Generic Data Format Approach for Generation of Security Test Data

Christian Schanes, Florian Fankhauser, Stefan Taber, Thomas Grechenig
Vienna University of Technology
Industrial Software (INSO)
1040 Vienna, Austria
E-mail: christian.schanes,florian.fankhauser,
stefan.taber,thomas.grechenig@inso.tuwien.ac.at

***Abstract*—Security testing is an important and at the same time also expensive task for developing robust and secure systems. Test automation can reduce costs of security tests and increase test coverage and, therefore, increase the number of detected security issues during development. A common data format as the basis for specific test cases ensures that the implementation of the generation logic for security test data is only needed once and can be used for various data formats by transforming the data to the common data format, generating the test data and transforming back to the original data format. The introduced approach enables to generate test data for various formats using a single implementation of the generation algorithm and applying the results for specific test cases in different data formats.**

***Keywords*—Software testing; Computer network security; Fuzzing.**

1. Introduction

Software systems are getting more and more complex today using various programming languages and protocols developed by many different companies. Security testing of such systems is required to increase robustness against attacks. The extent of the attack surface of systems with different interfaces requires multiple different tools for security testing, which also increases required resources for test execution. The generation of the required test data and the execution of security tests are time consuming and, therefore, expensive. Automatization of test execution is required to reduce costs and increasing the test coverage of a System Under Test (SUT).

Applications for conducting security tests require test data as input, which can be mutated using methods to alter the data to structures and values that are specific for security tests. For generating such information a possible available input format definition can be used, e.g., XML Schema Definition (XSD). Our approach considers the automated extraction of such a format definition based on a set of test data if such a definition is not available or not accurate enough for generating proper test data. Such test data can be gained during development or by using input data of functional tests.

This work presents an approach using Extensible Markup Language (XML) as generic data format for test execution. Using one generic format allows the implementation of one smart generation algorithm instead of using specific algorithms for various data formats and allows to use a reduced tool set for test execution. For additional protocols only new transformation rules are required and not the new implementation of a completely new generation algorithm.

For testing a single service multiple tools are required to test the various layers of the service. For example, a web service uses Hyper Text Transfer Protocol (HTTP) as transport protocol, XML or JavaScript Object Notation (JSON) for data transport and in security critical environments cryptographic methods are used to ensure protection goals, which often require X.509 certificates. Our approach allows to use one tool to generate test data for all of the used protocols in a web service. As description language we use XSD [1] introduced by W3C as a standard for describing how a certain XML document is supposed to look like. As a prototype we implemented transformation routines for Abstract Syntax Notation One (ASN.1), which is used for many Public Key Infrastructure (PKI) protocols of the X.509 standard like revocation lists or certificates [2].

The remainder of this paper is structured as follows: Section 2 lists related work. Section 3 discusses XML as a common format for security tests. Section 4 gives details about the introduced approach for generating test data. The implemented prototype was used for generating test data for PKI protocols, which is presented in Section 5. The paper finishes with a conclusion and ideas for further work in Section 6.

2. Related Work

Thompson [3] defines security failures as side effects of the software, which are not specified and make security testing hard. Fuzzing is one possible technique to find such side effects by executing the application with many automatically randomly or rule-based generated input data [4]–[6].

If no detailed specification of the interfaces are available to generate data automatically it is possible to extract test data from executed functional tests, e.g., by using stored test data, network traffic [7], [8] or by dynamic binary analysis [9]–[11].

Transforming from different data formats to XML and vice versa is discussed by various authors, e.g., for Resource Description Framework (RDF) [12], relational databases [13], [14] or ASN.1 [15]. ASN.1 [16] is a format commonly used by different protocols and applications. Yoon et al. [17] discuss the usage of ASN.1 for Simple Network Management Protocol (SNMP). The ASN.1 format is also used for various PKI protocols like X.509 [2].

Moreover, different authors discussed the similarity of ASN.1 and XML [18]–[22] and mapping of ASN.1 and XML [21].

Due to the widespread usage of XML processing systems the generation of test data was discussed by various authors. Different approaches are available, which are based on different sources for generation like document specification languages (e.g., XSD or Document Type Definition (DTD)), based on specific generation rules in a non XML format or by using example input data. Aboulnaga et al. [23] discuss a generator, which is based on simple implemented rules, which allow limited generation of XML data. Bertolino et al. [24]–[26] implemented TAXI, a XML generator, which generates documents (instances) based on a given XML schema and follows the XML-based Partition Testing approach (XPT), which is a modification of the Category Partition Method. Xu et al. [27] also use a XML schema as basis to generate valid and invalid messages for testing web services. ToXgene by Barbosa et al. [28] is a template based XML generator, which uses a template specification language within a XML schema to describe the rules for generating data. Pan et al. [29] use example instances to extract allowed values for the generated data additionally to the document specification.

3. XML as Common Format for Test Data Generation

Implementing techniques for optimized data generation for tests is often required for test execution to detect as many errors as possible with as less effort as possible. For this our approach considers the usage of a common data format where optimized algorithms for data generation have to be implemented only once.

3.1. Definition of Security Test Data

The attack surface of a SUT contains various interfaces using different protocols and data formats on various layers, e.g., network or application. Analyzing the attack surface of an application is important for conducting security tests. The application takes the input data and, often, at first performs syntactical validation of the data. For state based protocols another validation is done by the state machine, which only allows specific state transitions. Finally, the data will be handed over to the business logic, which processes the values. Understanding and considering such validation steps is important to test the intended piece of software within the application, which is only seen as black box.

For preparing input data for security tests, semantical, syntactical and state aspects have to be considered. Semantical aspects consider a standard compliant processing of the information. The generated values have to fulfill certain restrictions, e.g., a valid checksum. The definition can be stated in the technical interface specification, e.g., restrictions in XSD, or restrictions in the applications without explicit definition in the interface specification. Both aspects have to be considered. For testing semantical aspects structural and state valid data is required because otherwise the data will be discarded during the validation process of the SUT and the business logic, which semantically processes the data will not be triggered. State aspects consider state based protocols where the underlying state machine expects data in a specific order and otherwise discards it. The structure is considered by validating the syntax. It is the most basic aspect of the SUT interface because a structural invalid message will be rejected early before handling the information to the state machine or the business logic. It is required to generate structural valid documents according to the interface specification and structural invalid documents to test the robustness of the system with such data. For increased coverage of the SUT all of the mentioned aspects have to be considered when creating test data for security tests.

Common practice for tests is to inject only one fault per test case to ensure reproducibility [30]. This applies for semantical, syntactical as well as for state aspects. Therefore, valid data is required for security tests to be able to inject faults on specific positions only.

3.2. Data Generation Approach Based on XML

Figure 1 shows the test data generation process used in our approach. There are several alternative paths depending on the available source information for producing test data. One path is based on existing data definition languages, which are used directly if they are already available as XSD. If they are available in a different format they can be transformed to the used XSD format. Another path is the usage of available input data, which will first be transformed to XML and based on XML input data a data definition will be derived as XSD. The generation is always based on the given or generated XSD and an optional set of rules or example data to produce accurate values for the test data.

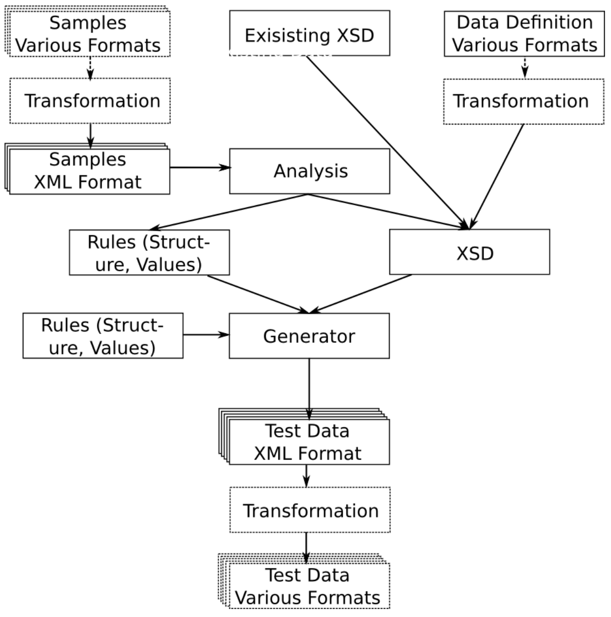

Figure 1. Process of Test Data Generation Approach Based on a Common Data Format

3.3. Using Transformation to Generate Data for Various Formats

Transformation is one of the key aspects of our approach. By transforming from various data formats to XML/XSD a generic data generation approach is possible. For many different data formats there already exist methods for transforming to and from XML [12]–[14], [20], [21], [31].

The approach also allows transformation of data definition languages, e.g., DTD. This allows the generation of test data if a data definition is available, which can be transformed to XSD. After the generation process all the produced XML test data will be transformed to the destination data format.

3.4. Extraction of Interface Definition Based on Samples

Additionally, the generation is also possible without a document specification as can be seen in Figure 1, e.g., based on a set of input data extracted from the network.

A way to find or rather deduce some kind of schema is to use a set of XML input data samples. Only valid XML instances can be used as samples. Such data can be identified, for example, by executing the application and logging the generated data.

The approach uses the given samples to deduce rules which the XML (its structure and content) follow and build a specification as XML schema. The resulting XML schema can then be used for the test data generation.

The samples usually can not cover every possible data variation that is actually allowed and some assumptions have to be made when deducing rules for a common input specification. This means that some information about the actual input specification gets lost and, therefore, the data that can be generated will be restricted.

4. Automated Generation of Test Data

The presented approach for generating test data uses XSD as source, which builds the model to generate the test data. Based on the given XSD, structural valid and invalid XML documents with valid and invalid values can be generated. XML is hierarchically structured, which allows to systematically traverse through the hierarchical tree, starting at the root-element. Each node (e.g., sequences, elements, attributes, . . .) of the tree will be processed and based on schema details, e.g., `min-`/`maxOccurs`, choices, various possible facets, etc. the test data will be generated.

The generated test data will be used for security testing by using a fuzzing approach. Two different aspects are considered during test data generation to allow usage by the fuzzing engine. Firstly, valid test data, which can be further used to inject faults are generated. This requires valid structures with valid values to avoid discarding by validation routines in the SUT. Secondly, a set of test data is required that is using invalid structures, which do not fulfill the available data definition.

4.1. Approach for Generation of Valid XML Structures

In XML schema it is possible to define customized data types in form of simple and complex types. These types can then be used like built-in data types to specify the content of elements or attributes. Simple types are used to constrain existing data types. They do not allow the definition of attributes or child elements. Complex types are used for defining a structure of elements with possible further child elements, which can again be simple or complex types. There are three indicators for ordering or choosing these elements, which are `sequence`, `all` and `choice`. The `sequence` indicator specifies that the elements have to be in a specific order, which has only one valid variant for ordering the elements, because every other sequence of the given elements makes the document schema invalid.

The order of the defined elements is irrelevant if the `all` indicator is used. Every permutation of those elements within the `all` indicator would represent a valid variant of the complex type.

The `choice` indicator differs from the others in the way that from the child elements only one element can be chosen. Therefore, there is one valid data variation for each element defined in the `choice`.

XSD further supports to define elements and attributes `optional` or `required`. If an element is marked as optional, then there exist at least two valid variants for this element. One with the element and one without it. In case of elements, it is also possible to define the occurrence of such an element by using the facets (restrictions or "rules" defined in XSD) `minOccurs` and/or `maxOccurs`, which also has to be considered for the generation of data. If such restrictions exist, a variant with the minimum occurrence, one with the maximum occurrence and one with an occurrence between those two limits are chosen. If the restriction of `maxOccurs` is "unbounded" a predefined maximum value has to be defined for the element.

The result of the test data generation in form of the generated XML instances in the end is basically just the combination of the available valid variants of each element to form a number of different XML documents. These XML documents will all be valid according to the given XML schema, because they were produced by applying every rule defined in the schema and using them to find meaningful variants based on the existing restrictions.

4.2. Approach for Generation of Schema Invalid Variants

In addition to the generation of valid XML documents, in this section different approaches for manipulating the structure of an XML document are explained, so that it is not valid according to its XML schema. Such instances are required for testing SUT behavior during processing schema invalid data. For the presented approach only invalid variants are possible, which are describable in XSD and no manipulated instances are generated, e.g., non XML well formed documents, because such documents can not be transformed back to the required data format, e.g., ASN.1.

For generating invalid variants only complex types and indicators are considered. Generating invalid simple types are not required in the used approach because this is done by the fuzzing approach.

The order of the elements within a `sequence` is crucial, which means that the elements can be swapped to produce an invalid XML instance. In the case of an `all` or a `choice` the order of the elements is irrelevant so it is not possible to produce invalid instances when shuffling the order. Invalid instances for all indicators are possible by inserting not allowed elements.

The `maxOccurs` and `minOccurs` attributes are used for producing an invalid XML instance with a wrong number of elements. For that, equivalence partitioning is applied. Moreover, specific security critical values are used as, for example, byte ranges. The same approach applies to `maxOccurs` and `minOccurs` within an indicator as well. For example, if a choice is used in which `minOccurs` and `maxOccurs` are 1, no item or several items are selected from the choice so that the XML instance is invalid.

Attributes can be optional or mandatory by using the attribute `fixed` in XSD. For mandatory attributes an invalid instance is given by skipping the attribute.

4.3. Generation of Proper Values

The generation of proper values is important to get valid XML documents for test execution. The documents will further be used as input in a fuzzing engine, which uses fault injection to prepare the final security test data.

For the generation the definitions of the XSD are used, which provide information about data types and specific constraints. Additionally, it is possible to provide a set of samples to extract proper values from.

The XML schema defines built-in primitive and derived types. For many types it is possible to define further constraining facets for their value space in the XML schema definition. For example, `string` allows the facets listed in Figure 2. During generation special aspects (e.g., format, value space and facet/restrictions) of all types have to be considered.

Facet	Description
length	For a fixed length string
minLength	Minimum length of the expected string
maxLength	Maximum length of the expected string
pattern	Regular expression restriction
enumeration	Only allow a set of possible values
whiteSpace	Definition of handling white spaces

Figure 2. Relevant Facets for a String Type in XSD

If samples are given for the generation process as shown in Figure 1 then the values from the samples are extracted. The example values can be further used for building valid test data. For this process the XML samples are parsed and each value is stored in a list together with the position of the value as XPath statement. Additionally, it is possible to use predefined values from a configuration file again as a tuple of the position as XPath statement and the value. This approach allows manual overriding of the automatic value generation.

For specific data it is possible to use a predefined generator implementation, which will build valid data, e.g., timestamps, unique IDs, . . . Additionally, the used fuzzing engine supports the extraction of values of the communication protocol, which is required for further input data, e.g., session ID in the response from the server.

5. Prototype Implementation: Generating Data for PKI

We implemented the presented approach as a prototype and used it for generating data for ASN.1 based PKI protocols. Our approach uses the BouncyCastle library to read the ASN.1 structure. We implemented the transformation between ASN.1 and XML and applied the approach for generating data for Online Certificate Status Protocol (OCSP) messages and X.509 certificates.

5.1. ASN.1 and XML

ASN.1 supports a number of different encoding rules, e.g., Basic Encoding Rules (BER) or Distinguished Encoding Rules (DER). The considered encodings are Type Length Value (TLV) based structures where type is a unique identifier for the type and basically resembles the element name in XML documents (e.g., `<TBSCertificate>`). Instead of using closing elements as used by XML ASN.1 defines the length of the value. The value can be a simple or complex type, which is again another TLV structure, which is similar to those of XML documents.

5.2. Fault Injection for Generating X.509 Certificates

X.509 certificates are used for many scenarios for securing infrastructures, e.g., Virtual Private Network (VPN) or Secure Socket Layer (SSL). Thorough testing of such security gateway implementations is important to ensure secure and robust implemented security functionality. We generated X.509 test certificates for security testing infrastructures. Therefore, we used samples available from functional tests to generate security test data. Figure 3 shows the process of generating certificates using a fuzzing approach for fault injection.

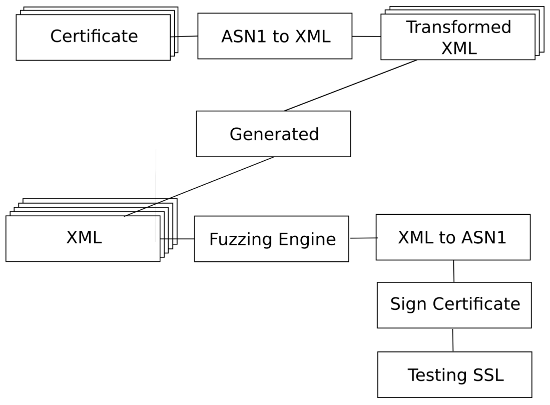

Figure 3. Process of Certificate Fuzzing Using XML

For testing structural robustness problems schema invalid variants were generated as discussed in Section 4.2. Faults for value fields for simple data will be injected by the used fuzzing engine based on random generated data and predefined attack vectors. This is done for each field in the X.509 certificate. The used transformation implementation additionally allows fuzzing of the ASN.1 length and type fields to ensure robustness of internal ASN.1 parsing functionality by manipulating the values so that the defined type and length are not fitting the content of the field. The algorithm for generation of the data considers the certificate content, e.g., key, subject, and the signature part, e.g., signature value, signature algorithm.

Finally, for X.509 certificates a valid signature is required. For this a valid signature will be attached to the certificate after transforming back to ASN.1. This allows the usage of the certificate for the tested use case, e.g., in Figure 3 the certificate will be used to establish a SSL connection.

6. Conclusion and Further Work

We presented an approach for using XML as a common format for the generation of security test data, which allows to test semantical, syntactical and state aspects. For the generation of security test data various data formats can be transformed to and from XML, which allows testing different protocols by implementing the generation algorithm for the security test data only once. This reduces costs and allows the introduction of a framework for the generation of test data. For new formats only the transformation routines have to be implemented.

The approach was applied for security testing ASN.1 based PKI protocols. As samples for the generation algorithm X.509 certificates of the functional tests were used. The process started by transforming ASN.1 based X.509 certificates to XML, generating test data and transforming back to ASN.1 for testing a SSL implementation.

For future work the approach should be extended by automatically detecting semantical constraints, which are not defined in the XSD to produce semantical valid test data. Currently, the performance of the implemented prototype leads to long running generation tasks. A more efficient implementation is required to increase the number of data variations for the test execution.

References

[1] W3C, "Xml schema," http://www.w3.org/TR/xmlschema-0/, [accessed: 2010-10-20].

[2] Y. Turcotte, O. Tal, S. Knight, and T. Dean, "Security vulnerabilities assessment of the x.509 protocol by syntax-based testing," vol. 3, Oct./Nov. 2004, pp. 1572–1578.

[3] H. H. Thompson, "Why security testing is hard," *IEEE Security & Privacy Magazine*, vol. 1, no. 4, pp. 83–86, 2003.

[4] B. P. Miller, L. Fredriksen, and B. So, "An empirical study of the reliability of unix utilities," *Commun. ACM*, vol. 33, no. 12, pp. 32–44, 1990.

[5] J. E. Forrester and B. P. Miller, "An empirical study of the robustness of windows nt applications using random testing," in *WSS'00: Proceedings of the 4th conference on USENIX Windows Systems Symposium*. Berkeley, CA, USA: USENIX Association, 2000, pp. 6–6.

[6] P. Godefroid, A. Kiezun, and M. Y. Levin, "Grammar-based whitebox fuzzing," in *PLDI '08: Proceedings of the 2008 ACM SIGPLAN conference on Programming language design and implementation*. New York, NY, USA: ACM, 2008, pp. 206–215.

[7] O. Udrea, C. Lumezanu, and J. Foster, "Rule-based static analysis of network protocol implementations," *Inf. Comput.*, vol. 206, no. 2-4, pp. 130–157, 2008.

[8] W. Cui, J. Kannan, and W. Helen, "Discoverer: automatic protocol reverse engineering from network traces," in *SS'07: Proceedings of 16th USENIX Security Symposium on USENIX Security Symposium*. Berkeley, CA, USA: USENIX Association, 2007, pp. 1–14.

[9] J. Caballero, H. Yin, Z. Liang, and D. Song, "Polyglot: automatic extraction of protocol message format using dynamic binary analysis," in *CCS '07: Proceedings of the 14th ACM conference on Computer and communications security*. New York, NY, USA: ACM, 2007, pp. 317–329.

[10] W. Cui, M. Peinado, K. Chen, H. Wang, and L. Briz, "Tupni: automatic reverse engineering of input formats," in *CCS '08: Proceedings of the 15th ACM conference on Computer and communications security*. New York, NY, USA: ACM, 2008, pp. 391–402.

[11] Z. Lin and X. Zhang, "Deriving input syntactic structure from execution," in *SIGSOFT '08/FSE-16: Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering*. New York, NY, USA: ACM, 2008, pp. 83–93.

[12] D. Van Deursen, C. Poppe, G. Martens, E. Mannens, and R. Walle, "Xml to rdf conversion: A generic approach," nov. 2008, pp. 138 –144.

[13] J. Fong, F. Pang, and C. Bloor, "Converting relational database into xml document," 2001, pp. 61 –65.

[14] M. Jacinto, G. Librelotto, J. Ramalho, and P. Henriques, "Bidirectional conversion between xml documents and relational databases," 2002, pp. 437 – 443.

[15] ITU-T, "X.693 information technology asn.1 encoding rules: Xml encoding rules (xer)," X SERIES: DATA NETWORKS, OPEN SYSTEM COMMUNICATIONS AND SECURITY OSI networking and system aspects - Abstract Syntax Notation One (ASN.1), Nov. 2008, identical standard: ISO/IEC 8825-4:2008 (Common).

[16] ——, "X.680, Abstract syntax notation one (ASN. 1): Specification of basic notation," 1994.

[17] J. Yoon, H.-T. Ju, and J. Hong, "Development of snmp-xml translator and gateway for xml-based integrated network management," *Int. J. Netw. Manag.*, vol. 13, no. 4, pp. 259–276, 2003.

[18] D. Mundy and D. Chadwick, "An xml alternative for performance and security: Asn.1," *IT Professional*, vol. 6, no. 1, pp. 30–36, 2004.

[19] D. Mundy, D. Chadwick, and A. Smith, "Comparing the performance of abstract syntax notation one (asn.1) vs extensible markup language (xml)," in *In Proceedings of the Terena Networking Conference*, 2003.

[20] T. Imamura and H. Maruyama, "Mapping between asn.1 and xml," *Applications and the Internet, IEEE/IPSJ International Symposium on*, vol. 0, 2001.

[21] A. Triglia, "The asn.1 language as a new schema definition language for xml," XML Europe 2002 Conference, May 2002.

[22] ITU-T, "X.690: ITU-T Recommendation X.690 (1997) Information technology-ASN.1 encoding rules: Specification of Basic Encoding Rules (BER)," 1997.

[23] A. Aboulnaga, J. F. Naughton, and C. Zhang, "Generating synthetic complex-structured xml data," in *In Proc. 4th Int. Workshop on the Web and Databases (WebDB2001*, 2001.

[24] A. Bertolino, J. Gao, E. Marchetti, and A. Polini, "Taxi– a tool for xml-based testing," in *ICSE COMPANION '07: Companion to the proceedings of the 29th International Conference on Software Engineering*. Washington, DC, USA: IEEE Computer Society, 2007, pp. 53–54.

[25] ——, "Systematic generation of xml instances to test complex software applications," in *RISE'06: Proceedings of the 3rd international conference on Rapid integration of software engineering techniques*. Berlin, Heidelberg: Springer-Verlag, 2007, pp. 114–129.

[26] ——, "Automatic test data generation for xml schema-based partition testing," in *Automation of Software Test , 2007. AST '07. Second International Workshop on*, may. 2007, p. 4.

[27] W. Xu, J. Offutt, and J. Luo, "Testing web services by xml perturbation," in *ISSRE '05: Proceedings of the 16th IEEE International Symposium on Software Reliability Engineering*. Washington, DC, USA: IEEE Computer Society, 2005, pp. 257–266.

[28] D. Barbosa, A. Mendelzon, J. Keenleyside, and K. Lyons, "Toxgene: a template-based data generator for xml," in *SIGMOD '02: Proceedings of the 2002 ACM SIGMOD international conference on Management of data*. New York, NY, USA: ACM, 2002, pp. 616–616.

[29] C.-C. Pan, K.-H. Yang, and T.-L. Lee, "A flexible generator for synthetic xml documents," in *Proceedings of the International Conference on Information Networking (ICOIN 2003)*, 2003, pp. 1232–1239.

[30] G. J. Myers and C. Sandler, *The Art of Software Testing*. John Wiley & Sons, 2004.

[31] E. Day, "The use of asn.1 encoding rules for binary xml," http://www.obj-sys.com/docs/ASN1forBinXML.pdf (last accessed: August 15, 2011).