Design Procedure of Knowledge Base for Practical Attack Graph Generation

Masaki Inokuchi, Yoshinobu Ohta,
Shunichi Kinoshita, Tomohiko Yagyu
Security Research Laboratories, NEC Corporation
{m-inokuchi@bu,y-ohta@cj,s-kinoshita@jy,yagyu@cp}
.jp.nec.com

Orly Stan, Ron Bitton, Yuval Elovici,
Asaf Shabtai
Department of Software and Information Systems
Engineering, Ben-Gurion University of the Negev
{stan,ronbit}@post.bgu.ac.il,{elovici,shabtaia}@bgu.ac.il

ABSTRACT

Cyber security assessment is an essential activity for understanding the security risks in an enterprise environment. While many tools have been developed in order to evaluate the security risks for individual hosts, it is still a challenge to identify multi-hop cyber security risks in a large-scale environment. An attack graph, which provides a comprehensive view of attacks, assists in identifying high-risk attack paths and efficiently deploying countermeasures. Several frameworks which generate an attack graph from system information and knowledge base have also been developed in the past. Although these tools are widely adopted, their expression capabilities are insufficient. The expansion of knowledge base is needed to handle comprehensive attack scenario.

In this research, we developed an attack graph generation system by extending the MulVAL framework which is widely adopted due to its high extensibility. We designed and implemented knowledge base (also known as "interaction rules" in the MulVAL framework) for practical attack graph generation. A structured design procedure is necessary to construct a knowledge base that enables comprehensive analysis, which is highly important for actual risk assessment. We describe the design procedure, design considerations and implementation of our rule set. Additionally, we demonstrate the improvement to the generated attack graph by the implemented rules in a case study.

KEYWORDS

Risk assessment, Attack graph, Knowledge base, Design procedure

ACM Reference Format:
Masaki Inokuchi, Yoshinobu Ohta,, Shunichi Kinoshita, Tomohiko Yagyu and Orly Stan, Ron Bitton, Yuval Elovici,, Asaf Shabtai. 2019. Design Procedure of Knowledge Base for Practical Attack Graph Generation. In *ACM Asia Conference on Computer and Communications Security (AsiaCCS '19), July 9–12, 2019, Auckland, New Zealand*. ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3321705.3329853

AsiaCCS '19, July 9–12, 2019, Auckland, New Zealand

ACM ISBN 978-1-4503-6752-3/19/07…$15.00
https://doi.org/10.1145/3321705.3329853

1 INTRODUCTION

Cyber security assessment including penetration tests are essential to keep enterprise networks secure. Current penetration tests and other security assessment methods are mainly performed manually. Some existing tools assist security experts in collecting system information [19]. However, recent cyber attacks are increasingly sophisticated. Attackers perform multi-step attacks against multiple hosts while using various attack methods. Therefore, security experts are required to invest a great deal of effort to enumerate the entire cyber security risks in a large-scale environment and determine the impact of software vulnerabilities.

An attack graph is a promising platform to draw the comprehensive attack paths existing in an organization network. A logical attack graph visualizes attack steps and prerequisites for each attack action. The automatic construction of an attack graph enables us to achieve continuous and comprehensive assessment for a large scale enterprise environment. MulVAL, a logic-based network security analyzer [12][11], is a widely adopted framework for constructing an attack graph. The MulVAL framework infers the attack graph from an initial attacker location to an attack goal from collected target system information on the basis of interaction rules (also regarded as knowledge base).

However, designing and creating these interaction rules is not a trivial task. Attack graphs for security assessment require high comprehensiveness. MulVAL's default rules have limited expression capability and model only a few specific attack scenarios. There are some existing works extending MulVAL to improve expression capability [1][4]. However, they are still insufficient in practice, and it is still necessary to design an extended rule set for practical security assessment. Additionally, existing works lack any discussion about the design procedure of the knowledge base for attack graph generation; they just show the model to express specific attack scenarios or specific system entities. Even though attacker carry out various attack actions, the scope of attack graph analysis is not specified clearly. Security experts must know what attack actions can be listed up by the assessment in actual security assessment situations. Therefore, the interaction rules for attack graph generation must be designed with a panoramic view of the attacker's entire behavior and clearly defined its expression capability.

We designed and implemented attack graph generation rules for practical security assessment. In this paper, we describe the design procedure, design considerations and implementation of our rule set. Our design procedure consists of mainly five steps: 1) scope definition, 2) granularity definition, 3) actions enumeration, 4) predicates definition, and 5) rules creation. We define the analysis

scope on the basis of adversary behavior categories [8] in the first step. Generally, the analysis scope is specified in an abstract way (e.g. lateral movement, credential access, etc.). We show how interaction rules can be materialized from the abstract scope in next four steps (step 2-5). In the second step, we define the expression granularity. Next, we enumerate attack actions that should be expressed in the attack graph. In the next two steps, we design and implement predicates and rules based on the results of the previous steps.

We conducted a case study to show the performance of our implemented rules. We demonstrate the improvement of the generated attack graph in comparison to an attack graph generated from MulVAL default rules. Although our rule set is a significant improvement over MulVAL's default rules in some respects, the main contribution of this paper is to present our sophisticated design procedure itself. To the best of our knowledge, no other work outlines a systematic design procedure for creating rules for attack graph generation. This work provides a practical guide for designing attack graph generation rules.

2 RELATED WORKS

2.1 Attack Graphs

Since an attack graph visualizes attack paths using a combination of attack actions, we can utilize it to enumerate high-risk attack paths and plan efficient countermeasures. Phillips and Swiler et al. [13][18], were the first to introduce the idea of the graph-based network vulnerability analysis and presented an automated tool for generating attack graphs. Although a wide variety of attack graphs are proposed since then, those that have an AND/OR logical structure are the most popular type [5] due to their high readability.

Scoring the attack graph nodes enable us to find high-risk attack paths and plan efficient countermeasures. Bayesian network techniques [10][9], which are originally developed in the artificial intelligence domain, are widely used to score attack graph nodes [15][14][20].

2.2 Logic based attack graph generation and MulVAL

MulVAL, a logic-based network security analyzer [12][11], is the widely adopted framework for constructing attack graphs. MulVAL automatically generates an attack graph by deductive reasoning. It uses two files to do this: an input file and an interaction rule file. The input file specifies the attacker and system state including the attacker's initial location, attacker's goal, host and network configuration, and existing vulnerabilities. Each fact that appear in the MulVAL inference is specified as a predicate. There are two types of predicates handled by the MulVAL framework: primitive and derived. All predicates in the input file are considered primitive. Derived predicates are inferred from a combination of other predicates, and are expressed by interaction rules. MulVAL comes with a default interaction rule file[1]. If users don't specify their own interaction rules, the default interaction rule file is used. Samples of interaction rules, input file, and corresponding output are shown in Appendix A.

[1]Released at http://www.arguslab.org/software/mulval.html, (accessed 2018-10-22).

2.3 MulVAL extensions

MulVAL is widely adopted to model various systems and extended for their objective. Liu et al. [6] used MulVAL to represent forensic evidence. Sembiring et al. [17] proposed Bayesian attack graph analysis using a graph generated by MulVAL. Mavani et al. [7] modeled 6LoWPAN networks. Although MulVAL is a very useful tool, its current expression capability is limited and not sufficient for practical purposes. Jing et al. [4] extended MulVAL's interaction rules to handle detailed vulnerability information. They focused on six types of vulnerabilities (DoS, ExecCode, Overflow, Bypass, +Info, +Priv) having different consequences, as opposed to MulVAL's default rule set that handles only one vulnerability type (gaining privilege). Although their model enhances the information on the generated attack graph, it doesn't fully utilize the expression capability of the vulnerability type. In a multi-step attack scenario, each step relies on the consequences of its preceding steps. Therefore, because of the limited expression of vulnerability exploitation consequence in the modeling of Jing et al., there might be attack scenarios that will not be fully captured. Other works have introduced new entities of attacks that the MulVAL framework does not consider. For example, Acosta et al. [1] modeled network attacks and proposed a model to handle ARP spoofing and route hijack. Saha [16] extended to include the more complex security policies that exist in advanced operating systems. Bacic et al. [2] modeled network components and assets. They modeled a router and a route entry as network components. They also modeled asset values and classifications. Froh et al. [3] extended the work of [2] to model high-level mission-centric IT services.

As mentioned above, there are various MulVAL extensions. However, they only show modeling for specific system entities or attacks, and their design procedures are unclear. They don't explicitly specify their scope of analysis in terms of attacker's behavior, although the attacker performs various action from reconnaissance to actions on objective in cyber kill chain at various entities. Therefore, it is difficult to determine whether they are suitable for specific purpose and how they can be extended. Moreover, for security assessment purposes, the assessment capability should be clearly defined because it is a key factor in the system operator's decision of whether to perform the security assessment.

3 RULE DESIGN PROCEDURE

3.1 Assumption of assessment

Attack graph generation rules should be designed with the assessment objective in mind. Figure 1 shows the outline of the attack graph-based security assessment that we try to implement, for a given initial location and goal of the attacker. The main objective of the attack graph-based assessment is to figure out all of the attack paths from the attacker's initial location to the attack goal and all of the prerequisites for the attack actions included in these paths. Such information can be used to determine high-risk paths or plan efficient countermeasures.

3.2 Overall Procedure

Figure 2 presents an overview of the design process. First, we **define the analysis scope** (scope definition step). In this step, we

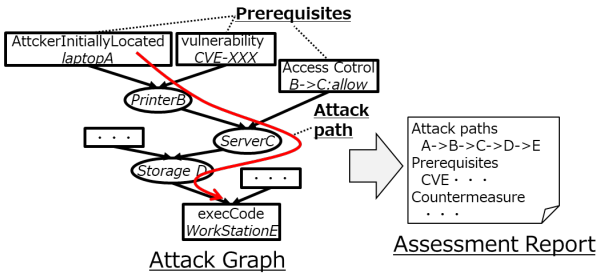

Figure 1: Attack graph-based security assessment

adopt the attack action categories defined in ATT&CK framework [8] for the scope definition. Next, in the granularity definition step, we **define the attacker states and facts, and how detailed they should be expressed**. The expression capability of the attack graph is determined by the scope definition and granularity definition steps. In the action enumeration step, we **list attack actions** that can be taken by the attacker to gain his/her next state. This step is important to reduce attack paths oversight. In the next step, the **predicates are defined** according to the enumerated attack actions. The **rule set is then implemented** in the rule creation step.

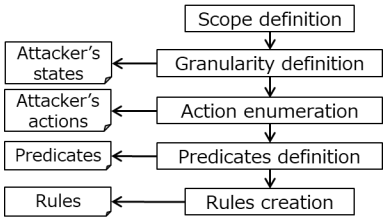

Figure 2: Overview of rule design process

3.3 Scope definition

In this step, the scope of attack action are defined. We define the scope of attacker's actions based on the 11 attacker behavior categories defined by the ATT&CK framework [8].

Attackers take various actions in order to achieve their goal. Some actions are essential, such as privilege escalation and lateral movement, while other actions (e.g. persistence, discovery, etc.) just reduce the future attack effort or increase the probability of attack success. The former actions should be expressed in the attack graph in order to realize attack paths. We do not model the latter actions because they include actions that are not directly related to attack success and contain too many states to be expressed in the attack graph. We determine the four categories of behavior as our scope, i.e., *Privilege Escalation*, *Credential Access*, *Lateral Movement*, and *Execution*.

3.4 Granularity definition

The granularity of an attack graph should be clearly defined. A fine-grained attack graph can express detailed attack steps and conditions, but the graph size and number of rules increases. It is difficult to implement rules that are too fine-grained because there is a tremendous amount of attack actions. Additionally, too fine-grained states require more extensive data collection process which sometimes can be infeasible. Attack actions in the attack graph are expressed by combination of pre-exploitation state and post-exploitation state. Therefore, granularity can be defined by the attacker's states. We define the attacker's states specified in generated attack graph as shown in Table1.

Table 1: Attacker's states expressed in the attack graph

Attacker's state		No.
Host access	Network access (remote access)	S1
	Login (local access)	S2
	Data injection	S3
Privilege	Local user	S4
	Administrator	S5
File access	Write	S6
	Read	S7
Gain credential		S8

We define three states for accessing a host: remote access, local access, and data injection. Remote access means that the attacker can send a signal to the host via the network without the disturbance of firewall and other defense mechanisms. Local access means that the attacker can log in to the host and execute arbitrary code with the log in user permission. Note that attackers can gain this state not only through legitimate log in, they also can gain a local access state in other ways, e.g., by exploiting a network service vulnerability. Data injection means that the attacker can inject data through various services or applications. For example, if the attacker can send an e-mail with an attachment, he/she obtains the data injection state. The current MulVAL default rule set does not include data injection state in an abstract way like this, and the data injection state is represented by NFS-specific dataflow and http-specific dataflow. They are indirectly expressed by the "nfsMouonted" and the "nfsExport" predicates, and the "accessMaliciousInput" predicate respectively. This way of modeling requires many predicates and rules to represent a specific software or service. Therefore, our approach is to abstract this dataflow. Local user permission and administrator permission are distinguished in the case of local access and are represented by two different states. Two types of file access are defined: write, and read. The state that the attacker knows the credentials is different from local access. In some cases, the attacker have credential information but may not be able to access the host. We define the "gain credential" state to distinguish this scenario.

These attacker's states should be designed such that attacker's behavior defined in Subsection 3.3 can be expressed. *Lateral movement* are expressed by the host access state. Similarly, login user privilege and file access states can be used to express *privilege escalation*. Whether an attacker gains credential or not, corresponds to *credential access*. Additionally, the prerequisites of malicious code *execution* can be indicated by a combination of these states.

3.5 Action enumeration

We can identify attacker's behavior which should appear in the generated attack graph by listing up the actions to change attacker's states, as shown in Table 1. Table 2 lists the typical attack actions corresponding to each state. To express attack actions, the corresponding rules should express the relation between pre-exploitation

states and post-exploitation states. Therefore, each action should include at least one attacker state as pre-exploitation state (denoted by underline).

We enumerate the attack actions for each attacker state. Breaking down the attacker's actions for defining the states can prevent overlooking potential attack actions and consequently improves the comprehensiveness of the rule set.

Table 2: Attacker's actions.

Target state	Action	No.
Remote access (S1)	Obtain local access to a host neighboring the target host	A1
Local access (S2)	Obtain legitimate user credential and log in through a legitimate login service	A2
	Exploit a remote code execution vulnerability	A3
	Make a legitimate user run injected malicious code	A4
	Make a legitimate user open an injected malicious file by a software with code execution vulnerability	A5
	Make a legitimate process run injected malicious code	A6
	Make a legitimate process with code execution vulnerability open an injected malicious file	A7
Data injection (S3)	Obtain local access to a host that can send any data to the target host	A8
	Obtain *write* file access to a host that can send any data to the target host	A9
Administrator or local user privilege (S4, S5)	Exploit privilege escalation vulnerability after login or from remote	A10
File access (S6, S7)	Obtain local access to the host that contains the target file	A11
	Exploit a vulnerability to enable the attacker to access a file after login or from remote	A12
Gain credential (S8)	Obtain file(or memory) access including credential information	A13

3.6 Predicate definition

We define predicates to specify the attacker's states defined in Subsection 3.4 and the attacker's action enumerated in Subsection 3.5. The predicates corresponding to the states in Table 1 are defined in Table 3.

Table 3: Predicates related to attacker's state

State no.	Predicate / description
S1	netAccess(principal, srcHost, dstHost, protocol, port) *principal* can access *dstHost* from *srcHost* using the *protocol* and the *port*.
S2, S4	localAccess(principal, host, user) *principal* can login to the *host* as the *user*.
S3	dataInject(principal, host, path) *principal* can inject data to *path* on *host*.
S5	isAdmin(adminUser,host) Account *adminUser* on *host* has administrator privilege. This predicate is used together with the *localAccess* predicate.
S6, S7	accessFile(principal, host, access, path) *principal* can access data at *path* on *host* with *access* privilege.
S8	hasAccount(principal, host, user) *principal* has the credential of account *user* on *host*.

Next, we define predicates to express the attack actions shown in Table 2. We observe previous work[12][11] and find that there are three types of facts that should be expressed other than the attacker state: human state (legitimate user), system state, and knowledge that does not depend on the assessed system (e.g. vulnerability and software functionality). On the basis of this observation, we extract the items from Table 2 that should be expressed in the attack graph, as shown in Table 4.

Human characteristic (I1) is an item that distinguishes persons' characteristics, e.g. "this person is an attacker who has malicious intentions". Host connectivity (I2) information is needed to identify the hosts that are reachable from a possible attacker position. Existing software (I3) and vulnerability information (I8) should be expressed in order to infer the scenarios where an attacker exploits software vulnerabilities. Dataflow (I4), file access permission (I5), data location (I6) and data type such as credential (I7) should be expressed in order to model attacks on data or attacks through dataflow such as data injection. Each software and service can have various functionalities. For example, SSH provides a login function. If an implementation of SSH has the vulnerability of authentication avoidance, it may allow an attacker to log in to the host without credential. Therefore, we need to model the functionalities of software and services (I9).

Table 4: Facts to be expressed in the attack graph.

Category	Items	No.
Human state	Human characteristics	I1
System state	Host connectivity (logical connectivity controlled by firewall and physical connectivity)	I2
	Installed software (running service)	I3
	Dataflow	I4
	File access permission	I5
	Data location	I6
	Data type	I7
Knowledge	Vulnerability	I8
	Functionality of software	I9

We define predicates for each of the items in Table 4 as shown in Table 5. In some cases, the possible values of predicates' arguments should be predefined. In particular, the consequence of vulnerability exploitation (*consequence* in I8) directly affects the expression capability of the attack graph. As mentioned in [4], the variation of the *consequence* argument in MulVAL's default rules is not sufficient for practical use. This argument can also be designed on the basis of how vulnerabilities are exploited (as defined in Table 2), as presented in Table 6. The possible values of this argument should correspond to the attacker's states listed in Table 1. This correspondence is an important feature in the design process. The authors in [4] also extended the vulnerability's consequences argument but they didn't mention the relation between the consequences and next attack action. Therefore, attack actions cannot be inferred from their extended consequences. In contrast, we define the consequences to match the attacker states which are designed as the prerequisites of attack actions.

3.7 Rule creation

In this step, the relations among the predicates defined in Subsection 3.6 are implemented as rules. We implement rules in order to infer the attack behaviors shown in Table 2; following is an example of the implemented rules.

Listing 1 reflects attack action A1. This rule infers that the attacker can remotely access a host after obtaining local access to its neighboring host. Descriptions of other representative rules are shown in Appendix B.

Table 5: Predicates related to legitimate user, system, and knowledge

Item no.	Predicates / descriptions
I1	-malicious(principal) *Principal* is an attacker. -inCompetent(principal) *Principal* is incompetent and can be taken advantage of by attackers.
I2	- located(host, subnet) *Host* is located in *subnet.* - fwAllowed(firewall, hostA, hostB, protocol, port) *Firewall* allows communication between *hostA* and *hostB* using *protocol* and *port.* -aclNW(srcHost, dstHost, protocol, port) Communication using *protocol* and *port* from *srcHost* to *dstHost* is not blocked by network firewall. Note that *srcHost* and *dstHost* can also represent subnet.
I3	- localService(host,program,user) *Program* runs on a *host* with *user* privilege. - networkService(host, program, protocol, port, user) *Program* runs on a *host* with *user* privilege and listens to *protocol* and *port.* - dataSharingService(host, program, dataflow, user) *Program* runs on a *host* with *user* privilege and sends or receives *dataflow*
I4	- dataFlow(srcHost, dstHost, flowname) There is a dataflow *flowname* from srcHost to dstHost.
I5	- localFileProtection(host, user, access, path) Access permission of *path* is set as *access* for *user* on the *host.* (*access* can be read, write or execute)
I6	-dataBind(data, host, path) *Data* is located at *path* on the *host.*
I7	-isCredential(data, host, user) *Data* is the credential of *user* on *host.* *This predicate is used together with the *dataBind* predicate.
I8	-vulHost(host, vullD, program, range, consequence) *Program* running on a *host* has the vulnerability *vulID* that can be exploited from *range* and cause *consequence*
I9	- isLoginService(program) *Program* provides the functionality to log in to the host.

Table 6: Consequences of vulnerability exploitation

Consequence	Description
privEsc	The attacker can execute arbitrary code with administrator privilege. This consequence corresponds to the attacker's state of local access with administrator privilege.
execCode	The attacker can execute arbitrary code with the same privilege of the exploited software/service. This consequence corresponds to the attacker's state of local access.
dataTheft	The attacker can read arbitrary data stored in the host. This consequence corresponds to the attacker's state of file access with read permission.
dataTamper	The attacker can modify or locate arbitrary data stored in the host. This consequence corresponds to the attacker's state of file access with write permission.

```
netAccess(Principal, SrcHost, DstHost, Prot, Port):-
    localAccess(Principal, SrcHost, SrcUser),
    aclNW(SrcHost, DstHost, Prot, Port)
```

Listing 1: Remote access from neighbor (A1).

4 CASE STUDY

4.1 Case study environment

We conducted a case study in order to demonstrate the usefulness of our implemented rules. The case study environment is shown in Figure 3. We used a similar environment to that used for testing MulVAL[12], which includes three hosts: webServer, fileServer, and workStation. Initially, the attacker is located on a host in the Internet. The webServer is located in the DMZ and connects to the Internet via a firewall that filters the communication between them. The fileServer shares files with the webServer and workStation through the NFS protocol. The webServer has write permission to the shared directory ('/export') on the fileServer. The shared directory ('/export') is also shared with the shared directory ('/user/local/share') on the workStation. Both the webServer and fileServer have a vulnerability that can be remotely exploited and makes it possible for the attacker to escalate privileges. The attacker's objective is to take over the workStation. If the attacker can inject malicious files to the workStation and make a legitimate user open them, the attacker can execute arbitrary code in the workStation.

To demonstrate improvement of our rule set, we assume one additional vulnerability in the fileServer that can be remotely exploited and makes it possible for the attacker to gain file access (*write*) to an arbitrary location.

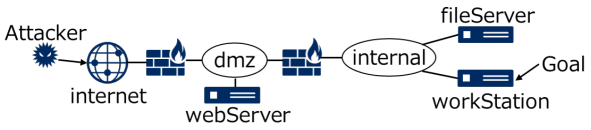

Figure 3: Case study environment.

4.2 Comparison of implemented rules and MulVAL's default rules

Figure 4 shows the attack graph generated by MulVAL's default rules. The Corresponding nodes description are shown in Appendix C. This attack graph presents two possible attack paths from node 13 to node 5. In the first attack path (the right-side path, going through node 23), the attacker injects malicious code to the workStation (node 3) via the shared directory on the fileServer (node 5) right after exploiting the vulnerability in webServer (node 13). In the second attack path, the attacker exploits a vulnerability in the fileServer (node 8). After that, the attacker injects malicious code to the workStation in the same way as the previous attack path.

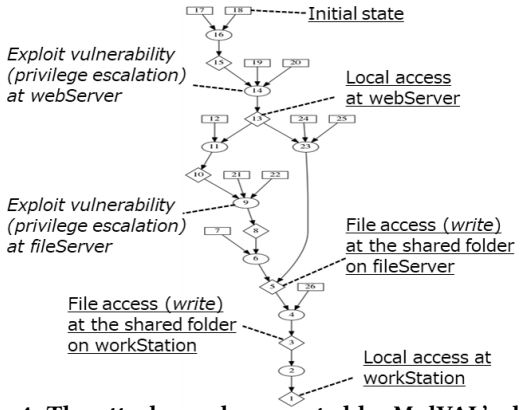

Figure 4: The attack graph generated by MulVAL's default rules.

Figure 5 shows the attack graph generated by our implemented rules that were designed according to the proposed procedure. The corresponding nodes description are also shown in Appendix C.

Three possible attack paths appear from node 18 to node 13; two of them (going through nodes 38 and 33) represent the same attack scenarios that appeared in Figure 4, one of them (going through node 14) only appears in this attack graph. We can observe the following improvements from the comparison between these two results:

- Our rule set successfully infers an additional attack path in which a vulnerability allows the attacker to gain file access (*write*). This is because we clearly defined the relation between consequences and attacker's state in design procedure (Table 6).
- Although we model dataflow in a general way, our generalized dataflow-related rule successfully inferred the attack paths same as MulVAL's default rule. Our model can be used for other dataflow types (e.g. SMB, e-mail, etc.) in contrast to the NFS-specific modeling in MulVAL.
- The attack graph in Figure 5 expresses the attack scenario more clearly than the one in Figure 4. For example, MulVAL infers the execCode predicate at the workStation (node 1 in Figure 4) only from the file access on the workStation (node3 in Figure 4). This is because the rule corresponding to node 2 in Figure 4 is too generic. Code execution requires additional conditions (e.g. user interaction) in general. On the other hand, our rule set infers the execCode predicate at the workStation (node 1 in Figure 5) from four other nodes (nodes 3, 4, 10, and 11 in Figure 5). Nodes 3 and 4 in Figure 5 explicitly indicate on legitimate user interaction. Considering the listed attacker's action (Table 2) in the attack enumeration step enabled the identification of three different attacker actions (A5, A6, and A7).

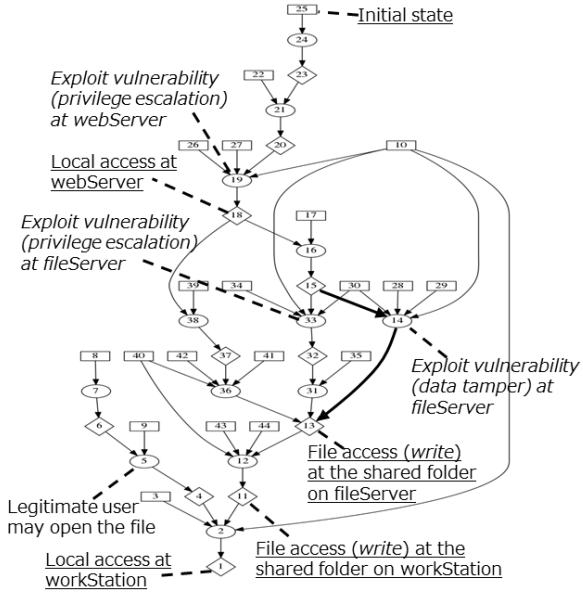

Figure 5: The attack graph generated by our rules.

5 CONCLUSION

In this paper, we describe the design procedure, considerations and implementation of an extended rule set for attack graph generation. We present in detail a design procedure that consists of five steps: 1) scope definition, 2) granularity definition, 3) action enumeration 4) predicate definition, and 5) rule creation. We determine the expression capability of the generated attack graph in the scope definition and granularity definition steps. The expression capability must be clearly determined for practical use in order to enable security expert to decide whether to perform a security assessment. Predicates and rules are implemented in the predicate definition and rule creation steps. We evaluate the improvement of our modeled rules through a case study. The evaluation indicates that 1) the rule set designed by our procedure can infer attack scenarios in which the attacker exploits various types of vulnerabilities other than privilege escalation, 2) we successfully generalize dataflow-related rules, and 3) our rule set can clearly express attack scenarios due to the adequate enumeration of attack actions.

REFERENCES

[1] Jaime C Acosta, Edgar Padilla, and John Homer. 2016. Augmenting attack graphs to represent data link and network layer vulnerabilities. In *Military Communications Conference, MILCOM 2016-2016 IEEE*. IEEE, 1010–1015.

[2] Eugen Bacic, Michael Froh, and Glen Henderson. 2006. *Mulval extensions for dynamic asset protection.* Technical Report. CINNABAR NETWORKS INC OTTAWA (ONTARIO).

[3] Michael John Froh and Glen Henderson. 2009. *MulVAL extensions II.* Defence R&D Canada-Ottawa.

[4] James Tan Wee Jing, Lim Wee Yong, Dinil Mon Divakaran, and Vrizlynn LL Thing. 2017. Augmenting MulVAL with automated extraction of vulnerabilities descriptions. In *Region 10 Conference, TENCON 2017-2017 IEEE*. IEEE, 476–481.

[5] Barbara Kordy, Ludovic Pietre-Cambacedes, and Patrick Schweitzer. 2014. DAG-based attack and defense modeling: Don't miss the forest for the attack trees. *Computer science review 13* 13 (2014), 1–38.

[6] Changwei Liu, Anoop Singhal, and Duminda Wijesekera. 2015. A logic-based network forensic model for evidence analysis. In *IFIP International Conference on Digital Forensics*. Springer, 129–145.

[7] Monali Mavani and Krishna Asawa. 2017. Modeling and analyses of IP spoofing attack in 6LoWPAN network. *Computers & Security* 70 (2017), 95–110.

[8] MITRE. [n.d.]. MITRE ATT&CK. Retrieved Jan 2, 2019 from https://attack.mitre.org/

[9] Richard E Neapolitan et al. 2004. *Learning bayesian networks*. Vol. 38. Pearson Prentice Hall Upper Saddle River, NJ.

[10] Thomas Dyhre Nielsen and Finn Verner Jensen. 2009. *Bayesian networks and decision graphs.* Springer Science & Business Media.

[11] Xinming Ou, Wayne F Boyer, and Miles A McQueen. 2006. A scalable approach to attack graph generation. In *Proceedings of the 13th ACM conference on Computer and communications security*. ACM.

[12] Xinming Ou, Sudhakar Govindavajhala, and Andrew W Appel. 2005. MulVAL: A Logic-based Network Security Analyzer. In *USENIX Security Symposium*, Vol. 8.

[13] Cynthia Phillips and Laura Painton Swiler. 1998. A graph-based system for network-vulnerability analysis. In *Proceedings of the 1998 workshop on New security paradigms*. ACM, 71–79.

[14] Nayot Poolsappasit, Rinku Dewri, and Indrajit Ray. 2012. Dynamic security risk management using bayesian attack graphs. *IEEE Transactions on Dependable and Secure Computing* 9, 1 (2012), 61–74.

[15] Xinzhou Qin and Wenke Lee. 2004. Attack plan recognition and prediction using causal networks. In *Computer Security Applications Conference, 2004. 20th Annual.* IEEE, 370–379.

[16] Diptikalyan Saha. 2008. Extending logical attack graphs for efficient vulnerability analysis. In *Proceedings of the 15th ACM conference on Computer and communications security*. ACM, 63–74.

[17] Jaka Sembiring, Mufti Ramadhan, Yudi S Gondokaryono, and Arry A Arman. 2015. Network security risk analysis using improved MulVAL Bayesian attack graphs. *International Journal on Electrical Engineering and Informatics* 7, 4 (2015), 735.

[18] Laura P Swiler, Cynthia Phillips, David Ellis, and Stefan Chakerian. 2001. Computer-attack graph generation tool. In *discex*. IEEE, 1307.

[19] Yien Wang and Jianhua Yang. 2017. Ethical Hacking and Network Defense: Choose Your Best Network Vulnerability Scanning Tool. In *2017 31st International Conference on Advanced Information Networking and Applications: Workshops (WAINA)*. IEEE.

[20] Peng Xie, Jason H Li, Xinming Ou, Peng Liu, and Renato Levy. 2010. Using Bayesian networks for cyber security analysis. In *Dependable Systems and Networks (DSN), 2010 IEEE/IFIP international conference on*. IEEE, 211–220.

A SAMPLE INPUT AND OUTPUT OF MULVAL

Listing 2 shows example rules extracted from MulVAL's default interaction rule set. The first rule represents an attack that exploits a vulnerability which causes arbitrary code execution. The *vulExist* predicate means that there is a vulnerability that can be exploited remotely (*remoteExploit*) and cause privilege escalation (*privEscalation*) at a software *S* on the host *H*. The *networkServiceInfo* predicate means that software *S*, which runs on the host *H* with the permission *Perm*, listens to *Protocol* and *Port*. The *netAccess* predicate means that attacker can access to the host *H* via the network using the *Protocol* and the *Port*. If these three predicates are satisfied, the attacker can execute arbitrary code on the host *H* with the permission *Perm* (represented by the *execCode predicates*).

The second rule represents the scenario that an attacker can access a host via the network. The *attackerLocated* predicate means that the attacker is located at *Zone* (e.g., the attacker gained access to a host in some subnet). The *hacl* predicate means that communication from *Zone* to the host *H* using *Protocol* and *Port* is allowed. If these two predicates are satisfied, the attacker can access the host *H* via the network by using *Protocol* and *Port* (represented by the *netAccess* predicate).

Listing 3 shows a sample input, and Figure 6 shows the attack graph generated by MulVAL from this input. Table 7 shows corresponding node descriptions. Nodes 2 and 4 correspond to the rules in Listing 2.

```
execCode(H, Perm) :-
    vulExists(H, _vulID, S, remoteExploit, privEscalation),
    networkServiceInfo(H, S, Protocol, Port, Perm),
    netAccess(H, Protocol, Port))

netAccess(H, Protocol, Port) :-
    attackerLocated(Zone),
    hacl(Zone, H, Protocol, Port)),
```

Listing 2: Sample rules included in default interaction rule set.

```
attackerLocated(zoneA).
attackGoal(execCode(hostX, admin)).

hacl(zoneA, hostX, tcp, 80).
vulExists(hostX, cveXXXX, vulSoftware, remoteExploit,
      privEscalation).
networkServiceInfo(hostX, vulSoftware, tcp, 80, admin).
```

Listing 3: Sample input file.

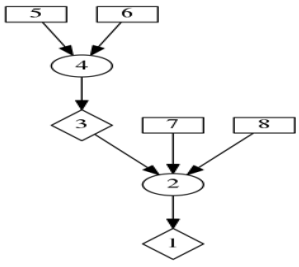

Figure 6: MulVAL output example

Table 7: Node description corresponding to Figure 6

1	execCode(hostX,admin)
2	RULE 2 (remote exploit of a server program)
3	netAccess(hostX,tcp,80)
4	RULE 6 (direct network access)
5	hacl(zoneA,hostX,tcp,80)
6	attackerLocated(zoneA)
7	networkServiceInfo(hostX,vulSoftware,tcp,80,admin)
8	vulExists(hostX,cveXXXX,vulSoftware,remoteExploit,privEscalation)

B CREATED RULES

Rules corresponding to attack actions A3 and A10 as an example of exploiting a vulnerability are specified in Listings 4 and 5 respectively.

```
localAccess(Principal, DstHost, NetwrokServiceUser):-
    netAccess(Principal, SrcHost, DstHost,Prot, Port),
    networkService(DstHost, Prog, Prot, Port, NetworkServiceUser),
    vulHost(DstHost, _vulID, Prog, remoteExploit, execCode),
    malicious(Principal)
```

Listing 4: Local access by exploiting vulnerability (A3).

```
localAccess(Principal, Host, admin):-
    localService(Host, Prog, User),
    vulHost(Host, _vulID, Prog, localExploit, privEsc),
    malicious(Principal),
    localAccess(Principal, Host, User)
```

Listing 5: Gain administrator privilege by exploiting a vulnerability (A10).

A rule that infers data injection by action A9 is specified in Listing 6.

```
dataInject(Principal, DstHost, Path2):-
    accessFile(Principal,SrcHost,write,Path1),
    dataBind(Flow,SrcHost,Path1),
    dataFlow(SrcHost,DstHost,Flow),
    dataBind(Flow,DstHost,Path2)
```

Listing 6: Data injection from existing data flow.

Legitimate file access after local access (A11) can be expressed by the combination of the *localAccess* and *localFileProtection* predicates as specified in Listing 7.

```
accessFile(Principal, Host, Access, Path):-
    localFileProtection(Host, User, Access, Path),
    localAccess(Principal, Host, User)
```

Listing 7: File access after local access.

Attacker's access to credential (A13) can be inferred by the rule specified in Listing 8.

```
hasAccount(Principal, Host2, User2):-
    accessFile(Principal, Host1, read, Path),
    dataBind(Credential, Host1, Path),
    isCredential(Credential, Host2, User2),
    malicious(Principal)
```

Listing 8: Gain credential by file access.

C NODE DESCRIPTONS FOR CASE STUDY

Table 8 and 9 show node descriptions corresponding to Figure 4 and 5.

Table 8: Node description corresponding to Fig.4

1	execCode(workStation,root)
2	RULE 4 (Trojan horse installation)
3	accessFile(workStation,write,'/usr/local/share')
4	RULE 16 (NFS semantics)
5	accessFile(fileServer,write,'/export')
6	RULE 10 (execCode implies file access)
7	canAccessFile(fileServer,root,write,'/export')
8	execCode(fileServer,root)
9	RULE 2 (remote exploit of a server program)
10	netAccess(fileServer,rpc,100005)
11	RULE 5 (multi-hop access)
12	hacl(webServer,fileServer,rpc,100005)
13	execCode(webServer,apache)
14	RULE 2 (remote exploit of a server program)
15	netAccess(webServer,tcp,80)
16	RULE 6 (direct network access)
17	hacl(internet,webServer,tcp,80)
18	attackerLocated(internet)
19	networkServiceInfo(webServer,httpd,tcp,80,apache)
20	vulExists(webServer,'CAN-2002-0392',httpd,remoteExploit, privEscalation)
21	networkServiceInfo(fileServer,mountd,rpc,100005,root)
22	vulExists(fileServer,vulID,mountd,remoteExploit,privEscalation)
23	RULE 17 (NFS shell)
24	hacl(webServer,fileServer,nfsProtocol,nfsPort)
25	nfsExportInfo(fileServer,'/export',write,webServer)
26	nfsMounted(workStation,'/usr/local/share',fileServer,'/export',read)

Table 9: Node description corresponding to Fig.5

1	localAccess(attacker,workStation,admin)
2	RULE 7 (incompetent user execute malicious code included in injected data)
3	inCompetent(administrator)
4	accessFile(administrator,workStation,exec,'/user/local/share')
5	RULE 5 (principal can access file)
6	localAccess(administrator,workStation,admin)
7	RULE 0 (localAccess derived from primitive)
8	localAccess_p(administrator,workStation,admin)
9	localFileProtection(workStation,admin,exec,'/user/local/share')
10	malicious(attacker)
11	accessFile(attacker,workStation,write,'/user/local/share')
12	RULE 6 (principal can inject arbitrary data from dataflow source)
13	accessFile(attacker,fileServer,write,'/export')
14	RULE 4 (vulnerability is exploited via network)
15	netAccess(attacker,webServer,fileServer,rpc,100005)
16	RULE 2 (principal can access from neighber)
17	aclNW(webServer,fileServer,rpc,100005)
18	localAccess(attacker,webServer,admin)
19	RULE 3 (vulnerability is exploited via network)
20	netAccess(attacker,internet,webServer,tcp,80)
21	RULE 2 (principal can access from neighber)
22	aclNW(internet,webServer,tcp,80)
23	localAccess(attacker,internet,admin)
24	RULE 1 (localAccess derived attackerLocated)
25	attackerLocated(internet)
26	vulHost(webServer,'CAN-2002-0392',httpd,remoteExploit,privEsc)
27	networkService(webServer,httpd,tcp,80,apache)
28	vulBind(fileServer,'CVE-XXXX-YYYY','/export')
29	vulHost(fileServer,'CVE-XXXX-YYYY',mountd,remoteExploit, dataTamper)
30	networkService(fileServer,mountd,rpc,100005,admin)
31	RULE 5 (principal can access file)
32	localAccess3(attacker,fileServer,admin)
33	RULE 3 (vulnerability is exploited via network)
34	vulHost(fileServer,vulID,mountd,remoteExploit,privEsc)
35	localFileProtection(fileServer,admin,write,'/export')
36	RULE 6 (principal can inject arbitrary data from dataflow source)
37	accessFile(attacker,webServer,write,'/export')
38	RULE 5 (principal can access file)
39	localFileProtection(webServer,admin,write,'/export')
40	dataBind(nfsFlow,fileServer,'/export')
41	dataFlow(webServer,fileServer,nfsFlow)
42	dataBind(nfsFlow,webServer,'/export')
43	dataBind(nfsFlow,workStation,'/user/local/share')
44	dataFlow(fileServer,workStation,nfsFlow)