Model-Driven Software Engineering in Practice: Privacy-Enhanced Filtering of Network Traffic

Roel van Dijk
Netherlands Forensic Institute
The Hague, The Netherlands
dijk@holmes.nl

Christophe Creeten
Netherlands Forensic Institute
The Hague, The Netherlands
christophe@holmes.nl

Jeroen van der Ham
Delft University of Technology
Delft, The Netherlands
National Cyber Security Centre
The Hague, The Netherlands
jeroen.vanderham@ncsc.nl

Jeroen van den Bos
Zuyd University of Applied Sciences
Heerlen, The Netherlands
Netherlands Forensic Institute
The Hague, The Netherlands
jeroen@infuse.org

ABSTRACT

Network traffic data contains a wealth of information for use in security analysis and application development. Unfortunately, it also usually contains confidential or otherwise sensitive information, prohibiting sharing and analysis. Existing automated anonymization solutions are hard to maintain and tend to be outdated.

We present *Privacy-Enhanced Filtering (PEF)*, a model-driven prototype framework that relies on declarative descriptions of protocols and a set of filter rules, which are used to automatically transform network traffic data to remove sensitive information. This paper discusses the design, implementation and application of PEF, which is available as open-source software and configured for use in a typical malware detection scenario.

CCS CONCEPTS

• **Software and its engineering** → **Model-driven software engineering**; **Domain specific languages**; • **Security and privacy** → *Usability in security and privacy*;

KEYWORDS

model-driven engineering, domain-specific languages, open-source prototype, privacy-enhancing technology

ACM Reference format:
Roel van Dijk, Christophe Creeten, Jeroen van der Ham, and Jeroen van den Bos. 2017. Model-Driven Software Engineering in Practice: Privacy-Enhanced Filtering of Network Traffic. In *Proceedings of 2017 11th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering, Paderborn, Germany, September 4–8, 2017 (ESEC/FSE'17),* 6 pages.
https://doi.org/10.1145/3106237.3117777

The work described in this paper was carried out as part of the *Privacy-Enhanced Filtering*-project, which was funded by a grant from the *Secure through Innovation*-programme of the Dutch National Coordinator for Security and Counterterrorism.
Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).
ESEC/FSE'17, September 4–8, 2017, Paderborn, Germany
© 2017 Copyright held by the owner/author(s).
ACM ISBN 978-1-4503-5105-8/17/09.
https://doi.org/10.1145/3106237.3117777

1 INTRODUCTION

As more systems become connected, the amount of data passing through networks increases. This data is an important source of information for many tasks, including security analysis, debugging and optimization of the involved systems as well as in a broader sense, scientific research. Unfortunately, public datasets are rare because network traffic data often contains private, confidential or otherwise sensitive information, such as information about the internal network layout of an organization.

A solution is to anonymize or otherwise remove sensitive information from network traffic data. Tools exist to perform this task, but are often difficult to use, understand and extend and (as a result) tend to be outdated. This is especially a problem for the typical end-users of tools in this domain, technical users such as security and privacy officers that do not generally have extensive software engineering skills.

We propose a model-driven approach to network traffic data filtering, using declarative descriptions of network protocols and separated transformation rules. In this paper we describe the design and an open-source prototype implementation in a filtering framework called Privacy-Enhanced Filtering (PEF)[1]. The framework comes configured to perform filtering in a popular malware-detection scenario, where DNS packets are transformed to remove sensitive information.

Our practical evaluation shows that the framework performs its tasks correctly, while relying on declarative protocol descriptions for parsing and unparsing (serializing). Additionally, we discuss trade-offs and design considerations in the development of the framework, as well as future directions to let end-users take full advantage of this approach.

The rest of this paper is organized as follows. We provide some background on the domain of network traffic analysis in Section 2. In Section 3 we present a high level design of a model-driven approach towards developing a network traffic anonymizer, followed by a description of the implementation of PEF in Section 4. Section 5 describes the scenario that we have used to develop PEF and for which it is preconfigured as released. We discuss our experiences and trade-offs in Section 6 and conclude in Section 7.

[1] https://github.com/NCSC-NL/PEF

2 NETWORK TRAFFIC ANALYSIS

Network security often depends on detecting specific signatures in network traffic data. There are however many other common patterns, such as the use of some specific protocol, some interaction model or communication with a specific set of servers. All these things can indicate whether the security of the network runs specific risks, or even whether computers on the internal network may have already been compromised.

Analyzing network traffic is also useful in the context of application development and maintenance, where detecting patterns in specific use cases can help to debug problems or optimize communication patterns. Especially with large applications with many users, typical usage of an application may be difficult to determine but can be quickly gleaned from network traffic data.

This leads to a more general scientific research area around networking, which depends heavily on available data and datasets. A widely cited dataset has been generated by DARPA, but is already almost 20 years old[17]. The situation has improved since the launch of the IMPACT project[6], making it possible for researchers from different countries to access computer security related datasets. These datasets are not publicly available though, because they may contain sensitive data.

2.1 Challenges around Sharing Network Traffic

The most common reason that network traffic is not available for scientific research is that it contains confidential or private information. This is not only present in the content of the network traffic, even the metadata of network communication, the header-data, may still contain personally identifiable information, or other sensitive information. Examples include IP-addresses and MAC-addresses, but it may also be timing information that can be correlated to existing datasets.

This sensitivity of network traffic has been an issue for scientifically shared data. Allman and Paxson[1] discuss the shortage of network trace datasets, but more importantly the issues and etiquette of dealing with these datasets. De-anonymizing measurement data is an important research topic, Allman and Paxon conclude that findings on specific datasets should be reported in a careful manner.

2.2 Automated Anonymization Tools

Anonymization of network traffic is possible, but is generally considered very difficult to do[19]. While there has been some research on traffic anonymization, only a few tools are available that implement anonymization broadly. Most notable and with an extensive feature set are pktanon[8], anontool[14] and PCAPAnon [16].

Anonymization frameworks and libraries, including those mentioned above, generally implement their functionality in C, exposing configuration options at the level of policies and transformation selection. The protocol parsers are either reused from existing frameworks such as Wireshark or implemented in a custom library. As a result, their target audience cannot easily extend or evolve protocol definitions or add new protocols to support. The complexity in maintaining such frameworks is also reflected in that most systems are not actively maintained, which has caused them to quickly become outdated.

3 A MODEL-DRIVEN APPROACH

Model-driven engineering (MDE) [21] is a good fit for addressing the challenges of network traffic filtering. Its focus on developing a domain-specific language or notation allows end users to develop and adapt solutions without having to understand or worry about the underlying implementation details. This means that adding new or maintaining existing protocol specifications and transformation rules can potentially all be handled by security and privacy officers directly, instead of relying on software engineers.

Additionally, because the underlying implementation is decoupled from these specifications, it can be evolved without knowledge of the current state of protocol specifications and transformation rules. In this case, the software engineers can freely optimize parsing and transformation code or add enhancements to scalability and other non-functional aspects without having to migrate or re-implement the domain logic.

A conceptual design of a solution in this area is shown in Figure 1. At a typical location where data is captured between an internal network and the internet, a system can be placed that copies and filters all data before passing it on to some third party or directly into an intrusion detection system (IDS). In our design this system is split into three separate tasks that each maps well onto a model-driven approach: *Parse*, *Transform* and *Unparse*.

The *Parse* task uses specifications of network protocols to implement a parser for data in those protocols. Binary data is converted into a structured representation in memory, which is then passed on. The *Transform* task uses transformation rules to replace specified values with some (derived) value. For example, fields that have been marked in the specification as privacy-sensitive may be replaced or transformed, in order to remove (part of) their content.

Finally, the *Unparse* task does the reverse of the *Parse* task, by serializing the structured representation back into a binary data stream. To accomplish this, it can use location information in the structured representation as well as the protocol specifications. By serializing back into a valid network traffic stream, the output can be used by regular network analysis tools such as Wireshark, even though the data has been transformed.

4 THE PRIVACY-ENHANCED FILTERING FRAMEWORK

We have created *Privacy-Enhanced Filtering (PEF)*, an open-source prototype framework that transforms network traffic in network capture files. The prototype can be configured to transform parts of network traffic that match specific criteria. Each network packet is processed in three steps: parsing the packet, transforming specified values and serializing the parsed values back into a valid network packet. It realizes the model-driven design discussed in the previous section.

4.1 Parsing Binary Data with METAL

METAL[2] is a Java library for parsing binary data formats, using declarative descriptions. It was developed at the Netherlands Forensic Institute to improve parsing of binary data, and is available as open source. In digital forensics, parsing binary data is a major

[2]https://github.com/parsingdata/metal

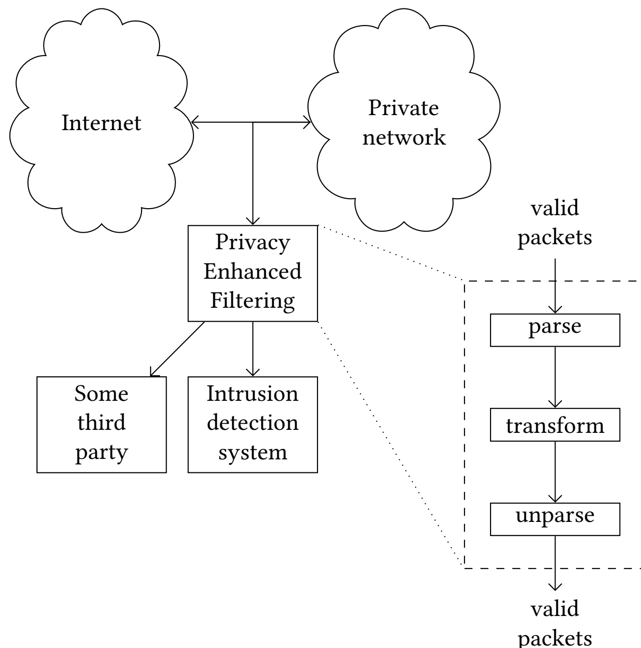

Figure 1: Personally identifiable information is automatically removed from captured network traffic data.

concern as nearly all digital evidence resides in often large amounts of binary data, such as live memory dumps, digital storage devices and network captures.

Traditionally, handling of binary data was a matter of handcrafted parsers, which were difficult to extend and evolve. A data description language such as METAL makes it possible to create specifications of the data formats to be supported, completely separate from the underlying implementation of the eventual parser. This makes METAL well-suited for PEF: parsing network packets is achieved by declaratively specifying the required network protocols, which are essentially just a special case of nested data formats.

4.1.1 Composing Parsers. Declarative descriptions in METAL are created using a Java internal DSL. Every statement in the METAL DSL is either a *token* or an *expression*. There are two types of tokens: terminals and non-terminals.

A terminal token in METAL is the `def` token, which reads bytes from the input. The non-terminal tokens provide a way to combine other tokens into more complex structures, such as sequences of tokens (`seq`), choices between tokens (`cho`), repetition of a token (`rep`) or conditional parsing of a token (`pre` and `post`).

Apart from these typical regular expression-style tokens, METAL also provides tokens to parse more complex data structures. Binary data formats often contain pointers to data structures at specific offsets in the input data. The `sub` token allows for location-dependent parsing, by parsing tokens at specific offsets. The `tie` token allows for nested parsing, e.g., parsing a token inside the result of some expression, which typically refers to (a composition of) earlier parsed values.

There are many different uses of expressions in token definitions. Examples are defining the size of terminal tokens or validating the value of some parsed value based on data elsewhere in the input data (both common in data-dependent parsing[10]).

Expressions are capable of operating on the value currently being parsed, referencing earlier parsed values or constant values. Apart from the regular arithmetic and bitwise operations, there are additional data handling expressions such as `fold`, concatenate (`cat`) and `map`. All expressions handle and return lists of values, which is useful in binary data formats where data dependencies are often more complex than only referring to directly preceding values.

METAL is byte-oriented, i.e., field lengths are expressed in number of bytes. Most binary data structures can be described using this approach, which makes it generally well-suited. It does sometimes pose a challenge when specifying smaller data structures such as bit fields or compression tables (e.g., Huffman tables). Even then, bitwise expressions can be used to specify these data formats correctly.

How to interpret values in comparisons can be configured in an *encoding* object used by the parser. This object is used to specify the interpretation of numeric values (such as endianness and whether they are signed or not) and strings (character encoding).

4.1.2 Executing Parsers. A constructed parser can be run by invoking the `parse`-method on a token, passing an input stream and an object specifying encoding defaults to it. Parsing succeeds if the top-level token that was invoked returns successfully, which indicates that all data could be read and that a path exists through the composition of tokens that satisfies all predicates.

Upon success, the parse result is stored in a graph structure. Each node in this graph corresponds to a token in the declarative description. Intermediate nodes correspond to the non-terminal tokens, such as sequences, choices and repetitions, the leaf nodes of the graph correspond to the terminal tokens, and contain the parsed values in the input stream. Each parsed value not only contains the bytes that were parsed, but also the offset at which these bytes occurred in the original stream.

The returned data structure is a graph and not a tree, because METAL supports parsing of circular data structures such as circular linked lists. To prevent parsing a circular data structure from creating a loop, it employs a cycle detection mechanism that triggers when a token is parsed at a location where the same (or an identical) token was previously parsed already. In this case the loop is created in the graph and parsing continues with the next token.

4.2 Parse

PEF uses METAL to specify network protocols, which are used to parse captured network packets. The protocols that have been implemented in this prototype are:

- Link Layer: Ethernet 2 Frame
- Internet Layer: IPv4, ICMPv4, IPv6 (partially)
- Transport Layer: UDP, TCP
- Application Layer: DNS-formatted application data (regular DNS, MDNS, LLMNR, NBNS)

Figure 2 shows an example of a declarative description of the IPv4 header using the METAL DSL. The header is defined as a sequence

```
Token IPV4_HEADER = seq(
  def("versionihl", 1,
    and(
        eqNum(shr(self, con(4)), con(4)),
      gtEqNum(and(self, con(0x0F)), con(5)))),
  def("dscpecn", 1),
  def("iplength", 2, gtEqNum(con(20))),
  def("identification", 2),
  def("flfr", 2),
  def("timetolive", 1),
  def("protocol", 1),
  def("headerchecksum", 2),
  def("ipsource", 4),
  def("ipdestination", 4)
);
```

Figure 2: METAL token of the IPv4 header

of ten fields[20]. The sequence (`seq`) token requires that all nested tokens should parse successfully in the order as specified. In this case, the ten nested tokens are all define-value (`def`) tokens. A `def` token specifies a value to parse from the input stream, and is defined by a name, a length in bytes, and an optional predicate.

The statement `def("iplength", 2, gtEqNum(con(20)))` defines a field named `"iplength"`, which has a length of 2 bytes. The predicate `gtEqNum(con(20))` states that the value of this field should be larger than or equal to 20, as defined in the IPv4 specification. If this value could be read, but its interpreted value is smaller than 20, the parsing of the `def` fails, and because of that the parsing of the enclosing `seq` fails as well.

Predicates on values can be more complex, as is seen in the `"versionihl"` field. Its predicate:

```
and(
      eqNum(shr(self, con(4)), con(4)),
    gtEqNum(and(self, con(0x0F)), con(5))))
```

states that two conditions have to be met: the first four bits have to equal the value 4 (indicating IPv4), and the second four bits need to have a value larger or equal to the value 5 (indicating the minimum length of 5×32 bits = 20 bytes).

The protocols in the different network layers have been implemented using the METAL DSL, and combined into a single Ethernet 2 Frame token. This is the level at which individual transformations are performed.

4.3 Transform

Once a network packet parses successfully, the values in the parse graph are transformed if they match a set of constraints. Each constraint consists of three parts: a list of network protocols that should be present in the network packet, the name of the field to transform, and the method to use for transformation. Transformation types are typically rewriting or (checksum) recalculation.

Examples of transformations used in the prototype are:

- Rewrite the `"ipsource"` field in a network packets containing IPv4, UDP and DNS.
- Rewrite the `"destinationaddress"` field in a network packets containing IPv6, UDP and DNS.
- Recalculate the `"udpchecksum"` field of IPv4 packets containing UDP.

There are several types of rewriting that can be used, including replacement with a fixed value, up to more complex transformations such as application of a filter or cryptographic function. Transformations are implemented in Java but assigned and parameterized through transformation rules applied to the protocol descriptions.

Some of the network protocols parsed by PEF contain checksum values which ensure the validity of the network packet: IPv4, TCP, UDP and ICMPv4 all contain checksums. Since parts of network packets are altered, the checksums need to be recalculated before unparsing. The PEF prototype ensures that checksums are recalculated in all transformed packets. The resulting network capture will then contain valid network packets that can be analyzed by other tools.

4.4 Unparse

Every value in a METAL parse graph contains the offset at which it occurred in the original data stream, along with the byte values (the data itself) and an encoding object. A PEF transformation retains the offset and the size of the data. Serialization of the transformed packets is therefore trivial: collect all the values and output them at their offsets.

5 PRIVACY-ENHANCED MALWARE DETECTION

To facilitate experimentation and evaluation, the open-source release of PEF is configured to be usable directly in a typical scenario: detecting Advanced Persistent Threat-type malware through DNS requests in network traffic captures[24]. This relates to a type of malware that settles on an infected computer for an extended period and controls the system via received instructions from a command and control (C&C) server.

To receive commands, the malware must contact a C&C server, which often involves DNS requests. Intrusion detection systems (IDS) can be used to analyze network traffic and detect DNS requests about suspicious hostnames. This approach poses a problem however, because the IP address of a device on a network can be directly related to the person using that device. Analyzing DNS requests will not only inspect malicious network traffic, but also expose legitimate and personally identifiable network traffic.

Analyzing DNS requests in a network capture is a good use case for PEF, as it can be configured to hide IP source addresses of all those requests. Transforming original IP addresses can significantly complicate tracing the request back to the person making those requests, which enhances privacy of the users. IP addresses can be transformed in various ways, e.g., by replacing them with an empty or randomly generated value. This basic approach will create a problem during malware analysis, because it either seems that all DNS requests come from the same empty IP address, or multiple requests from the same device all have randomly created IP addresses and cannot be related to or separated from each other.

5.1 Source Address Pseudonymization

Another option to hide IP addresses in network traffic is using a form of format-preserving encryption. This technique ensures that the input format is encrypted into the same output format, e.g., a 32-bit IPv4 address is encrypted into a different 32-bit IPv4 address. The prototype provides an implementation of the FFX mode of operation for format-preserving encryption, as defined by NIST [2]. Applying FFX to IP addresses pseudonymizes the network traffic, because the same IP address will always be transformed into the same adjusted IP address. This is a useful approach, because it allows a malware analysis to determine how widespread an infection is on a network by counting unique IP addresses.

PEF has been configured to support this use case. The command line tool supports parsing PCAP files, and pseudonymizing IP addresses found in network packets that contain DNS traffic. The tool writes the results to a new PCAP file, which can be analyzed by traditional network analysis tools.

The open-source release contains several example PCAP files which are used in the test code to verify expected behavior. The prototype also has experimental support for real-time rewriting of streams so that it can be included in real-time monitoring solutions such as an IDS.

6 DISCUSSION

6.1 Technology Choice and Tool Development

PEF is part of an on-going experiment at the Netherlands Forensic Institute to develop and deploy a model-driven solution to the problem of maintaining parsers for quickly evolving binary data formats. As an internal DSL in Java, Metal works well for forensic software engineers, since the library is just a small dependency and integrates easily in the Java-based workflow at the NFI. While IDE support is not extensive or domain-specific, an internal DSL in Java can quickly be made usable by utilizing relatively simple things such as factory methods and Javadoc (for intellisense support).

For a framework such as PEF, the eventual technology choice may end up being different as the target users are not software engineers but security and privacy officers. For these users, installing a Java development environment, recompiling solutions after modification and handling dependencies with Maven may prove difficult. In this situation, extending Metal towards a language with its own concrete syntax and standalone tooling may be necessary.

One of the forerunners of Metal was such a solution, including a full language[3] and IDE [4]. An important lesson learned in this area is that in order to make the right technology choices in MDE, it is not only important to have deep knowledge of the application domain, but also of the exact usage scenarios of any solution, along with a complete view of the users. It seems that these are at least equally important in deciding which approach to use.

6.2 Experiment and Evaluation

We have not provided an extensive empirical evaluation of the filtering solution in practical use in this paper. One of the reasons is that our research focuses on whether the area of network traffic filtering lends itself well to an MDE approach. Our goal is to show that a working solution can be constructed and that it works well based on the provided specifications and test data. Questions such as whether IP address pseudonymization and malware detection via DNS are viable and useful are important, but not in the context of this paper.

Any evaluation should take two questions into account however: whether an anonymization scenario can be fully realized using PEF or a similar approach, but also whether this approach actually works well for the type of users that need to maintain the solution.

Additionally, as an indication that anonymization tools are not widely used, it is quite difficult to publish a data set to test such tools. The only acceptable way policies typically allow the release of network traffic data is *after* anonymization. This would at the very least complicate using it as evaluation data to show anything about the workings of a framework that does similar things.

6.3 Related Work

There is ample research in the area of data description languages and associated tools such as interpreters and code generators. Fisher et al.[7] provide an extensive and authoritative overview, including a discussion of their PADS language and tools. More recently, parsing binary data has gained an interest from the security community through an approach called language-theoretic security[18]. Active projects in this area are binary data parsing toolkits such as Hammer[22] and Nom[5]. Metal fits into this as a pragmatic library that attempts to cover the domain of binary data formats completely.

Model-driven engineering is a large area both in research and practice that is continually evaluated in many studies[9, 11, 12, 15]. What sets this research apart from most evaluations is that we provide a full prototype model-driven solution as open-source software configured for use in a realistic scenario, facilitating immediate experimentation.

The transformation pipeline of *parse-transform-unparse* that PEF uses is inspired by and similar to those used in other domains. Examples of such approaches are EASY [13] in source code transformation, and ETL [23] in data warehousing. Especially in source code transformation the concern of serializing back to the same valid structures is common, as the resulting output source code typically needs to be a valid program according to the same grammar as the input.

7 CONCLUSION

In this paper we present an open-source prototype implementation of a model-driven approach to network traffic filtering, called *Privacy-Enhanced Filtering*. It separates specification of protocols from parser implementation, allowing many maintenance tasks to be performed by its direct users, which usually are security and privacy officers. We have based our implementation on prior experiences in developing model-driven solutions in this area, including building upon our binary data description DSL Metal.

The framework comes configured for use in a relevant and practical network traffic filtering scenario: pseudonomyzing DNS requests to enhance end-user privacy when attempting to detect malware infections on internal networks. The prototype can be installed to work in tandem with an existing intrusion detection system.

Our implementation demonstrates that it is possible to implement a network traffic filtering framework using declarative data descriptions, both in general and in the specific case when using the METAL DSL. It shows that data structure description, transformation rule implementation as well as definition of where and how to apply those transformations can all be cleanly separated.

We intend to expand this work to include more end-user tools, such as a standalone development environment to integrate the declarative descriptions with the transformation rules. Additionally, we plan to continue active development of PEF itself, adding more protocols and application scenarios, as well as perform several practical evaluations.

REFERENCES

[1] Mark Allman and Vern Paxson. 2007. Issues and Etiquette Concerning Use of Shared Measurement Data. In *Proceedings of the 7th ACM SIGCOMM Conference on Internet Measurement (IMC'07)*. ACM, 135–140. https://doi.org/10.1145/1298306.1298327

[2] Mihir Bellare, Phillip Rogaway, and Terence Spies. 2010. The FFX mode of operation for format-preserving encryption. *NIST submission* 20 (2010).

[3] Jeroen van den Bos and Tijs van der Storm. 2011. Bringing Domain-Specific Languages to Digital Forensics. In *Proceedings of the 33rd International Conference on Software Engineering (ICSE'11)*. ACM, 671–680. https://doi.org/10.1145/1985793.1985887

[4] Jeroen van den Bos and Tijs van der Storm. 2013. TRINITY: An IDE for The Matrix. In *Proceedings of the 29th IEEE International Conference on Software Maintenance (ICSM'13)*. IEEE, 520–523. https://doi.org/10.1109/ICSM.2013.86

[5] Geoffroy Couprie. 2015. Nom, A Byte oriented, streaming, Zero copy, Parser Combinators Library in Rust. In *Security and Privacy Workshops (SPW), 2015 IEEE*. IEEE, 142–148. https://doi.org/10.1109/SPW.2015.31

[6] Department of Homeland Security. 2016. The Information Marketplace for Policy and Analysis of Cyber-risk & Trust (IMPACT). (2016). Retrieved 2 July 2017 from https://www.impactcybertrust.org/

[7] Kathleen Fisher, Yitzhak Mandelbaum, and David Walker. 2010. The Next 700 Data Description Languages. IT J. ACM 57, 2 (2010), 10:1–10:51. https://doi.org/10.1145/1667053.1667059

[8] Thomas Gamer, Christoph P. Mayer, and Marcus Schöller. 2008. PktAnon–A Generic Framework for Profile-based Traffic Anonymization. *PIK-Praxis der Informationsverarbeitung und Kommunikation* 31, 2 (2008), 76–81.

[9] John Edward Hutchinson, Mark Rouncefield, and Jon Whittle. 2011. Model-Driven Engineering Practices in Industry. In *Proceedings of the 33rd International Conference on Software Engineering (ICSE'11)*. IEEE, 633–642. https://doi.org/10.1145/1985793.1985882

[10] Trevor Jim, Yitzhak Mandelbaum, and David Walker. 2010. Semantics and Algorithms for Data-Dependent Grammars. In *Proceedings of the 37th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL'10)*. ACM, 417–430. https://doi.org/10.1145/1706299.1706347

[11] Stefan Karg, Alexander Raschke, Matthias Tichy, and Grischa Liebel. 2016. Model-Driven Software Engineering in the OpenETCS Project: Project Experiences and Lessons Learned. In *Proceedings of the ACM/IEEE 19th International Conference on Model Driven Engineering Languages and Systems (MODELS'16)*. ACM, 238–248. https://doi.org/10.1145/2976767.2976811

[12] Michael Kläs, Thomas Bauer, Andreas Dereani, Thomas Soderqvist, and Philipp Helle. 2015. A Large-Scale Technology Evaluation Study: Effects of Model-based Analysis and Testing. In *Proceedings of the 37th International Conference on Software Engineering (ICSE'15)*. IEEE, 119–128. https://doi.org/10.1109/ICSE.2015.141

[13] Paul Klint, Tijs van der Storm, and Jurgen J. Vinju. 2009. EASY Meta-programming with Rascal. In *Generative and Transformational Techniques in Software Engineering III - International Summer School (GTTSE'09)*, Vol. 6491. Springer, 222–289. https://doi.org/10.1007/978-3-642-18023-1_6

[14] D. Koukis, S. Antonatos, D. Antoniades, E. P. Markatos, and P. Trimintzios. 2006. A Generic Anonymization Framework for Network Traffic. In *Proceedings of the International Conference on Communications (ICC'06)*, Vol. 5. 2302–2309. https://doi.org/10.1109/ICC.2006.255113

[15] Vinay Kulkarni. 2016. Model Driven Development of Business Applications: A Practitioner's Perspective. In *Proceedings of the 38th International Conference on Software Engineering (ICSE'16) - Companion Volume*. ACM, 260–269. https://doi.org/10.1145/2889160.2889251

[16] Y. D. Lin, P. C. Lin, S. H. Wang, I. W. Chen, and Y. C. Lai. 2016. PCAPLib: A System of Extracting, Classifying, and Anonymizing Real Packet Traces. *IEEE Systems Journal* 10, 2 (2016), 520–531. https://doi.org/10.1109/JSYST.2014.2301464

[17] Richard P. Lippmann, David J. Fried, Isaac Graf, Joshua W. Haines, Kristopher R. Kendall, David McClung, Dan Weber, Seth E. Webster, Dan Wyschogrod, Robert K. Cunningham, et al. 2000. Evaluating intrusion detection systems: The 1998 DARPA off-line intrusion detection evaluation. In *Proceedings of the DARPA Information Survivability Conference and Exposition (DISCEX'00)*, Vol. 2. IEEE, 12–26. https://doi.org/10.1109/DISCEX.2000.821506

[18] Falcon Momot, Sergey Bratus, Sven M. Hallberg, and Meredith L. Patterson. 2016. The Seven Turrets of Babel: A Taxonomy of LangSec Errors and How to Expunge Them. In *Proceedings of IEEE Cybersecurity Development (SecDev'16)*. IEEE, 45–52. https://doi.org/10.1109/SecDev.2016.019

[19] Ruoming Pang, Mark Allman, Vern Paxson, and Jason Lee. 2006. The Devil and Packet Trace Anonymization. *SIGCOMM Comput. Commun. Rev.* 36, 1 (2006), 29–38. https://doi.org/10.1145/1111322.1111330

[20] Jon Postel (Ed.). 1981. *RFC 791 Internet Protocol - DARPA Internet Program, Protocol Specification*. Internet Engineering Task Force. http://tools.ietf.org/html/rfc791

[21] Douglas C. Schmidt. 2006. Model-Driven Engineering. *Computer* 39, 2 (2006), 25–31. https://doi.org/10.1109/MC.2006.58

[22] UpstandingHackers. 2012. Hammer, Parser combinators for binary formats in C. (2012). https://github.com/UpstandingHackers/hammer

[23] Panos Vassiliadis. 2011. A Survey of Extract-Transform-Load Technology. In *Integrations of Data Warehousing, Data Mining and Database Technologies - Innovative Approaches*. IGI, 171–199. https://doi.org/10.4018/978-1-60960-537-7.ch008

[24] Guodong Zhao, Ke Xu, Lei Xu, and Bo Wu. 2015. Detecting APT Malware Infections Based on Malicious DNS and Traffic Analysis. *IEEE Access* 3 (2015), 1132–1142. https://doi.org/10.1109/ACCESS.2015.2458581