnP devices are now widely available in the SOHO market, making device conformance and reliability important. UPnP conformance is supported by automated test suites from the UPnP Implementor's Corporation (UIC). While the UIC test suite is helpful, it covers normal case tests only and performs no semantic checks on the device.

We present the UPnP Device Tester (UDT), a test framework for UPnP devices. Both normal and exceptional tests are supported and semantic checks are performed. UDT has been used to test two commercial Internet Gateway Devices, revealing serious errors.

Key-Words: - UPnP Testing, Protocol Testing

1 Introduction

Universal Plug and Play (UPnP) is a communications architecture and protocol standard for seamlessly connecting devices in the SOHO (small-office/home office) environment.

UPnP functionality is divided between devices and control points. UPnP devices are typically small embedded systems:

- Internet gateway devices (IGDs), such as home routers,
- multimedia devices, such as audio and video players, and
- home security/home automation devices.

UPnP control points are often PCs, although there are other possibilities. The UPnP protocols are carried primarily by HTTP over TCP/IP; Layer 2 is typically supplied by Ethernet.

Certification of UPnP devices is controlled by the UPnP Implementers Corporation (UIC) [1]. The UIC provides an automated test suite used by UIC members for device conformance testing [2]. With the test suite, syntactically correct messages are sent to UPnP devices. Checks for correct device behavior are limited, perhaps because of the wide variety of UPnP device functionality.

While failure in a UPnP media player is merely irritating, failure in an IGD or home security device can be safety critical. Therefore, it is important to extend UPnP testing to include abnormal network traffic.

We present a framework for automated testing of UPnP devices. The framework purposely seeds syntactic and semantic faults in UPnP packets. The framework has been applied to two commercial IGDs, revealing serious faults. The framework can easily be applied to other UPnP device types.

Section II presents our approach to specifying faulty packets. Section III describes the software used to generate packets for transmission and monitor device behaviour. Section IV summarizes the test results.

2 Fault Specification

Two types of faults were used in our tests: syntactically faulty packets and semantically faulty packets.

2.1 Syntactic faults

The UPnP protocol consists of six phases [3].

1) *Addressing.* A UPnP device acquires an IP address with DHCP or Auto IP.

2) *Description.* Devices announce their presence and advertise services they provide on the network.

3) *Discovery.* Control points find devices and services on the network and retrieve information about them.

4) *Control.* Control points use services provided by a device.

5) *Eventing.* Control points monitor state changes in a device through a publish/subscribe model.

6) *Presentation.* A device presents a web-based user interface for manual control and event notifications.

```
(1) POST /uuid:0014-bf92-7b9d020099dc/WANIPConnection:1 HTTP/1.1
(2)
(3) SOAPAction: "urn:schemas-upnp-org:service:WANIPConnection:1#AddPortMapping"
(4) Content-Length: 701
(5)
(6) <?xml version="1.0" encoding="utf-8"?>
(7) <NewRemoteHost>192.168.1.11</NewRemoteHost>
(8) </s:Envelope>
```

Fig. 1. UpnP Message

The first three UPnP phases occur in sequence; however, the last three phases may occur in any order.

A UPnP message is made up of three sections: a method declaration, a list of fields, and a body. UPnP typically uses HTTP, but other protocols may also be used. Figure 1 shows an example message. The method is declared in Line 1, the fields are in Lines 3-4, and the body is in Lines 6-8. For this paper, a UPnP message is referred to as a Protocol Data Unit (PDU).

A PDU that contains incorrect field names or illegal or incorrect values is considered syntactically faulty. For instance, in the second line of Figure 1, if the field name SOAPAction is instead spelled SOPAAction, then the PDU contains a syntactic fault. While syntactic faults can appear in any of the three parts of a PDU, UDT only injects faults into the fields and body sections.

Both the body and _elds sections consist of a list of namevalue pairs, which we'll call elements. In the field section, an element has the following form: *name: value.* In the body section, an element looks like the following:

<name>value</name>

Faults are injected into PDUs by manipulating these elements.

UDT provides six different ways of manipulating elements. Text can be inserted, appended, replaced, deleted in two different ways, and duplicated. All of an element, or just its name or value, may be modified in one of these six ways. Therefore, there are a total of 18 different ways of creating faulty elements. Of these 18 methods, 11 are provided by UDT, as the other seven are either equivalent to the supported methods or they do not provide a useful syntactic fault. An example of each of the 11 supported methods is depicted in Table 1. In the table, **xx** represents text that has been added by the fault injection method; text that has a ~~strikethrough~~ represents text that has been removed. For the remainder of this paper, faults are denoted as F_{OP}, where O stands for one of the six manipulation operations, and P denotes which part of the element is being modified. For instance, when text is inserted into an element's value, this fault is represented as F_{IV}. The symbol for each operation and element part can be found in Table 1.

2.2 Semantic Faults

A PDU is semantically faulty when it cannot be correctly processed by a device because the device is not in the proper state. Generally, the validity of UPnP messages is not dependent on the device state. However, this property does not hold during UPnP's Eventing stage. A device enters the Eventing stage when it receives a Subscribe message, and it ends when it receives an Unsubscribe message; a Renew message extends the Eventing stage. While Subscribe messages can be sent at any time, Unsubscribe and Renew messages should only be sent during the Eventing stage. UDT sends semantically faulty PDUs by sending Unsubscribe and Renew messages outside the Eventing stage. Note that for our tests, semantically faulty PDUs were syntactically correct.

2.3 Related Work

Fault-injection methods have been used to test other systems. Hsueh and Tsai provide an overview of a generic fault-injection system, as well as describe existing fault-injection tools [4]. Fault-injection testing has been used to test web services [5] and OSPF [6]. The ASPIRE testing tool [7] uses syntactically and semantically faulty packets to test HTTP and SMTP.

3 Test Generation and Execution

This section explains how the tester creates faulty packets with UDT and describes the setup and methodology for testing IGDs.

3.1 Test Generation

To generate faulty UPnP packets, UDT requires three input files: a PDU description file, a fault specification file, and a test scenario file. All of these files are written in XML.

1) *PDU Description File*: The PDU description file (PDF) specifies the contents of a PDU. One file can describe multiple PDUs, so each PDU is assigned a unique ID which will be referenced in the test scenario file. The PDU descriptions do not include any syntactic faults, and so only syntactically correct PDUs are specified. Separating the PDU description from the fault

	Value (V)	**Name (N)**	**Element (E)**
Insert (I)	<Tag>CC**xx**NC</Tag>	<T**xx**ag>CCND</T**xx**ag>	N/A
Append (A)	<Tag>CCNC**xx**</Tag>	<Tag**xx**>CCND</Tag**xx**>	N/A
Replace (R)	<Tag>CC**xx**</Tag>	<T**xx**>CCND</T**xx**>	N/A
Delete Portion (P)	<Tag>CC~~NC~~</Tag>	<T~~ag~~>CCNC</T~~ag~~>	N/A
Delete All (Q)	<Tag>~~CCNC~~</Tag>	N/A	~~<Tag>CCNC</Tag>~~
Duplicate (D)	N/A	N/A	<Tag>CCNC</Tag> **<Tag>CCNC</Tag>**

Table 1. Syntactic Faults

```
<Fault FieldName = "Callback">
  <OP fpa="VALUE" fop="INSERT" fstart="3" />
  <OP fpa="VALUE" fop="BLANK" />
  <OP fpa="NAME" fstring="fij" />
</Fault>
```

Fig. 2. Syntactic Faults

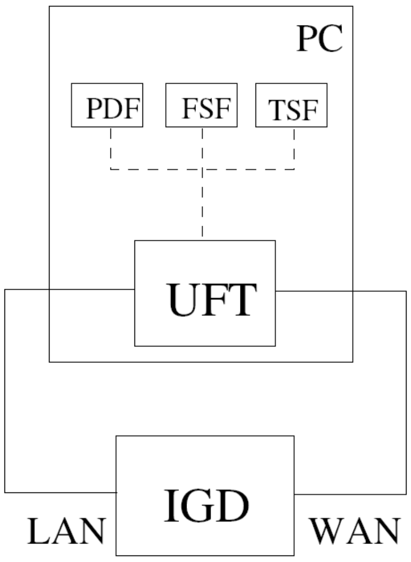

Fig. 3. Test Setup

specification improves file reusability and reduces the tester's workload. For instance, a tester can inject different faults into the same PDU without having to repeatedly respecify identical PDU contents.

2) *Fault Specification File*: The fault specification file (FSF) specifies where and how to insert syntactic faults into the PDUs described in the PDF. An FSF snippet is depicted in Figure 2, which shows how to insert faults into the Callback element. Each *OP* tag corresponds to one of the 11 syntactic fault types supported by UDT. The *fpa* attribute specifies which part of the element will be modified and *fop* indicates which element manipulation operation is performed. Other attributes, such as *fstart*, *fstring*, and *flength* are only applicable to certain operations; they allow the tester to have greater control over how the text is manipulated. All attributes are optional, and if any are missing, then UDT randomly assigns them values.

3) *Test Scenario File*: A test scenario is a sequence of *PDUs, some of which are faulty. A test scenario file* (TSF) allows a tester to specify one or more of these test scenarios.

3.2 Test Setup

The setup used in our tests is depicted in Figure 3. A PC running UDT was connected to one of two commercial UPnP Internet gateway devices: IGD A and IGD B.

3.3 Test Methodology

The purpose of the tests is to determine how reliable the two UPnP devices are when faced with syntactically and semantically faulty PDUs. The PDUs used in our tests are summarized in Table 2. Note that these PDUs cover four of the six UPnP stages. The addressing and presentation steps are not tested.

1) *Syntactic Tests*: Each test consists of sending one of the PDUs in Table 2 with a single syntactic fault. There are eleven different syntactic faults that can be injected into a PDU field. Therefore, the total number of syntactic tests is the total number of fields in all of Table 2's PDUs multiplied by 11.

2) *Semantic Tests*: The semantic tests use the same PDUs that are depicted in Table 2. However, no syntactic faults are injected into these PDUs. For our tests, PDUs 5 and 6, which are used to renew and terminate subscriptions, respectively, act as our semantically faulty packets.

3) *Oracle*: For both the syntactic and semantic tests, after a faulty packet is sent, four syntactically and semantically correct PDUs are sent:

- A PDU that subscribes to the WANCommonIFC service.
- A PDU that subscribes to the WANIPConn1 service.
- A PDU that terminates the WANCommonIFC subscription.
- A PDU that terminates the WANIPConn1 subscription.

PDU ID	Description	UpnP Step	# of Elements	Element Names
1	Announce that the device has joined the network	Discovery	3	ST, MX, MAN
2	Get information about RootDevice	Description	1	Content-length
3	Get information about OSInfo Service	Description	1	Content-length
4	Subscribe to OSInfo service	Eventing	6	NT, Callback, Timeout, User-Agent, Content-length, Pragma
5	Renew OSInfo Subscription	Eventing	1	TIMEOUT
6	Terminate OSInfo Subscription	Eventing	3	User-Agent, Content-Length, Pragma
7	Request to perform a service	Control	11	SOAPAction, CONTENT-TYPE, Content-Length, SOAP.NewRemoteHost, SOAP.NewExternalPort, SOAP.NewProtocol, SOAP.NewInternalPort, SOAP.NewInternalClient, SOAP.NewEnabled, SOAP.NewPortMappingDescription, SOAP.NewLeaseDuration

Table 2. Test PDUs

PDU	**Behavior**
1	Correct Behavior
2	Correct Behavior
3	Correct Behavior
4	No Response
5	Error Message Sent
6	Error Message Sent
7	No Response

Table 4. IGD A Behavior when Severely Compromised

If the IGD responds to these four packets abnormally, then it can be concluded that the faulty packet has somehow damaged the IGD.

After the four UPnP packets are sent, a single non-UPnP UDP packet is sent to determine if the IGD can still forward IP packets. In this manner, the IGD's non-UPnP functionality is tested, making it possible to determine if breaking the IGD's UPnP functionality will also compromise the device's other services.

4 Test Results

4.1 Syntactic Fault Test

Syntactic fault test results are grouped into three categories, depending on how the IGD responds to the syntactically faulty packet and subsequent UPnP requests.

1) *Correct Behavior*. The IGD sends a response packet indicating that it has received an illegally formed UPnP packet. Subsequent UPnP packets are handled correctly.
2) Slightly Compromised. The IGD does not send a response packet indicating that it has received an illegally formed UPnP packet. Furthermore, in some cases, the IGD provides the requested UPnP service despite the syntactic error contained in the packet. However, subsequent UPnP packets are handled correctly.
3) *Severely Compromised*. Some or all UPnP services are no longer provided.

Table 3 summarizes the test results for both of the IGDs. The table only shows the results for tests that resulted in the IGD being slightly or severely compromised. Faults injected into PDUs 2, 3, and 6 did not produce deviant behavior in either IGD. For IGD A, faults injected into two PDU elements resulted in the device being severely compromised, whereas only one syntactic fault did the same for IGD B. IGD B appears to be particularly susceptible to faults in the body portion of a PDU, since at least one fault in each body element *(SOAP.*)* resulted in the IGD being slightly compromised.

When IGD B was severely compromised, it did not send any response packets nor did it provide any UPnP services. Upon closer inspection, it was determined that the device had closed two ports that were usually open: UDP port 1900 and TCP port 5431.

IGD A exhibited different behavior in its severely compromised state, as shown in Table 4. Note that the PDU number is the same as in Table 2. Unlike IGD B, some UPnP services, such as those in the Description and Discovery steps, were still provided. However, Eventing and Control services were no longer available.

For both devices, normal behavior can be restored by rebooting the devices. Also, while both IGDs were severely compromised, they were still able to act as gateways. Therefore, while the IGDs were no longer able to provide all of their UPnP services, they could still handle non-UPnP packets correctly.

PDU	Element	IGD A		IGD B	
		Minor Error	Severe Error	Minor Error	Severe Error
1	ST	F_{DF}			
	MX	F_{DF}			
	MAN	F_{DF}			
4	NT	F_{IV}, F_{AV}, F_{RV}, F_{PV}, F_{QV}, F_{DF}		F_{IV}, F_{AV}, F_{RV}, F_{PV}, F_{QV}, F_{IN}, F_{AN}, F_{RN}, F_{PN}, F_{DF}, F_{QF}	
	CallBack	F_{IV}, F_{AV}, F_{RV}, F_{PV}, F_{DF}	F_{RV}, F_{PV}	F_{IV}, F_{AV}, F_{PV}, F_{DF}, F_{QF}	
	TimeOut	F_{IV}, F_{AV}, F_{RV}, F_{PV}, F_{QV}		F_{IV}, F_{AV}, F_{RV}, F_{PV}, F_{QV}, F_{DF}	F_{IN}, F_{AN}, F_{RN}, F_{PN}, F_{QF}
5	TIMEOUT	F_{IV}, F_{AV}, F_{RV}, F_{PV}, F_{QV}, F_{QF}		F_{IV}, F_{AV}, FRV, F_{PV}, F_{QV}, F_{DF}	F_{IN}, F_{AN}, F_{RN}, F_{PN}, F_{QF}
7	SOAPAction	F_{IV}, F_{DF}		F_{IV}, F_{AV}, FRV, F_{PV}, F_{QV}, F_{IN}, F_{AN}, F_{RN}, F_{PN}, F_{DF}, F_{QF}	
	CONTENT-TYPE	F_{IV}, F_{AV}, F_{RV}, F_{PV}, F_{QV}, F_{IN}, F_{AN}, F_{RN}, F_{PN}, F_{DF}, F_{QF}		F_{IV}, F_{AV}, F_{RV}, F_{PV}, F_{IN}, F_{AN}, F_{RN}, F_{PN}, F_{QF}	
	Content-Length	F_{AV}, F_{DF}		F_{AV}	
	SOAP.NewRemoteHost	F_{IV}, F_{AV}, F_{RV}, F_{PV}, F_{QV}, F_{AN}, F_{DF}		F_{RV}	
	SOAP.NewExtenalPort	F_{AN}, F_{DF}			
	SOAP.NewProtocol	F_{IV}, F_{AV}, F_{RV}, F_{PV}, F_{QV}, F_{AN}, F_{DF}		F_{AV}, F_{RV}	
	SOAP.NewInternalPort	F_{AN}, F_{DF}			
	SOAP.NewInternalClient	F_{IV}, F_{AV}, F_{RV}, F_{PV}, F_{QV}, F_{AN}, F_{DF}			
	SOAP.NewEnabled	F_{AN}, F_{DF}		F_{IV}, F_{AV}, F_{RV}, F_{PV}, F_{QV}	
	SOAP.NewPortMappingDescription	F_{IV}, F_{AV}, F_{RV}, F_{PV}, F_{QV}, F_{AN}, F_{DF}			
	SOAP.NewLeaseDuration	F_{AV}, F_{DF}		F_{IV}, F_{PV}, F_{QV}	

Table 3. Syntactic Fault Test Results

4.2 Semantic Fault Test

Renew and Unsubscribe messages were sent at three different times:

1) Before a Subscribe PDU was sent.
2) After a Subscribe/Unsubscribe pair was sent.
3) After a subscribe message was sent, and after the subscription had expired. UPnP subscriptions automatically expire after a certain amount of time has elapsed.

In each of the above cases, both IGDs A and B dealt with these semantically faulty packets correctly; an error message was sent, and subsequent packets were handled correctly.

5 Conclusion

UPnP devices are now widely available in the SOHO market, making device conformance and reliability important. While the UIC test suite is helpful, it covers normal case tests only and performs no semantic checks on the device.

We have presented UDT, a test framework for UPnP devices which supports normal and exceptional tests, and semantic checks. UDT has been used to test two commercial Internet Gateway Devices, revealing serious errors. By modifying UDT's Fault Speci_cation Files and PDU Description Files, UDT can be easily applied to testing other UPnP device types.

6 Acknowledgements

This research was supported by the MIC(Ministry of Information and Communication), Korea, under the ITFSIP(IT Foreign Specialist Inviting Program) supervised by the IITA(Institute of Information Technology Advancement). This research was also supported by the KAI(Korea Aerospace Industries).

References:

[1] *Upnp implementers corporation*, http://www.upnp-ic.org.

[2] *Upnp device certi_cation process document version 1.0*, http://www.upnp.uic.org/docs/CERTIFICATION_OF_UPnP_DEVICES_FINAL_10_29_01.pdf.

[3] M. Jeronimo and J. Weast, *UPnP Design by Example: A Software Developer's Guide to Universal Plug and Play*. Intel Press, 2003.

[4] M. Hsueh, T. Tsai, and R. Iyer, *Fault injection techniques and tools*, Computer, vol. 30, no. 4, pp. 75.82, 1997.

[5] W. Tsai, R. Paul, W. Song, and Z. Cao, *Coyote: an xml-based framework for web services testing*, in Proc. of the 7th IEEE Int. Symposium on High Assurance Systems Engineering (HASE'02), 2002, pp. 173.174.

[6] O. Tal, S. Knight, and T. Dean, *Syntax-based vulnerability testing of frame-based network protocols*, in Proc. 2nd Annual Conf. on Privacy, Security and Trust, 2004.

[7] A. Vasan and A. Memon, *Aspire: Automated systematic protocol implementation robustness evaluation*, in Proc. of the 2004 Australian Software Engineering Conference (ASWEC'04), 2004, pp. 241.250.