Applying Software Transformation Techniques to Security Testing

Thomas Dean
Electrical and Computer Engineering
Queen's University
thomas.dean@ece.queensu.ca

Scott Knight
Electrical and Computer Engineering
Royal Military College of Canada
knight-s@rmc.ca

Abstract

Application protocols have become sophisticated enough that they have become languages in their own right. At the best of times, these protocols are difficult to implement correctly. Combining the complexity of these protocols with other development pressures such as time to market, limited processor power and/or demanding performance requirements make it even more difficult to produce implementations without security vulnerabilities. Traditional conformance testing of these implementations does not reveal many security vulnerabilities. In this paper we describe ongoing research where software transformation and program comprehension techniques are used to to assist in the security testing of network applications.

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 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[13] and JPG[9]), a vulnerability in Cisco routers running OSPF[4], and a proof of concept of the first virus for 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, sometimes the security vulnerabilities involve a data item that could not possibly occur in the normal operation of a protocol. As an example, an Xmas tree packet is a low level IP packet that has every single flag in the header enabled. Some of these flags are mutually exclusive. In the mid 1980's, several implementations of TCP/IP operating systems were unable to handle these packets, and this became an effective Denial of Service (DOS) attack.

Our position is that evolution transformation techniques can be fruitfully applied to structured data such as network protocols. The protocols used by network applications have become languages in their own right with both syntax and semantics. One approach to security testing is Syntax Testing [2]. In this approach, syntax and semantic errors are intentionally made to produce variants of the data to attempt to expose vulnerabilities.

The PROTOS project[8] at Oulu University uses a protocol grammar to generate variant packets. The grammar specifies the possible packets right down to the values of fields. The grammar is modified manually to allow the desired errors and then a walker walks the grammar tree, automatically generating the packets.

While the general technique is the same, our approach is different. We capture a valid set of data by sniffing the network and transforming it to generate alternate packets. We also are automatically generating the test plans based on the syntax and semantics of the protocol without manual intervention. This represents a novel cross-fertilization between the software transformation and the security communities.

2. System Structure

Figure 1 shows the overall structure of our system. 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 protocol data unit (PDU) that was sent from the client to the test system. A PDU may be a single packet, or it may be spread over multiple packets. 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[5], 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 [6]. 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,

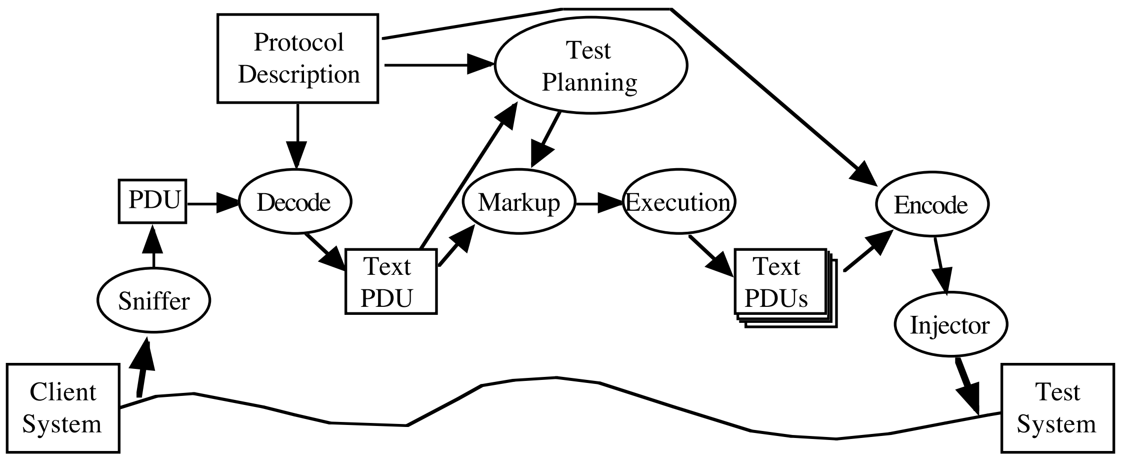

Figure 1. Protocol Tester General Structure

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, may generate more than one 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 make markup to simultaneous fields. The execution engine, responsible for implementing the transforms, need not know about relationships between fields.

The description of the protocol contains a variety of information. It contains the syntax of the protocol, transfer encoding information and semantic information such as constraints between fields, ordering of sequences and if the sequence must be unique. Most protocols are described in a document which contains the syntax of the protocol in a standard form such as EBNF or ASN.1[7]. The semantic constraints of the protocol tend to be described in the prose of the document. Some means of describing these constraints in addition to the syntax is needed. We are interested in identifying the constraints that exist both between fields of a given PDU (current work) and between PDUs in a sequence (future work).

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 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 a test engineer to translate the prose in a standard protocol description to a finite state machine in order to extract simple constraints seemed to be counter productive.

Figure 2 shows a description of our simple house description protocol in our protocol description language as a frame based protocol. Our protocol description language is an XML based extension to the existing ASN.1 standard. An XML markup is added to each non-terminal in the description and provides information about transfer encoding and constraints. In the Fig. 2, the transfer encoding for the first non-terminal (*HDP_Packet*) indicates that the *number_of_houses* field is encoded as a 4 byte integer. The constraint markup for the first non-terminal also indicates that the value of the first field (*number_of_houses*) gives the cardinality of the second field (*houses*). Other constraints include value constraints (e.g. version is 0, 1 or 2, range is 0 to 255), length constraints (cardinality is number of items, length is number of bytes). The third non-terminal(*House*) has a single markup indicating the transfer encoding for the *house_number* and *family_name* fields.

This description is designed for a human test engineer to read and write. Currently we are working on an Eclipse plugin which will allow the test engineer to author the ASN.1 directly and to enter the constraints interactively, never having to deal with the XML directly.

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

```
HDP_Packet ::= SEQUENCE {
      number_of_houses  INTEGER
      houses            Houses
}
<size>
      number_of_houses is 4 bytes
</size>
<constraints>
Cardinality(houses):=number_of_houses
<constraints>
```

```
Houses ::= SEQUENCE OF House
House :: = SEQUENCE {
      house_number   INTEGER
      family_name    VisibleString
}
<size>
      house_number is 2 bytes
      family_name is 100 bytes
</size>
```

Figure 2. Protocol description of the House Description Protocol

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 ER model that contains the information in the protocol in an form easily used by the test planner. Protocols usually describe more than one PDU types (multiple request PDU types, various response PDU types). The ER instance contains the constraints for all of the PDU types, which includes constraints not relevant to the captured PDU. So the first task of the test planner is to filter the information in the ER instance based on the PDU to be mutated. The test planner 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 approach currently in progress is table driven, with the first field in the table identifying the constraint type identifying one or more strategies which leads to a template expression that marks up the fields involved in the constraint. For example, the cardinality constraint given in Figure 2 leads to several mutant PDUs where the value of the *number_of_houses* field disagrees with the number of House entities in the *houses* field.

3. Current Status

The base system for DER based protocols (e.g. X.509[11,12] and SNMP[3]) was completed as part of Yves Turcotte's M.Sc. Thesis[15]. This comprises the sniffer, the decoder, markup and execution engines, the encoder and the injector. A scripting tool that drove the markup and encoding engines was built that allowed a user to indicate what errors should be applied to which fields. A large variety of error strategies were designed and implemented as part of this work. The work was used to independently confirm errors in SNMP implementations and to test an implementation of X.509 that was adopted by the Canadian Department of National Defense. A new potential denial of service attack was discovered in the implementation of the X.509 protocol which increased the processing time of a certificate from a fraction of a second to over two hours of CPU time. As ASN.1/DER based protocols are self describing, this system is completely protocol independent. That is, the system has no knowledge of the protocol syntax or semantics.

This infrastructure was extended by Dr. Oded Tal [14] to handle frame based protocols (e.g. OSPF). This involved adding a simple description of the protocol that was used by the decoder and encoder to translate between the binary and textual forms of the PDUs. But it also involved investigation of the types of errors that apply to frame based protocols. For example, some of the syntax based mutations appropriate to DER based protocols do not apply to frame based protocols. Deleting a field from the middle packet simply shortens the packet and the test system will interpret the following field as the contents of the deleted field. Thus, deleting a field is the same as generating random values for subsequent fields. Similarly encoding errors are limited. The possible DER vulnerability of valid over length integer values is not reasonable in a protocol which mandates a single length to fields.

However mutations based on values of fields and on relationships between values of fields are effective. For example, in OSPF there is a field within the packet describing the length of some data fields in the packet. In some implementations, this field is used, even if it says that the data in the packet is longer than the length of the packet itself. This leads to segmentation faults on Unix systems and system failures in Windows 2000 Server. To the best of our knowledge, this is a new vulnerability.

The protocol description language and the test planner are currently in the process of being implemented and integrated into the system.

4. 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 include:

State based protocols. 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 get a response. Each request is, in some sense, independent. Extending the framework to deal with stateful protocols such as Voice over IP protocols is an interesting avenue to pursue. This will involve analyzing constraints between packets and generating mutated packet sequences.

The protocols described are also currently 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 recent 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. So adding abstract implementation entities and the relationship to the protocol description can help test the implementations.

5. Conclusions

This paper has presented some of the ongoing research into network security at the Royal Military College of Canada and Queen's University. This research is the direct result of combining current research approaches from two very diverse communities: the software evolution and transformation community and the software security community.

References

[1] BBC, *'Game virus' bits mobile phone*s, BBC news, UK edition, Aug. 11, 2004.

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

[3] Case, J., Fedor, M., Schoffstall, M., Davin, J., Simple Network Management Protocol, Internet RFC 1157, 1990.

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

[5] Cordy, J., The TXL Programming Language, v. 10, http://www.txl.ca/, 2000.

[6] 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.

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

[8] Kaksonen, R., Laasko, M. and Takanen, A., *Vulnerability Analysis of Software through Syntax Testing*,http://www.ee.oulu.fi/research/ouspg/protos/analysis/WP2000-robustness/index.html, 2001.

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

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

[11] PKCS#7-Cryptographic Message Syntax Standard. http://www.rsasecurity.com/rsalabs/pkcs/pkcs-7/, RSA Data Security, Inc, 2004.

[12] Public-Key Infrastructure (X.509) (pkix). http://www.ietf.org/html.charters/pkix-charter.html, 2004.

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

[14] 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.

[15] 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.

[16] 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.