The Inference Graph of Cybersecurity Rules

Dawood Sheniar
Faculty of HES
University of Southern Queensland
Toowoomba, Australia
dawoodsallemhussian.sheniar@usq.edu.au

Nabeel Hadaad
Faculty of HES
University of Southern Queensland
Toowoomba, Australia
NabeelMahdy.Hadaad@usq.edu.au

Ron Addie
Faculty of HES
University of Southern Queensland
Toowoomba, Australia
ron.addie@usq.edu.au

***Abstract*—The concept that cybersecurity architecture is the discovery, definition and validation of *rules* is introduced. The new concept of *inference graphs* for illustrating the relationship between cybersecurity rules is defined. Three increasingly complex examples of inference graphs for systems needing cybersecurity architecture are presented, including the detailed proofs which form the basis of these inference graphs, in some cases. The software which has been developed to support the development and use of cybersecurity inference graphs is described including details of the public server where it can be used. It is shown that cybersecurity inference graphs can significantly contribute to development of, and validation of cybersecurity and also that rigorous validation of cybersecurity is not necessarily as difficult as previously thought.**

***Keywords*—Cybersecurity rules, inference graph, security design, proof, stakeholder security analysis, cybersecurity architecture, Netml.**

I. Introduction

Cybersecurity is essentially the enforcing of the rules required by the stakeholders of the system under consideration. The collection of cybersecurity rules forms a graph, in which the vertices represent cybersecurity rules, and the edges represent *inference*, in the sense that the collection of vertices with edges *terminating* at a certain vertex, V for example, correspond to the set of rules which are sufficient to prove the rule corresponding to V. This representation of cybersecurity provides a way to develop cybersecurity architecture[10]. These diagrams can also be used in security design, as a way to validate security designs, for documentation, and as a way to visualise and develop alternative designs.

Cybersecurity systems have used Knowledge Graphs (KG) to store and retrieve data to make decisions about cyber-attacks [9], [8], [5], [3]. Graphical representation of logical or structural properties of computer systems is a well established practice in computer science [4]. Inference graphs, as introduced and explained in this paper, provide a graphical representation of cybersecurity architecture.

A. Objectives of this research

The aim of this research is to develop concepts and tools which assist cybersecurity professionals to define the security objectives of their systems, the rules which enforce this security, and the reasoning behind the choice of these rules. The concepts and tools we propose and illustrate in this paper will be shown to be sufficient to record and analyse these rules, both the objectives and the rules which are enforced, and to ensure that there are no logical flaws in the system which has been designed. Note that we assume our cybersecurity objectives include the control of all important risks.

In Example 1 of Section III, we see how a key design objective is achieved rigorously by a clever decision which emerges from the use the cybersecurity inference graph of this system. Example 2 is based on a cybersecurity weakness in a web service administered by one of the authors, which was analysed and solved previously [12] so in this case the inference graph is used to illustrate the proofs which were developed then. In Example 3, finding the correct proofs was a complicated task which was facilitated by the use of the inference graph of rules and objectives of the system.

B. Stakeholder Analysis

Stakeholder analysis [7], [11], [6] begins by identifying those who influence or who are influenced by decisions or outcomes of the system. There are several ways to classify stakeholders some of which depend on their threat potential.

As set out in [6], the most important reason for a careful stakeholder analysis, from the point of view of this paper, is that if we are able to identify a sufficient set of rules to ensure the willing participation of each stakeholder, and if we can enforce these rules, then the system is self-evidently secure. Through experience, stakeholders may, during the lifetime of a system, discover that there are rules which were not initially obvious and which need to be added to their required rule set. This should not inhibit us from attempting to identify as complete as possible a set of stakeholder rules. Furthermore, if sufficient care is taken with stakeholder analysis, discoveries of missing rules should be infrequent.

C. The Netml system

Netml is a cloud-based system developed by the University of Southern Queensland and City University of Hong Kong. It is used by students to analyse, design, and simulate networks [1], [2]. It is freely available for use (in the cloud) at `https://netml.org`. The Netml system is used in this research to create and modify inference graphs.

A tool which enables graphical editing of inference graphs is essential because the layout of an inference graph can improve its clarity and usefulness considerably, and the only practical way to develop this layout is by manual editing. In addition, the tools developed in this research, which are described in Section IV, may be used via the most recent version of the public Netml server.

Section II defines and explains the concept of an inference graph. Section III uses three examples to illustrate their use. Section IV describes the software tools which have been developed to enable inference graphs to be readily developed and managed. Finally, Section V presents conclusions. A sample of the software which has been developed is presented in the appendix.

II. Inference Graphs

An inference graph shows the relationship of *inference* (what implies what) which applies between the different cybersecurity rules which apply in a system. Examples of inference graphs are shown in Figures 2 and 4. These will be explained in Section III.

The following types of rules arise in cybersecurity analysis: *objective*, *enforced rule*, *axiom*, *proposition*, and *assumption*. An objective is a rule which is required to be true at all times. An enforced rule is a condition which it is possible to enforce, and which it has been decided to enforce, by technical means. For example, access to many systems is only provided if a user is able to enter a valid username and password.

An axiom is a condition which is held to be true *a priori*; an assumption is a condition which we *choose* to believe. For example, under some conditions we assume that users do not reveal their password to other users. A proposition is a rule or statement which we define in order to express a useful stage of reasoning.

An *edge* in an inference graph connects each of the rules which are referenced in a proof to the rule which is proved. Objectives are typically the destination of edges, while enforced rules usually occur only as the origin of an edge.

A. Proof as a relationship

The *goal* of cybersecurity is to guarantee that certain objectives are maintained. For example, it is likely that a bank will have, as an objective, that no transactions – transfers of money from one account to another – occur except with valid authorization.

The aim of cybersecurity *design* is to discover or instantiate axioms, assumptions, and enforced rules which enable us to *prove* that the *objectives* are true. Along the way to doing this there may be some *intermediate propositions* that we also wish to prove.

Thus, *objectives* and *intermediate propositions* have proofs. On the other hand, *axioms* do not have proofs because these are fundamental truths that are true from logical principles or, in some cases, because they express follow from the definition of the predicates they contain, *assumptions* are true by assumption (which might not always hold, but at present we adopt them), and *enforced rules* are true because we make sure, in the system, that they are true, so none of these rule types have proofs. The appearance of a reference to a proposition, assumption, axiom, objective, or enforced rule, in a proof, constitutes a relationship between that rule and the rule being proved. The *inference graph* has vertices or nodes corresponding to the rules, and directed edges or links from any rule which is referenced in a proof to the rule which is proved.

Table I: Rules for a parcel box

Rule Name	Details
O1	Parcels cannot be stolen (taken from the box by someone other than the owner) from the parcel box
O2	Parcels can be retrieved from the parcel box by the owner of the box
O3	Deliverers are able to store delivered goods in the box whenever they visit the property with an item to be delivered
A1	The owner of the parcel box does not allow access to the codes they receive to anyone else
E1	Codes are sent by a secure path to the owner of the parcel box
E2	Codes are sent by a secure path to the parcel box
E3	Codes for access to the box are generated and stored on the server in a system which does not provide read access to any person, agent, or process, except for a process which sends them to the owner of the parcel box, and to deliverers, and this process cannot be used to send the codes to anyone else
E4	Parcels cannot be removed from a locked box without a code for opening it
E5	Parcels can be removed from a locked box by anyone with the code for opening associated with the parcel it contains
E6	The parcel box can be opened by the code sent to a deliverer, when, and only when, it is empty.
E7	Deliverers are scheduled to visit the parcel box only when it is empty.

In the diagrams which are included below, the different types of rules are represented by nodes of different shapes and colours, as indicated in the Legend, shown in Fig. 1.

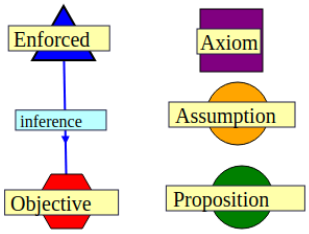

Figure 1: Legend for Inference graphs

III. Examples

A. A parcel box

Consider a box for delivering parcels which can be opened only by the owner and those delivering parcels

The objectives of the system as a whole are:

- Deliverers are able to store delivered goods in the box whenever they visit the property with an item to be delivered.
- Goods can not be taken from the box except by the owner(s) of the box or their agents.
- The owner of the box is fully informed concerning any deliveries to or removals from the box.

The objective, assumptions, and enforced conditions for this example are listed in Table I.

The inference graph for these rules is shown in Fig. 2.

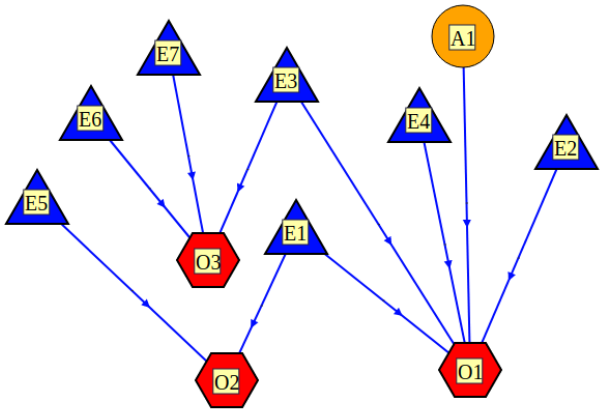

Figure 2: Inference graph of rules for a parcel box

This example illustrates the way in which assumptions can, and should, influence design of a system. If we assume that deliverers can be trusted implicitly, there is no need to ensure that deliverers can only open a parcel box when it is empty. If, on the other hand, we do not make this assumption, the system will need to ensure that deliverers cannot open a parcel box which contains a parcel. This approach is more secure, but a little more difficult to manage. In the past, assuming that deliverers can be implicitly trusted would not seem unreasonable, but in future, as the range of delivery options increases, such an assumption might come to seem unnecessary and unrealistic.

The decision to schedule deliveries to take place only when the parcel box is empty was made, in the development of this system, in conjunction with, and under the influence of the cybersecurity inference graph. Objective O1 could be ensured by assuming that deliverers can be trusted*, but a significantly better design results by searching for an alternative to this risky assumption.*

B. A password reset system

Consider a portal which maintains accounts for users of its services and which has a *password reset service*, such as discussed in [12]. The rules which apply to the stakeholders of a service for which a password reset service is available are listed in Table II.

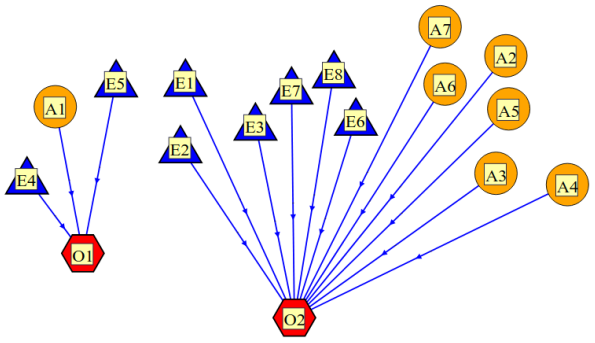

Figure 3: Inference graph of rules for a password reset system

Table II: Rules for the tickets in a password reset system

Rule Name	Details
O1	Users are able to reset their password by receiving a ticket by email and using this ticket at the website within 30 minutes, assuming that they provide an incorrect ticket at most twice.
O2	Agents/persons other than a valid user cannot use this system to reset the password of a user.
A1	Ticket's sent to users are received by email within 1 minute.
A2	Only persons/agents who know a user's email password can access email sent to the user within the last 30 minutes.
A3	The user's password cannot be guessed.
A4	The user is the only person/agent who knows their email password.
A5	The possibility of an attacker intercepting a ticket on the user's email client, during its 30 minutes of valid life is negligible.
A6	The possibility guessing a ticket in fewer than 10,000 attempts is negligible.
A7	The server administrator never acts in a way to subvert the intentions of the system he/she administers.
E1	Use of an invalid ticket to reset a password three times or more in 10 minutes, for a certain user, causes any outstanding tickets for that user to become invalid.
E2	Tickets supplied 31 minutes after generation are no longer valid.
E3	When a ticket is successfully used, it will not be valid in future.
E4	If a user supplies a valid ticket, at the service web site, they can use this site to set their password to a new setting, without knowing the existing password.
E5	Any user can cause a ticket to be generated and sent to the user's previously registered email address, by appropriate actions at the service web site.
E6	The *only way* to reset a password is by supplying a valid password reset ticket.
E7	The *only way* to generate a password reset ticket is by the password reset service.
E8	Password reset tickets cannot be accessed by any user other than the administrator, on the server where they are generated.

The key requirements for this system are O1 and O2 in this table, i.e. the service can be used to change passwords, but users can only change passwords for *their own* accounts. Such systems are vulnerable to logical errors, as discussed in [12]. An inference graph for this system, which can be used to guide its validation, is shown in Fig. II.

C. Signatures

The validity of the signature of a document can be checked by the digital signature algorithm. Once this checking has been done, we can be confident that a person who has, in their posession, the private key of a certain digital certificate signed the document.

What is being checked here is not that the signer is the author, but that they are the signer, of the document. Suppose we want to check that Joan is a signatory of a certain document, `d`. It is not sufficient to merely check the signature, with the document, and a certain certificate. The validity of the signature also needs to be checked, and this in turn requires additional checks. A proof of the statement "Joan is a valid signatory of document `d`", which undertakes all these needed checks, is illustrated in Fig. 4. The rules which are used, in this checking process, are described in Table III.

With the help of the predicates in Table IV and the names of the objects listed in Table V, the axioms, listed informally in Table III, can now be written, more formally as follows:

Table III: Rules for signatures

Rule Name	Details
O1	Joan is a valid signatory of document jdoc.
E1	The document d has signature ds.
E2	Document d is valid according to the digital signature ds,which uses certificate jc
E3	Joan has certificate jc.
E4	The certificate jc has signature jcs.
E5	Certificate jc is valid according to signature jcs which uses certificate cc
E6	CA has certificate cc
E7	cc has signature cs
E8	cc has signature cs which is valid according to certificate vc
A1	If A is a valid authority, c is a certificate with signature s and the signature s is valid according to A, then c is a valid certificate.
A2	If A has certificate c, d has signature s, certificate c is valid and the signature s is valid according to c, then A is a signatory of d.
A3	VA is a valid authority with certificate vc
A4	If A has certificate c and c is valid, then A is a valid authority with certificate c
P1	cc is a valid certificate.
P2	CA is a valid authority with certificate cc.
P3	jc is a valid certificate.

Table IV: Predicates used to validate signatures

Predicate	Meaning
HasSig(x,y)	x is a document and y is its signature
IsValidAccTo(x,y,z)	x is a signature for document y which is valid according to certificate z
IsSignatory(d,A)	A is a signatory of the document d
HasCert(A,c)	c is a certificate owned by A
IsValidCert(x)	x is a valid certificate
IsValidAuthority(X,xc)	X is a valid a certificate authority with certificate xc

A1: $\forall A,c,s,ac$: $\texttt{IsValidAuthority}(A,ac) \wedge \texttt{HasSig}(c,s) \wedge \texttt{IsValidAccTo}(s,ac,A) \supset \texttt{IsValidCert}(c)$.

A2: $\forall A,d,c,s : \texttt{HasCert}(A,c) \wedge \texttt{HasSig}(d,s) \wedge \texttt{IsValidCert}(c) \wedge \texttt{IsValidAccTo}(s,d,c) \supset \texttt{IsSignatory}(d,A)$.

A3: $\texttt{IsValidAuthority}(\texttt{VA},\texttt{vc})$.

A4: $\forall A,c$: $\texttt{HasCert}(A,c) \wedge \texttt{IsValidCert}(c) \supset \texttt{IsValidAuthority}(A,c)$.

Notice that Axiom A4 asserts that *anyone* with a valid certificate is a valid authority. In a more detailed account, certificates need to include a list of specific capacities that are granted by the certificate, which will then allow some certificates to empower their owners to sign other certificates, as well as documents, and others to only sign documents. This feature of certificates has been omitted from this example in the interest of simplicity.

Some key objects which are referenced by name in Table III are listed, with their explanations, in Table V. When the informal statements listed in Table III are stated in formal logic we need to use certain *predicates*, the definitions of which are provided in Table IV.

D. A proof

Let us now review the proofs of P1, P2, P3 and O1. Note that although these proof are conceptually simple, actually finding them and including all details is not so easy, and the proofs given here required several iterations, due to the discovery of logical flaws present in the first attempts. Such logical flaws can be the basis for attacks if they are not discovered by software developers. In cases where the need for absolute rigour is paramount, finding all the details required for a proof to be algorithmically verified may be warranted.

Table V: Names of objects and entities which appear in the proof that Joan is the signatory of the document

Name	Description
CA	Certisign digital certificate authority
VA	Verisign
cc	The certificate owned by Certisign which authorizes them as a certificate signing authority
jc	Joan's certificate
ds	The signature on Joan's document
jdoc	The document of Joan

Finding the proofs of all the propositions and objectives in this example was achieved with the aid of the inference graph of their relationships.

Proof of P1

To prove that a certificate is a valid, we can use Axiom A1 with VA in place of A, cc in place of c, vc in place of ac, and cs in place of s. The process of generating a specific version of an axiom in this way is a feature of the predicate calculus which is needed here and explains why we need the predicate calculus, not just the propositional calculus, to correctly model the logic of this system. With these substitutions, Axiom A1 becomes:

$$\texttt{IsValidAuthority}(\texttt{VA},\texttt{vc}) \wedge \texttt{HasSig}(\texttt{cc},\texttt{cs}) \wedge \texttt{IsValidAccTo}(\texttt{cs},\texttt{vc},\texttt{VA}) \supset \texttt{IsValidCert}(\texttt{cc}). \quad (1)$$

The antecedent conditions of this specialized axiom are A3, E7 and E8. Two of these conditions are *enforced*, and the other is an axiom. Hence the conclusion of (1) follows, which is what we wished to prove.

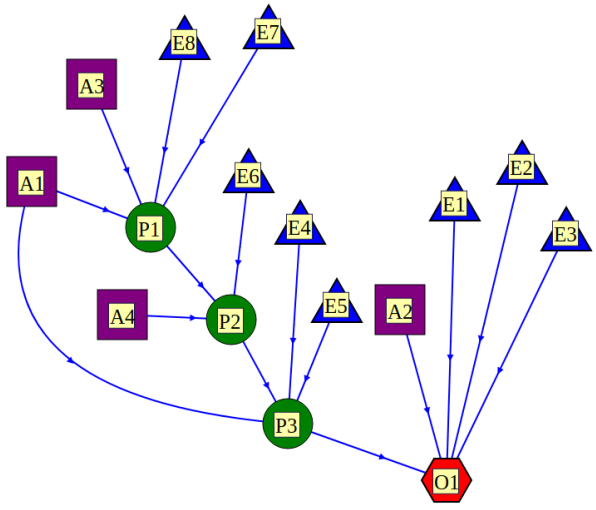

Figure 4: Graph of cybersecurity rules and their interdependencies for proof that Joan is a signatory of document d

Proof of P2

This follows from A4, with P1 and E6 to confirm the antedent conditions.

Proof of P3

To prove that certificate jc is a valid, we can use Axiom A1 again this time with CA in place of A, jc in place of c, cc in place of ac, and js in place of s. With these substitutions, Axiom A1 becomes:

$$\texttt{IsValidAuthority}(\mathbf{CA},\mathbf{cc}) \wedge \texttt{HasSig}(\mathbf{jc},\mathbf{js}) \wedge \texttt{IsValidAccTo}(\mathbf{js},\mathbf{cc},\mathbf{CA}) \supset \texttt{IsValidCert}(\mathbf{jc}).$$

The antecedent conditions of this specialized axiom are P2, E7 and E8. Two of these conditions are *enforced*, and the other is an axiom. Hence the conclusion of (1) follows, which is what we wished to prove.

Proof of O1

To prove that an author is a signatory, we can use Axiom A2. In the present instance, O1 follows from A2 with Joan in place of A, jdoc in place of d, ds in place of s, and jc in place of c. The process of generating a specific version of an axiom in this way is a feature of the predicate calculus which is needed here and explains why we need the predicated calculus, not just the propositional calculus, to correctly model the logic of this system. With these substitutions, the axiom becomes:

$$\texttt{HasCert}(\mathbf{Joan},\mathbf{jc}) \wedge \texttt{HasSig}(\mathbf{jdoc},\mathbf{ds}) \wedge \texttt{IsValidCert}(\mathbf{jc}) \wedge \texttt{IsValidAccTo}(\mathbf{ds},\mathbf{jdoc},\mathbf{jc}) \supset \texttt{IsSignatory}(\mathbf{jdoc},\mathbf{Joan}).$$

The antecedent conditions of this specialized axiom are P3, E1, E2 and E3. Three of these conditions are *enforced*, and P3 has already been proved, above. Hence the conclusion of this axiom follows, which is what we wished to prove.

IV. Software

Two scripts have been developed: *latex2netml* and *netml2latex*. These convert between two representations of a set of rules, in LaTeX, and in xml. Fig. 5 shows how a rule is represented in LaTeX, and Fig. 6 shows the form this same rule takes when it has been translated into xml form. Part of the main script implementing *latex2netml* is shown in the appendix.

The LaTeX representation is suitable for textual representation of rules and their proofs. The Netml representation allows a cybersecurity rule-set to be represented and edited graphically. Scripts for conversion in both directions allow both these representations to be used, in appropriate situations.

The Netml server (https://netml.org/netml4_69) can be used as a way to access these scripts and to graphically edit inference graphs. When a file is loaded, if the filename has the extension .tex, the latex2netml will be applied to the file and the resulting xml file is then loaded and displayed. The Netml user-interface can then be used to modify the layout of the inference graph. The inference graph can then be saved either as an xml file or as a tex file.

```
\begin{objective}s\label{O3}
Deliverers are able to store delivered goods in
 ↪  the box whenever they visit the
property with an item to be delivered
\end{objective}
\begin{proof}
\ref{E3}\ref{E6}\ref{E7}
\end{proof}
```

Figure 5: Representation of a rule in LaTeX

```
<node id='O3' type='objective'>
    <label>O3</label>
    <name>Objective O3</name>
    <dependson>E3</dependson>
    <dependson>E6</dependson>
    <dependson>E7</dependson>
    <xposition>642</xposition>
    <yposition>325</yposition>
    <statement>
Deliverers are able to store delivered goods in
 ↪  the box whenever
they visit theproperty with an item to be
 ↪  delivered
    </statement>
    <proof>
                \ref{E3}\ref{E6}\ref{E7}
    </proof>
</node>
```

Figure 6: Representation of a rule in xml

The latex2netml script is able to take either *just* a tex file, or both an xml file *and* a tex file. In the former case the positions of the nodes in the xml file will be chosen by an algorithm which attempts to separate nodes from each other in an effective way. In the latter case, the xml file is used as a guide for where nodes should be placed. If a file of the same name is loaded into the public server, the existing xml file, in the user's repository, will be used to determine the node positions.

The shape of the nodes, and their colour, can also be used, as we have seen in several examples, in Section III, to represent inference relationships between rules more clearly.

V. Conclusion

The proofs required in most cybersecurity problems are not mysterious. It is often obvious how the statement of a rule follows from the rules on which it depends. The challenge in creating systems which are rigorously secure is more an issue of *clarity* and *attention to detail*. Rigorous proof requires much greater *precision* of detail than in normal speech or writing.

It is often sufficient to merely indicate which rules are required to prove an objective or condition, for the proof to be obvious. Alternative designs with different assumptions should be considered, to find the correct balance between security and convenience. In some cases, despite their simplicity, precisely specifying the details of a proof can be

challenging, and an inference graph helps to achieve the necessary clarity. In all such cases, an inference graph can be used to help find and to graphically illustrate the proofs, once they have been found.

Inference graphs, as defined and illustrated in this paper, provide a way to visualise and validate cybersecurity rules and their relationships, in a way which can be very helpful in cybersecurity architecture and design. Publicly available tools for inference graphs have been developed.

References

[1] Ron Addie, Stephen Braithwaite, and Abdulla Zareer. Netml: a language and website for collaborative work on networks and their algorithms. In *Proceedings of the Australian Telecommunication Networks and Applications Conference (ATNAC 2006)*, pages 1–5. University of Melbourne, 2006.

[2] Ronald G Addie, Yu Peng, and Moshe Zukerman. Netml: networking networks. In *Dependable, Autonomic and Secure Computing (DASC), 2011 IEEE Ninth International Conference on*, pages 1055–1060. IEEE, 2011.

[3] Yuli Deng, Duo Lu, Dijiang Huang, Chun-Jen Chung, and Fanjie Lin. Knowledge graph based learning guidance for cybersecurity hands-on labs. In *Proceedings of the ACM Conference on Global Computing Education*, pages 194–200. ACM, 2019.

[4] Luc Engelen and Mark van den Brand. Integrating textual and graphical modelling languages. *Electronic Notes in Theoretical Computer Science*, 253(7):105–120, 2010.

[5] Nirnimesh Ghose, Loukas Lazos, Jerzy Rozenblit, and Ronald Breiger. Multimodal graph analysis of cyber attacks. In *2019 Spring Simulation Conference (SpringSim)*, pages 1–12. IEEE, 2019.

[6] Nabeel Hadaad, Luke Drury, and Ron Addie. Protecting services from security mis-configuration. In *International Telecommunication Networks and Applications Conference (ITNAC)*, 2015.

[7] Project Management Institute. A guide to the project management body of knowledge (pmbok® guide)-(simplified chinese). Project Management Institute, 2018.

[8] Yan Jia, Yulu Qi, Huaijun Shang, Rong Jiang, and Aiping Li. A practical approach to constructing a knowledge graph for cybersecurity. *Engineering*, 4(1):53–60, 2018.

[9] Aditya Pingle, Aritran Piplai, Sudip Mittal, and Anupam Joshi. Relext: Relation extraction using deep learning approaches for cybersecurity knowledge graph improvement. *arXiv preprint arXiv:1905.02497*, 2019.

[10] Neil Rerup. *Hands-on cybersecurity for architects : plan and design robust security architectures*. Packt Publishing, Birmingham, UK, 2018.

[11] Kenneth H Rose. A guide to the project management body of knowledge (pmbok® guide)—fifth edition. *Project management journal*, 44(3):e1–e1, 2013.

[12] Dawood Sheniar, Nabeel Hadaad, David Martin, Ron Addie, and Shahab Abdullah. Experiments and proofs in web-service security. In *International Telecommunication Networks and Applications Conference (ITNAC)*, 2018.

Appendix

The main script `latex2netml` which converts LaTeX files to Netml files is shown in Listing 1.

```
#!/bin/bash
if [[ "$1" == "-h" ]]
then
        echo "latex2netml [ -l] [-o outputfile] [-p
        ↪  directoryForPositionXMLFile] file1.tex
        ↪  file2.tex ..."
    echo "-l to use labels rather than ids to label nodes"
        exit 0
fi
if [[ "$1" == "-l" ]]
then shift
        # echo "latex2netml.pl -f $* \; -l >
        ↪  ~/tmp/tmp.xml"
        if [[ "$1" == "-o" ]]
        then shift
                outfile=$1
                shift
        else
                outfile="none"
        fi
        if [[ "$1" == "-p" ]]
        then shift
                positiondir=$1
                shift
        else
                positiondir="none"
        fi
        if [[ $positiondir == "none" ]]
        then
        latex2netml.pl -f $* \; -l -o /tmp/tmp.xml
        else
        # echo "latex2netml.pl -f $* \; -l -o
        ↪  /tmp/tmp.xml -p \"$positiondir\" "
        latex2netml.pl -f $* \; -l -o /tmp/tmp.xml -p
        ↪  "$positiondir"
        fi
else # echo "latex2netml.pl -f $* \; >
↪  ~/tmp/tmp.xml"
        if [[ "$1" == "-o" ]]
        then shift
                outfile=$1
                shift
        else
                outfile="none"
        fi
        if [[ "$1" == "-p" ]]
        then shift
                positiondir=$1
                shift
        else
                positiondir="none"
        fi
        if [[ $positiondir == "none" ]]
        then
        latex2netml.pl -f $* \; -o /tmp/tmp.xml
        else
        latex2netml.pl -f $* \; -o /tmp/tmp.xml -p
        ↪  "$positiondir"
        fi
fi
xsltproc /usr/bin/addlinks.xsl /tmp/tmp.xml > /tmp/tmp2.xml
xsltproc /usr/bin/addlinks.xsl /tmp/tmp2.xml >
↪  /tmp/tmp3.xml
xsltproc /usr/bin/addlinks.xsl /tmp/tmp3.xml >
↪  /tmp/tmp4.xml
xsltproc /usr/bin/addlinks.xsl /tmp/tmp4.xml >
↪  /tmp/tmp5.xml

if [[ "$outfile" == "none" ]]
then xsltproc /usr/bin/addlinks.xsl /tmp/tmp5.xml
else xsltproc /usr/bin/addlinks.xsl /tmp/tmp5.xml >
↪  $outfile
fi
```

Listing 1: `latex2netml` main script

This script invokes a Perl script to convert the LaTeX files to an `xml` file. This `xml` file is then processed by the xslt script `addlinks.xsl`.