SCL: A Language for Security Testing of Network Applications

Sylvain Marquis
Royal Military College of Canada
Kingston, Canada
sylvain.marquis@rmc.ca

Thomas R Dean
Queen's University
Kingston, Canada
dean@cs.queensu.ca

Scott Knight
Royal Military College of Canada
Kingston, Canada
knight-s@rmc.ca

Abstract

Security of network applications has become increasingly important in the past several years. Syntax-based testing is a black box, data driven testing technique, for applications for which input can be described formally. SCL is a component of Protocol Tester, a project at RMC and Queen's, that uses syntax-based testing to evaluate the security of network applications. As a language, SCL can describe the syntax and the semantic constraints of a given protocol, constraints that pertain to the testing of network application security. This paper describes how SCL captures the input syntax of a network application including both syntax and semantic constraints. Standard reverse engineering and program comprehension techniques are used to extract a detailed model from the description. This model can be used to automate the selection and generation of test cases in Protocol Tester.[1]

1. Introduction

The security of network applications is an increasingly important topic in both academia and industry. The cheap availability of bandwidth world wide has increased the ability of people to communicate, but has also provided convenient access to many systems for those with malicious intent. This increased access to bandwidth is not just access to the internet, but other networks such as the cellular phone networks (both voice and data). Additionally, implementations formerly restricted to executing on closed network protocols are moving to public protocols such as the move of telephone networks from packet switched networks to Voice over IP protocols.

Some recent incidents include vulnerabilities in libraries used to display images (BMP[17] and JPG[15]), a vulnerability in Cisco routers running OSPF[3,16], and a proof of concept suggesting computer viruses can propagate on cellular phones[1].

Conformance testing of these applications tends to focus on the correct implementation of the application to valid requests and obvious errors. However, in some cases, certain security vulnerabilities involve a data item that could not possibly occur in the normal operation of a protocol. For instance, at different occasions during the course of a TCP connection, state information is exchanged using the control flags located in packet headers.

One of the tactics developed by intruders in the mid-1980's was to enable every single control flags in the same packet, called the *Xmas tree* packet. The packet was sent to the target matchine, effectively sending functionally meaningless data to a TCP stack. Vulnerable stacks would then fail, compromising the machine. This became an effective Denial of Service attack.

TCP/IP and a number of other network applications exchange units of information using protocol data units (PDU), units composed of one or more fields; in turn consisting of a contiguous stream of bits or bytes. A PDU may be a single packet, or it may be spread over multiple packets. The order and arrangement of fields within PDUs corresponds to a specific syntax [2]. Syntax-based testing of network applications [10,19,21] use that syntax.

Using their own generation engines, both research of PROTOS [10] and Protocol Tester [19] gave results that demonstrated the feasibility of generating a large number of malformed PDUs based on the specification of a relatively small set of modifications commands by the human analyst.

In a previous research [18], Protocol Tester technique was used to make the technique of exploiting sophisticated constraints more efficient, the intervention of the analyst, to write new constraint-exploiting scripts needs to be reduced.

The *Structure and Context-Sensitive language* (SCL) was created to capture interesting application-level constraints on the syntax of PDUs by adding extensions to a subset of ASN.1. The goal of these extensions is to capture the application-level constraints on individual PDU fields and constraints between the value of one field and the syntax of another field, such constraints are called context-sensitive. This allows the analyst to work with SCL at a higher level of abstraction, much closer to the actual protocol specification. This paper discusses SCL, how it captures constraints and how it was validated.

2. Background

Syntax-based testing as defined by Beizer, [2] is a black box technique used to test the robustness of implementations that take structured data as input (text or binary). He identifies several mutation strategies to be applied to the input, or in the case of security vulnerability testing, to PDUs:

Syntax errors which consist of violations in the construction of more complex structures by adding, removing or altering the order of PDU fields.

Delimiter errors which alter the characters that appear as separators between fields. This includes changing the size of fixed length fields.

Value Errors which include using values that may be acceptable according to the input grammar, but are incompatible with the implementation or with other values in the data. An example is version fields which are sometimes used to identify the version of the protocol. Typically only a limited number of values are valid in the version field.

Recursion. Loops and recursions are checked with single and multiple iterations. One measure of the robustness of an application is how it deals with structures that exceed iteration and recursion limits of the application.

The PROTOS project[10,11] at Oulu University uses syntax testing to test the security of protocol implementations. They use higher order attribute grammars and a walker to generate the variant PDUs. The grammar specifies the possible PDUs down to the values of the terminal fields. The grammar is modified using a scripting language that allows the tester to modify values of terminals, add alternative values, and add alternative productions to the grammars.

The higher order attribute actions are written in Java and are triggered as the walker visits nodes in the grammar tree. Some of the actions include calculating checksums, sending and receiving packets.

While the general goal is the same, our approach is different. We capture a valid set of data by sniffing the network and transform it to generate mutant packets. The first version of Protocol Tester also uses a scripting language to drive the testing process.

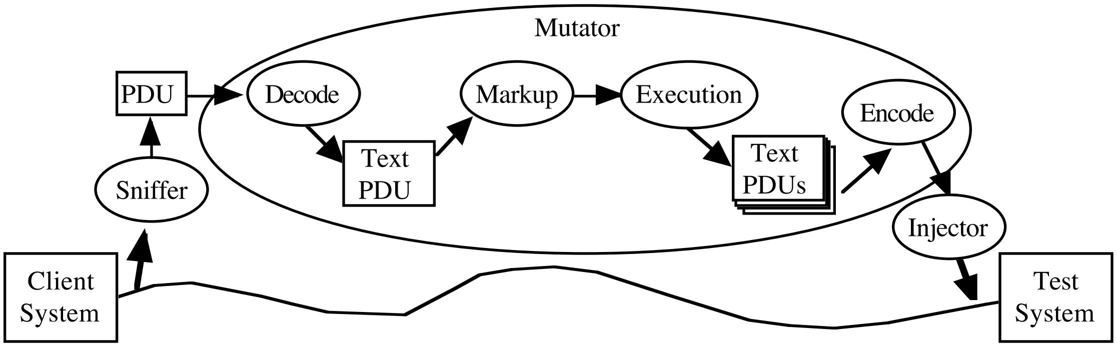

Figure 1. Protocol Tester General Structure

Figure 1 shows the overall structure of Protocol Tester [18,19,20]. At the bottom of the figure we have a network containing the test system and a client system that is interacting with the test system. A sniffer is used to capture a valid PDU that was sent from the client to the test system. The PDU at this point in time is a binary data file. This file is decoded into a textual representation by a decoder.

The markup and execution engine, implemented in TXL[4], are used to generate variants of the packet which are then re-encoded and injected into the network. The original, valid packet is injected between each of the mutated packets to verify that the test system is still functional and responsive.

The markup and execution approach is modeled on previous software evolution and transformation research [5]. This approach separates the planning of the testing suite from the execution of the testing suite. The markup that is generated is rather simple. It includes markup to delete a field, change the encoding of a field, duplicate a field, change the value of a field, and other similar tasks. The execution engine carries out most of the markup (encoding markup is carried out by the encoder).

Thus markup is always done on the original valid packet, while execution generates the modified packets. The markup phase may generate more than one marked up packet, each of which is independent. This separation of concerns is important. Testing strategies that depend on simultaneous changes to multiple fields communicate through the markup. That is, they simultaneously make markup to multiple fields. The execution engine, responsible for implementing the transforms, need not know about relationship between fields.

Protocol Tester has been used to independently verify known vulnerabilities in Trivial FTP (TFTP) and Simple Network Management (SNMP) implementations and has been used to test X.509, OSPF and BGP implementations. In both the X.509 and OSPF tests, new vulnerabilities were discovered by Protocol Tester [18,19].

One deficiency in both PROTOS and Protocol tester is that the test planning must be done manually. In both cases a scripting language is used to identify particular fields of the PDU for mutation, either by name, by pattern or by type.

```
phoneBook SEQUENCE{
   numberOfPhoneNumber INTEGER 02

   phoneNumbers SEQUENCE{
      phoneNumbers*1 INTEGER 5551212
      phoneNumbers*2 INTEGER 5551313
   }
}
```

Figure 2. Text PDU for a Phone Book application

As an example, Figure 2 shows the textual form of a PDU exchanged between clients of a fictitious phone book applications. In the figure, each field value is preceeded with a label name and a keyword giving the type. All fields are included in a composite SEQUENCE field where the order of fields represent their application syntax. The first field of phoneBook PDU contains an integer number, the second field, phoneNumbers, contains two other integer fields each representing a phone number.

While the syntax of phoneBook states the type and placement of every PDU field, other more sophisticated constraints are not shown and need to be described in prose. The value of numberOfPhoneNumber gives the cardinality, or number of elements in the phoneNumbers field. The phone book application also requires that no more than 20 phone numbers may be sent in the same phoneBook PDU. A last application constraint requires phoneNumbers to be sorted in an ascending numeric order. Consequently, 5551212 must always be placed before the other phone number, 5551313.

In the previous version of Protocol Tester, the human analyst provides a script with commands to swap the elements of the phoneNumbers field or change the values of the numberOfPhoneNumber field as well as other possible changes. The analyst is not making the changes at random. Each command in the script is targeting an application constraint. The are derived from the semantic information describe of the protocol specification. Changing the order of the fields of phoneNumbers breaks the order constraint. Changing the value of the numberOfPhoneNumber field breaks the constraint between the value field and the number of elements of the phoneNumbers fields.

This shows the need for some notation or protocol description language which captures the constraints applied to PDU field values as a consequence of the application semantics. The notation can be used to automate the test planning and generate the scripts used by protocol tester (or possibly by PROTOS) automatically.

When investigating existing protocol description languages, we discovered that almost all of them describe the syntax of the protocol, some describe the transfer syntax, and some describe the semantics of the protocol either as finite state machines[7] or as high level algorithms. We are looking for constraints such as the permissible values of a version field, the relationship between a length field and the data item governed by the length field, or that a sequence of items must be unique. In state and algorithm based protocol languages, extracting these relationships and constraints can be difficult. Furthermore, many of the protocols we are interested in are not currently described in these extended languages. Requiring an analyst to translate the prose in a standard protocol description to a finite state machine in order to extract simple constraints seems to be counter productive.

3. The New Protocol Tester

Figure 3 shows the structure of the new mutator. The basic flow of the PDU data is the same. The binary PDU is translated to text, marked up with instructions to generate multiple variant PDUs, an execution engine implements the markup and the mutant PDUs are translated back to binary.

This basic flow is extended with a protocol description expressed in SCL and a test planning engine. The protocol description is designed for a human test engineer to read and write. The information in this description is used in two ways. The first is to generate protocol syntax and transfer information for the decoder to decode the binary PDU that was retrieved from the network. It is also used by the encoder to re-encode the packet for injection.

The other way the protocol description is used is by the test planner. A design recovery extractor is run over the protocol description to generate an instance of an Entity-Relationship model that contains the information in the protocol in a form easily used by the test planner. Protocols usually describe more than one PDU type (multiple request PDU types, various response PDU types). The ER model contains the constraints for all of the PDU types, which may include constraints not relevant to the captured PDU. So the first task of the test planner is to filter the information in the

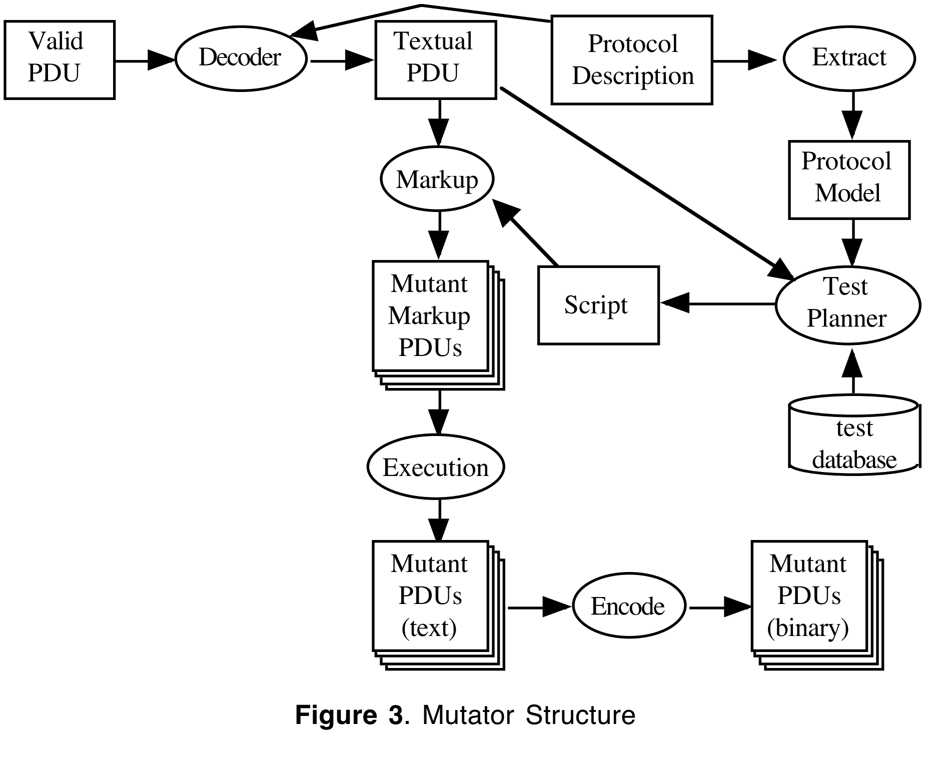

Figure 3. Mutator Structure

ER instance based on the PDU to be mutated.

The test planner [14] then invokes appropriate test plans for each of the remaining constraints. These are invoked by using the markup engine to markup the appropriate fields. The rest of this paper deals with the protocol description language, and the model that is extracted from the description.

4. SCL: Semantically Extended ASN.1

SCL extends the ASN.1 language with XML style markup. There are three XML markup blocks that may be added to an ASN.1 Grammar production in SCL. These are *size*, *transfer* and *constraints* markup. The size markup is used to identify the size of each element. The transfer element is used to identify constraints that are used to determine the binary encoding of the information. The constraints markup is used to identify other semantic constraints that must be implemented by applications.

We introduce our protocol description language with an example fictitious protocol. This protocol, the Simple Pseudo Protocol (SPP), is designed to show the various features of SCL. We first show the ASN.1 core of the protocol and then present the extensions provided by SCL and the extracted model.

4.1. SPP: an Example Protocol

Figure 4 shows the ASN.1 description of the SPP protocol. As with a traditional description of network protocols given in ASN.1, we describe the semantic constraints in textual prose.

The protocol consists of two types of PDUs, net-info PDUs which contain network configuration information, and room-loc PDUs which contain information on physical locations. Both start with the same header, which contains two fields. The first field is 2 bytes long and gives the length of the PDU, the second is also 2 bytes and identifies the type of the PDU. A value of 1 means a net-info PDU and a value of 2 means a room-info PDU.

The net-info PDUs contain lists of addresses. Each address is one of three types, an IP address, a MAC address or an error code. The numOfSubH field (1 byte) gives the number of addresses in the subheader field.

Each address consists of a single byte type field (1 = IP, 2 = MAC and 3 = Error) and a value, the data type and length of which depends on the type of address. IP addresses are 4 byte integers, MAC addresses are 6 byte octet strings and ErrorCodes are 2 byte integers.

The room-info PDU consists of only a header and a set of room info items. The number of room

```
PDU ::= (net-info | room-loc)

header ::= SEQUENCE {
  length       INTEGER
  sppType      INTEGER
}

net-info ::= SEQUENCE {
  headerNfo    header
  numOfSubH    INTEGER
  subheader    SET OF address
}

address ::= (IP-Address|
        MAC-Address | ErrCode)

IP-Address ::= SEQUENCE {
  subType      INTEGER
  ipaddress    INTEGER
}
```

```
MAC-Address ::= SEQUENCE {
  subType      INTEGER
  macaddress   OCTET STRING
}

ErrCode ::= SEQUENCE {
  subType      INTEGER
  code         INTEGER
}

room-loc ::= SEQUENCE {
  headerNfo    header
  subheader    SET OF room-info
}

room-info ::= SEQUENCE {
  floor        INTEGER
  officeNumber INTEGER
  extension    VISIBLESTRING
}
```

Figure 4. ASN.1 Description of SPP

info items is determined by the remaining bytes of the PDU. Each room info item is a constant length, consisting of a floor number (2 byte integer) and office number (2 byte integer) and a extension number (4 byte visible string).

A textual description of the typical protocol will also describe the flow of data between applications. We do not model this in SCL since Protocol Tester does not test relationships between packets at this time.

4.2 Semantic Markup in SCL

As mentioned earlier, there are three XML markup blocks that may be added to an ASN.1 grammar production in SCL: *size*, *transfer* and *constraints*.

The first of these, the size markup, is used to specify the sized of primitive fields. We use the following markup for the header definition.

```
<size>
  length is 2 bytes
  sppType is 2 bytes
</size>
```

Similar encodings are used for all sequences. The type used in the grammar specification simply states how the bytes will be interpreted. For example, the grammar portion identifies subtype as an INTEGER and macaddress as an OCTET STRING. The size markup for Mac-Address is

```
<size>
   subType is 1 bytes
   macaddress is 6 bytes
</size>
```

Thus subtype is a 1 byte INTEGER and macaddress is a 6 byte OCTET STRING.

When the size of a compound field is determined only by the size of its subfields, it is identified as SELFDEFINED. Fields whose size is determined by the value of some other field are identified as CONSTRAINED. For example the size markup for the net-info sequence is:

```
<size>
   headerNfo is SELFDEFINED
   numOfSubH is 1 bytes
   subheader is CONSTRAINED
</size>
```

The transfer block specifies constraints that are used when decoding and encoding the data. They are also used when testing. For example, a length constraint identifies situations where the length of one or more fields depends on the value of another field. While this is used when encoding and decoding a PDU, it is also a potential vulnerability in implementations [18]. The three constraints used here are: match constraints (the value of a field identifies which PDU is used), length and cardinality constraints. For example, the transfer markup for the net-info grammar production is:

```
<transfer>
  MATCHES(headerNfo.sppType==1)
  CARDINALITY(subheader) ==
     numofSubH
</transfer>
```

The final markup block is the constraints block. It covers more general constraints. Two constraints are currently implemented: value and order constraints. The value constraint is used to limit the number or range of legal values for a field. For example, the valid error codes are 1, 2 and 3. Thus, the markup for the ErrCode grammar production is:

```
<constraints>
  VALUE(code) == 1|2|3
<constraints>
```

The order constraint indicates that the values in an array field must be sorted by one or more subfields. An example is the path attribute in the Border Gateway Protocol. Path attributes have a type code located in the third byte of each attribute. Although ordering of attributes is not specified in the standard, some vendor-specific devices require that the path attributes must be ordered by the type code. In the SPP protocol, the room-info items must be sorted first by floor and second by office number. The constraint block for the room-loc production looks like:

```
<constraints>
  ORDER(subheader) = ASCENDING
    USING(subheader.floor&
    subheader.officeNumber)
<constraints>
```

Figure 5 shows the entire SPP protocol specified in SCL.

4.3 Checking and Model Extraction

Since SCL is an input to the system, it must be checked for errors. This is done using the TXL programming language[4]. We use TXL to check for the errors and to extract the protocol model from the SCL specification.

TXL has a built in parser that will report any syntax errors in the SCL program. This is augmented by TXL rules which check that all non-terminals that are used are defined exactly once, that all terminal fields have a known type and associated size constraints in the size markup. It also makes sure that all fields of a given record are uniquely named and the all of the references to fields in constraints are consistent. It also checks that only predefined functions are used in the constraints.

A separate TXL program is used to extract the protocol model from SCL. The model is represented in Rigi Standard Fomat (RSF). The model contains an unordered abstract syntax tree of the grammar, allowing the test planner to identify the parent child relationship between fields and the records that contain them.

The model also contains the text of the constraints as well as the an abstract model of the constraints. That is, the relations in the model identify which fields are involved in value, cardinality, ordered and length constraints. The test planner module can then mutate those fields based on the constraints.

4.4. Validation

We have validated the SCL language against both the OSPF and BGP protocols. Both protocols are frame based protocols with multiple types of PDUs. This validation has two components.

The first is that we have validated that the description is capable of describing the encoding of these protocols so that the decoder can translate instances of each packet type of the OSPF and BGP protocols from the binary to the textual form. This was done by decoding captured instances of each type of packet and manually verifying that the decoding was done correctly.

We have also manually checked that the model recovered from the specification includes the information necessary to generate scripts for Protocol Tester. In particular, we have verified that the information necessary to automatically generate the test scripts used to discover previously known vulnerabilities in OSPF is contained in the model. The implementation of the automated test planner is part of ongoing research [14].

5. Future Work

The system we have described is a very general infrastructure with a great deal of potential. Some of the future work we are planning on pursuing includes the following research.

The current protocols we have investigated are state independent protocols. That is, the protocols exist as request/response exchanges – send a request to a server and get a response. Each request is, in some sense, independent. Extending the framework to deal with stateful protocols such as SMB (the Windows file sharing protocol) and Voice over IP protocols is an interesting avenue to pursue. This will involve analyzing constraints between packets and generating mutated packet sequences. SCL will be extended with additional constraint primitives which can, in turn, be used by the test planner.

The protocols currently described are binary protocols. Textual protocols such as HTTP, SMTP and SOAP (XML over HTTP) can also be security tested using a transformation based process. The interesting part of these protocols is that the decoder/encoder becomes redundant, and transfer encoding is textual.

The current approach is also based on a black box approach. On some occasions when we have had source code to the test system, we have tracked down the bug in the system manually. Expanding to a white box style of testing has some potential. One option is to use the erroneous PDU to isolate the error automatically. The other option is to use a light weight program comprehension/design recovery step to identify potential security failures in the system. A full comprehension approach can be expensive both in time and resources. A light weight identification could be more aggressive in identifying potential vulnerabilities which are used to provide information to the test planner.

Alternatively, we can have the developers provide some information to the test planner. The re-

```
PDU ::= (net-info | room-loc)

header ::= SEQUENCE {
  length      INTEGER
  sppType     INTEGER
}
<size>
   length is 2 bytes
   sppType is 2 bytes
</size>

net-info ::= SEQUENCE {
  headerNfo   header
  numOfSubH   INTEGER
  subheader   SET OF address
}
<size>
   headerNfo is SELFDEFINED
   numOfSubH is 1 bytes
   subheader is CONSTRAINED
</size>
<transfer>
  MATCHES(headerNfo.sppType == 1)
  CARDINALITY(subheader) =
      numOfSubH
</transfer>

address ::= (IP-Address|
        MAC-Address | ErrCode)

IP-Address ::= SEQUENCE {
  subType     INTEGER
  ipaddress   INTEGER
}
<size>
   subType is 1 byte
   ipaddress is 4 bytes
</size>
<transfer>
   MATCHES(subType == 1)
</transfer>

MAC-Address ::= SEQUENCE {
  subType      INTEGER
  macAddress   OCTET STRING
}
<size>
  subType is 1 byte
  macAddress is 6 bytes
</size>
<transfer>
   MATCHES(subType == 2)
<transfer>
```

```
ErrCode ::= SEQUENCE {
  subType      INTEGER
  code         INTEGER
}
<size>
  subType is 1 byte
  code is 2 bytes
</size>
<transfer>
  MATCHES(subType == 3)
</transfer>
<constraints>
  VALUE(code) == 1|2|3
<constraints>

room-loc ::= SEQUENCE {
  headerNfo    header
  subheader    SET OF room-info
}
<size>
  headerNfo is SELFDEFINED
  subheader is CONSTRAINED
</size>
<transfer>
  MATCHES(headerNfo.sppType == 2)
  LENGTH(subheader) =
     PDULENGTH - SIZEOF(headerNfo)
<transfer>
<constraints>
  VALUE(headerNfo.length) ==
     PDULENGTH
  ORDER(subheader) = ASCENDING
    USING(subheader.floor &
       subheader.officeNumber)
</constraints>

room-info ::= SEQUENCE {
  floor        INTEGER
  officeNumber INTEGER
  extension    VISIBLESTRING
}
<size>
  floor is 2 bytes
  officeNumber is 2 bytes
  extension is 4 bytes
</size>
```

Figure 5. SCL Description of SPP

cent OSPF bug exposed a dependency in some versions of CISCO routers between certain requests and the value of the hello timer in the router. While the implementation of the hello timer is not part of the protocol, the existence and the relationship between the hello timer and certain PDUs is. Adding abstract implementation entities and the relationship to the protocol description can help test the implementations.

We have also started to develop an Eclipse plugin. The plugin includes an editor for the SCL language. The SCL editor shows only the ASN.1 subset of SCL, hiding the XML markup from the user. The analysis uses the editor to express constraints which are inserted whenever the file is saved.

6. Conclusions

One of the deficiencies with Protocol Tester and PROTOS is that manual effort and expertise is needed to design tests sets when performing vulnerability testing of network applications.

This paper has presented SCL, a language designed to express the syntax and context sensitive constraints of protocols. This information can be used to decode binary packets into a textual form, but, more importantly, is used by the test planning phase to automatically generate test scripts for Protocol Tester. This reduces the manual effort needed when testing implementations of new (and old) protocols. The allows the analyst to work with SCL at a higher level of abstraction, much closer to the actual protocol specification. The SCL language has been validated by expressing two existing protocols, OSPF and BGP, extracting protocol models suitable for scripting Protocol Tester from the SCL descriptions.

About The Authors

Sylvain Marquis is a masters student in the software engineering program at the Royal Milliary College of Canada. His education background is from the Royal Military College in Computer Engineering where he studied the engineering topics related to information systems, avionics and marine components enenabling the operations of Canadian Foreces equipment assets. His current research interests are security of network-enabled applications, fuzz testing and source code transformation.

Thomas Dean is an Assistant Professor in the Department of Electrical and Computer Engineering at Queen’s University and an Adjunct Associate Professor at the Royal Military College in Kingston. His background includes research in air traffic control systems, language formalization, and five and a half years as a Sr. Research Scientist at Legasys Corporation where he worked on advanced software transformation and evolution techniques in an industrial setting. His current research interests are software transformation, web site evolution and the security of network applications.

Scott Knight is an Associate Professor in the Department of Electrical and Computer Engineering at the Royal Military College of Canada; he is also cross-appointed to Queen’s University. Dr. Knight was appointed to the academic faculty at RMC in 2000 on retirement from 21 years of service in the Canadian Air Force. He has worked with the National Defence Intelligence and Security communities on the development of secure computing networks to be used for handling classified and national security related information. He has also founded the Computer Security Laboratory at RMC, and continues to lead this research group. this research group has a close working relationship with the Canadian Forces Information Operations Group and focuses on computer network defence and support to information operations

References

[1] BBC, *‘Game virus’ bits mobile phones*, *BBC news*, UK edition, Aug. 11, 2004.

[2] Beizer, B., *Software Testing Techniques*, 2nd Edition, Van Nostraad Reinhold, New York, 1990.

[3] Cisco, *Cisco Security Advisory: Cisco IOS Malformed OSPF Packet Causes Reload*, Document ID: 61365, Cisco Systems, San Jose, California, Aug. 2004.

[4] J.R. Cordy, “TXL - A Language for Programming Language Tools and Applications”, Proc. LDTA 2004, *ACM 4th International Workshop on Language Descriptions, Tools and Applications*, Edinburg, Scotland, January 2005, pp. 3-31.

[5] Dean, T.R., Cordy, J.R., Schneider, K.A.,

Malton, A.J., "Using Design Recovery Techniques to Transform Legacy Systems", *ICSM 2001 - The International Conference on Software Maintenance*, Florence, Italy, November 2001, pp 622 - 631.

[6] Dubuisson, O., "ASN.1 Communication between Heterogeneous Systems", Academic Press, San Diego, 2001.

[7] M.G. Gouda, *Elements of Protocol Design,* Wiley, New York, 1998.

[8] International Standard 8824 - INTERNATIONAL TELECOMMUNICATION UNION X.208, “Information technology -- Open Systems Interconnection -- Specification of Abstract Syntax Notation One (ASN.1)”, 1988.

[9] International Standard 8825-1 - INTERNATIONAL TELECOMMUNICATION UNION X.690, “Information Technology - ASN.1 Encoding Rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER)”, http:// www.itu.int /ITUT/studygroups/com17/languages/X690_ 0702.pdf, 2002.

[10] R. Kaksonen, *A Functional Method for Assessing Protocol Implementation Security,* Licentiate Thesis. Espoo. Technical Research Centre of Finland, VTT Publications 447. ISBN 951-38-5873-1

[11] R. Kaksonen, M. Laakso, A. Takanen, "Software Security Assessment through Specification Mutations and Fault Injection". *Proc. of Communications and Multimedia Security Issues of the New Century / IFIP TC6/TC11 Fifth Joint Working Conference on Communications and Multimedia Security (CMS'01)*, Darmstadt, Germany, May 2001, ISDN 0-7923-7365-0.

[12] Lougheed, K. Rekhter, Y. “A Border Gateway Protocol 4” (BGP-4), IETF 1995, ftp://ftp.rfc-editor.org/in-notes/rfc1771.txt

[13] S. Marquis, *A Protocol Messages Specification Methodology for Security Vulnerability Testing*, M.Sc. Thesis, Dept of Electrical and Computer Engineering, Royal Military College of Canada, 2005.

[14] W. McInnnis, *Developing Refined Strategies for Vulnerability Testing on Network Protocols Using Syntax Testing,* M.Sc. Thesis, Dept of Electrical and Computer Engineering, Royal Military college of Canada, in progress.

[15] Microsoft, *Buffer Overrun in JPEG Processing (GDI+) Could Allow Code Execution*, Microsoft Security Bulletin MS04-28, Sept. 2004.

[16] Moy, J., OSPF Version 2, Internet RFC 2328, 1998..

[17] SecurityTracker.com, *Microsoft Internet Explorer Integer Overflow in Processing Bitmap Files Lets Remote Users Execute Arbitrary code*, Security Tracker ID: 1009067, Feb 2004.

[18] Tal, O., Knight, S., Dean., T., Syntax-based Vulnerability Testing of Frame-based Network Protocols, *Proceedings of the Second Annual Conference on Privacy, Security and Trust*, Fredericton, Canada, October 2004, 6 pp., to appear.

[19] Turcotte, Y., *Syntax Testing of the Entrust Public Key Infrastructure for security vulnerabilities in the X.509 Certificate*, M.Sc. Thesis, Department of Electrical and Computer Engineering, Royal Military College of Canada, 2003.

[20] Turcotte, Y., Oded, T., Knight, G.S., Dean, T., “Security Vulnerabilities Assessment Of the X.509 Protocol By Syntax–Based Testing”, *Proceedings of MILCOM 04*, Monterey, California, October 2004, 7 pages, to appear.

[21] S. Xiao, L. Deng, S. Li, X. Wang, “Integrated TCP/IP Protocol Software Testing for Vulnerability Detection:, *Proc. 2003 International Conference on Computer Networks and Mobile Computing*, Shanghi china, Oct. 2003.