ON ATTACK GRAPH MODEL OF NETWORK SECURITY

Hasmik Sahakyan, Daryoush Alipour

Abstract: *All types of network systems are subject to computer attacks. The overall security of a network cannot be determined by simply considering the vulnerable points in the network; it is essential to realize how vulnerabilities can be combined in the same host or in a set of connected hosts to initiate an attack. Attack graph is a tool for modeling compositions of vulnerabilities and thus representing possible multi-stage multi-host attacks in networks. Attack graphs can be used for measuring network security; supporting security solutions by identifying vulnerabilities that should be removed such that no attack can be realized targeting given critical resources, and thus hardening the network. We consider a general model of attack graphs and a scheme of attack graph generating algorithm; and investigate graph-theoretical problems related to particular tasks of network hardening.*

Keywords: *Attack graph model; Network security*

ACM Classification Keywords: *C.2 Computer-communication networks; G2.2 Graph Theory; G2.3 Applications*

1. Introduction

All types of network systems are subject to computer attacks. The overall security of a network cannot be determined by simply considering the vulnerable points in the network. To evaluate the network security, it is essential to understand how vulnerabilities can be combined in the same host or in a series of connected hosts to initiate attacks. Attack graph is a tool for modelling compositions of vulnerabilities and thus enumerating multi-stage multi-host attacks in networks (see e.g. [Aslanyan et al, 2013; Barik et al, 2014; Noel et al, 2010; Sheyner et al, 2002; Zhang et al, 2009]). Generally, attack graphs are large and complex, and automatic and efficient generation of attack graphs is an important issue. It is also important to analyze attack graphs for measuring network security, supporting security solutions by identifying vulnerabilities that should be removed such that none of the attack paths leading to a given critical resource can be realized, and thus by hardening the network. In this paper we address graph-theoretical problems related to particular tasks in the network security.

The paper is organized as follows. A brief overview of network security is given in Section 2 below. Section 3 introduces network vulnerability model and the role of attack graphs. A simple algorithm of

generating attack graph is described for an attack graph model. Section 4 is devoted to graph-theoretical problems and algorithms related to particular network hardening tasks.

2. Network Security

A *computer network* is a group of computer systems and other computing hardware devices that are linked together through communication channels enabling communication, data exchange and resource sharing between users (Figure 1).

Network security refers to protection of resources. The resources to be protected include:

- All types of information resources (user-generated data, programs, computer services and processes);
- Communication infrastructure (communications devices, transmission paths, communication data);
- Computer system (hardware software operability).

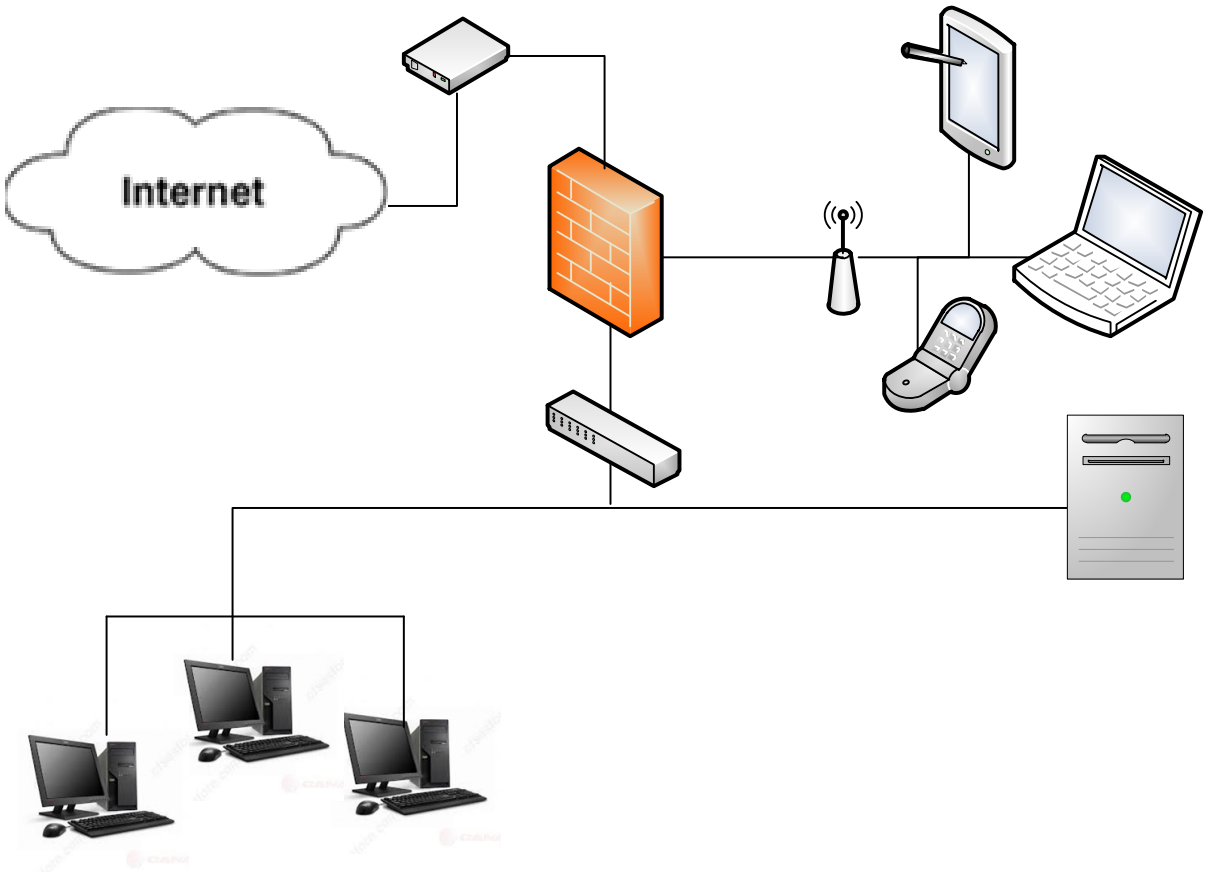

Figure 1. Computer network in an example

In general, three main components of *network security* are of basic interest:

- Confidentiality: the information must be protected from disclosure to unauthorized parties;
- Integrity: the information must be protected from being modified by unauthorized parties;
- Availability: the information must be available when it is needed and access right is in place, - must be protected against unauthorized deletion/modification of data or causing a denial of service of data.

The base elements of network attacks are threats and vulnerabilities.

Threats are actions, events, or circumstances that have the potential to compromise network security, for example, to obtain unauthorized access to data.

Vulnerability is a flaw or weakness in a network that can be exploited by one or more threats to gain access to a system or network.

For example, the vulnerability can allow an attacker:

- To execute commands as another user;
- To access data that has access restrictions;
- To conduct a denial of service action, etc.

Vulnerabilities can be exploited by human or another (technical) system to initiate an *attack*. *Network attack* is an intrusion on the network infrastructure. First it can collect and analyze information in order to exploit the existing vulnerabilities. In such cases the purpose is only to get some information from the system, - these are *passive attacks*. *Active attacks* occur when resources or data are modified, disabled or destroyed.

Examples of network attacks are:

DoS (Denial-of-Service) – Send more requests to the computer than it can handle.

Unauthorized access – Access some resource that the computer should not provide the attacker. For example, a host might be a web server and should provide anyone with requested web pages. However the host should not provide command shell access to a person who should not get it.

There are some standards to classify vulnerabilities (see e.g. [CVE], [NVD], [OSVD]). Common *Vulnerabilities and Exposures* (CVE®) is a dictionary of common names for publicly known information security vulnerabilities. The CVE vulnerabilities have three parts. Their styles are in the following format: CVE-year-vulnerability number. For example:

Name: *CVE-1999-0012*

Description: Some web servers under Microsoft Windows allow remote attackers to bypass access restrictions for files with long file names.

Common Vulnerability Scoring System (CVSS) is a standard designed to convey vulnerability severity and help determine urgency and priority of response ([CVSS]). CVSS has been adopted by a number of vulnerability database providers. CVSS consists of 3 groups: Base, Temporal and Environmental. Each group produces a numeric score ranging from 0 to 10, and a Vector, a compressed textual representation that reflects the values used to derive the score. The Base group represents the intrinsic qualities of vulnerability. The Temporal group reflects the characteristics of vulnerability that change over time. The Environmental group represents the characteristics of vulnerability that are unique to any user's environment. CVSS base score consists of *exploitability metrics* and *impact metrics*. In the exploitability metrics there are three metrics for *Access Vector* (AV), *Attack Complexity* (AC) and *Authentication* (Au). The exploitability metrics measure characteristics of the vulnerability that affect the difficulty of exploitation of the vulnerability. The impact metric contains the following components: Confidentiality Impact (C), Integrity Impact (I), and Availability Impact (A). The impact metric measures how vulnerability, if exploited, will directly affect an IT asset, where the impacts are independently defined as the degree of loss of confidentiality, integrity, and availability. For example, vulnerability could cause a partial loss of integrity and availability, but no loss of confidentiality.

Exploitability Metrics

Access Vector (AV)		
Local (AV:L)	Adjacent Network (AV:A)	Network (AV:N)
Access Complexity (AC)		
High (AC:H)	Medium (AC:M)	Low (AC:L)
Authentication (Au)		
Multiple (Au:M)	Single (Au:S)	None (Au:N)

Impact Metrics

Confidentiality Impact (C)		
None (C:N)	Partial (C:P)	Complete (C:C)
Integrity Impact (I)		
None (I:N)	Partial (I:P)	Complete (I:C)
Availability Impact (A)		
None (A:N)	Partial (A:P)	Complete (A:C)

Scoring equations and algorithms for the base, temporal and environmental metric groups are also described.

Consider the following example. Let assume that in one computer of the network “Microsoft Office 2010” is installed.

Related to “Microsoft Office 2010” CVE includes the following vulnerability:

Name: CVE-2015-1649

Description: Use-after-free vulnerability in Microsoft Word 2007 SP3, Office 2010 SP2, Word 2010 SP2, Word Viewer, Office Compatibility Pack SP3, Word Automation Services on SharePoint Server 2010 SP2, and Office Web Apps Server 2010 SP2 allows remote attackers to execute arbitrary code via a crafted Office document, aka "Microsoft Office Component Use After Free Vulnerability."

Consider the corresponding CVSS vector for CVE-2015-1649.

Exploitability Metrics

Access Vector (AV)		
Local (AV:L)	Adjacent Network (AV:A)	Network (AV:N)
Access Complexity (AC)		
High (AC:H)	Medium (AC:M)	Low (AC:L)
Authentication (Au)		
Multiple (Au:M)	Single (Au:S)	None (Au:N)

Impact Metrics

Confidentiality Impact (C)		
None (C:N)	Partial (C:P)	Complete (C:C)
Integrity Impact (I)		
None (I:N)	Partial (I:P)	Complete (I:C)
Availability Impact (A)		
None (A:N)	Partial (A:P)	Complete (A:C)

In the part of Exploitability Metrics

1. Access Vector is "Network", since CVE-2015-1649 can be exploited remotely;
2. Access Complexity is not "High" because this vulnerability is not exploitable at the attacker's whim, and it is not low because some additional access or specialized circumstances need to exist for the exploit to be successful;
3. Authentication is "None" because the attacker does not need to authenticate to any additional system.

In the part of Impact Metrics

If an administrative user were to run the virus scan, causing the buffer overflow, then a full system compromise would be possible. Since the most harmful case must be considered, each of the three Impact metrics is set to "Complete" because of the possibility of a complete system compromise.

Thus, CVSS base score for CVE-2015-1649 is: (AV: N/AC:M/Au:N/C:C/I:C/A:C).

There are known software /vulnerability scanners/ designed to scan computers and networks for vulnerabilities and then report about the identified vulnerabilities. Network-based vulnerability scanners, such as Port Scanners (Nmap, Nessus), Web application security scanner, Network vulnerability scanner (BoomScan) - are installed on a computer that scans a number of other hosts on the network. Host-based scanners, such as Database Security Scanner, - are installed in the host.

However, to achieve the attack goals attackers may need to use not only separate vulnerabilities but also combinations of vulnerabilities, i.e. they can attack a vulnerable computer and then use it for further attack goal. Thus, the overall security of the network cannot be determined by simply counting the vulnerabilities, and an important task in network security is to analyze which vulnerabilities are acceptable risks; how particular vulnerabilities or exploits can be combined and exploited in complex attacks; and to support security solution.

One approach for modeling how particular vulnerabilities can be combined for an attack that is our interest in this article - is the model of attack graphs.

3. Network Vulnerability Model and Attack Graphs

Generally, to efficiently evaluate security of a network system, it is necessary to develop a network vulnerability model that illustrates the security risk properties of the system.

For composing *network vulnerability model* it is necessary to know network configuration (hosts, operating systems, application programs, network services, etc.); network connectivity, including the connectivity-limiting effects of devices such as firewalls and router access control lists. Then it should be identified vulnerabilities and interdependency between them.

Thus the model will have the following components:

- Hosts;
- Services in every host;
- Vulnerabilities of every service;
- Connectivity between services/vulnerabilities;
- Possible attacks.

Let $H = \{h_1, \cdots, h_n\}$ denote the set of *hosts* in a network that can potentially be targeted by an attacker.

Let $V_i = \{v_{i,1}, v_{i,2}, \cdots, v_{i,m_i}\}$ denote the set of vulnerable services running at host $h_i \in H$, $i = 1,2, \cdots, n$; we suppose that vulnerabilities descriptions are known (for example, from CVE, CVSS).

Let $C_{i,j} = \{c_{i,j,1}, c_{i,j,2}, \cdots, c_{i,j,m_j}\}$ denote the set of connectivity relations from the host h_i to the vulnerable services running at host h_j, $i = 1,2, \cdots, n, j = 1,2, \cdots, n, i \neq j$

$$c_{i,j,k} = \begin{cases} 1, if\ there\ is\ a\ connection\ from\ host\ h_i\ to\ v_{j,k} \\ 0, \quad otherwise \end{cases}$$

Let A denote the set of possible attacks.

Generally, an attack $a \in A$ can be initiated if some certain network conditions exist (for example, a service running in the destination host can be accessed from a source host) and/or an attacker has certain privilege on certain hosts (for example, attacker has *user* privilege on source host). These are *attack preconditions*. Successful execution of an attack may create new attacker privilege or new network conditions. These are *postconditions* of the attack. Let a_{pre} and a_{post} denote the sets of preconditions and postconditions of the attack a, respectively.

Thus, each attack a can be given by the following elements:

- Source host h_{src} from where a is launched;
- Target host h_{dest};
- Target vulnerability v that exist at h_{dest};
- Set a_{pre} of attack preconditions that enable to attack the vulnerability v from the host $h_i \in H$;
- Set a_{post} of attack postconditions on host h_{dest} obtained after successfully attacking target vulnerability v.

To achieve the attack goals attackers may need to use not only separate vulnerabilities but also combinations of vulnerabilities. However, the vulnerability scanners cannot directly identify the complex attack routes on the network. The attack postconditions of an attack $a \in A$ initiated from the source host h_{src} to the destination host h_{dest}, can be the attack precondition for another attack $a' \in A$ from the host h_{dest}. By knowing the characteristics of vulnerabilities, the preconditions required exploiting them, and the postcondition of exploiting them, it becomes possible to chain possible simple/atomic attacks $a \in A$ together into a sequence of attacks that achieve a certain goal. This is the information that attack graphs represent.

Thus, *attack graph* is a tool for enumerating multi-stage multi-host attacks in networks. Without this tool it is very difficult to manually discover how an attacker can combine vulnerabilities in the same host or in connected hosts to compromise critical resources. The task becomes more difficult as the number of vulnerabilities as well as the size of network increases. Attack graphs can be used for measuring

network security, supporting security solutions by identifying vulnerabilities that should be removed such that none of the attack paths leading to a given critical resource can be realized, and thus by hardening the network. The low cost of removing the vulnerabilities is also important.

There are different types of attack graphs, and different algorithms for generating them. We will address a general model of attack graphs based on exploit dependency attack graph model.

Formally, attack graph can be represented as a directed bipartite graph $G = (V_1 \cup V_2, E)$. Nodes in V_1 correspond to either *attacker privilege* or *network conditions*. Nodes in V_2 correspond to *Attacks/Exploits.* Directed edges from V_1 to V_2 are preconditions of attacks/exploits. Directed edges from V_2 to V_1 are postconditions of executing exploits/attacks.

Consider nodes in V_1. An attacker can have certain privilege on certain hosts. For example, the attacker can have *user* privilege on host "h". Then, there will be a corresponding node in V_1: *user("h")*. Certain services running at the destination host can be accessed from a source host. There will be corresponding nodes in V_1, for example, if *ftp* service is running, then the node *ftp("h1","h2")* will refer that *ftp* service running at "*h2"* is accessible from "*h1"*.

To execute attack attackers may need multiple network conditions (preconditions of the attack). Successful execution of an exploit may create new attacker privilege or new network conditions (postconditions of the attack). Nodes in V_2 can be of form: *Exploit("h1","h2")* or *Exploit("h1")* - meaning that having some privilege on "*h1"*, an attacker can perform the exploit *Exploit at "h2"*; or the *Exploit* can be performed locally at "*h1".*

Consider a simple example: the network consisting of *host1*, *host2*, *host3*. Let in *host1* and *host2* "*Office Web Apps Server*"[1] has been installed. It is mentioned in CVE database that this software has a vulnerability CVE-2015-1649, which allows remote attackers to execute arbitrary code via a crafted Office document, aka "Microsoft Office Component Use after Free Vulnerability." Therefore in our assumed network, attacker using this vulnerability can gain *user* privilege on the host running the service "*Office Web Apps Server*". According to sources of CVE, this kind of vulnerability is called Use-after-free, that means that when a user execute a weak software then an error occurs, and the pointer is immediately freed ([CWE]). However, this pointer is later incorrectly used in the other function.

Attacker uses a *buffer overflow* or *use-after-free* kinds of *memory corruption* errors to overwrite control-data and control flow of the program finally.

[1] Office Web Apps Server is a new Office server product that delivers browser-based versions of Word, PowerPoint, Excel, and OneNote. A single Office Web Apps Server farm can support users who access Office files through SharePoint 2013, Lync Server 2013, Exchange Server 2013, shared folders, and websites.[https://technet.microsoft.com/en-us/library/jj219437.aspx]

Assume that "*Office Web Apps Server*" service at host2 can be accessed from both host3 and host1, and "*Office Web Apps Server*" service at host1 can be accessed from host2 only. The attacker has initially *user* privilege on host3. Figure 2 demonstrates the network.

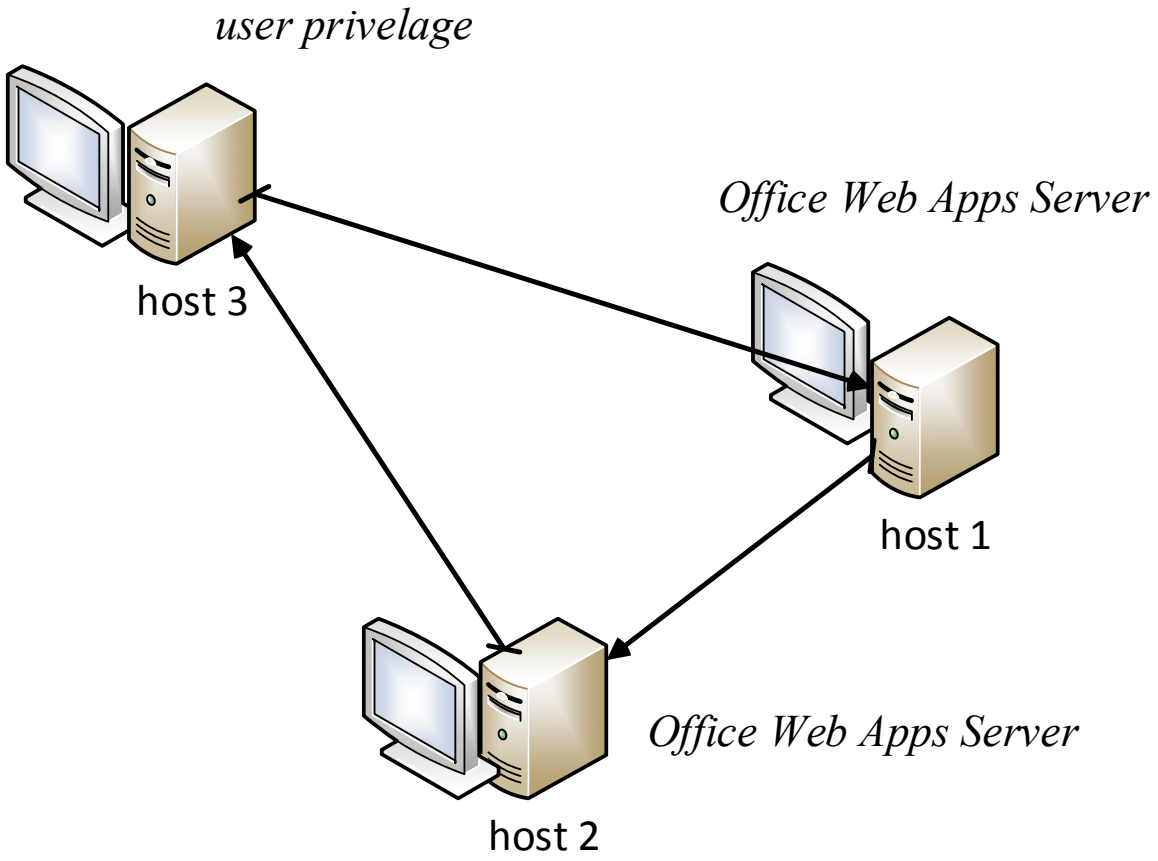

Figure 2. An example network to construct the Attack Graph

Now we construct the corresponding attack graph. Initially, the *precondition* nodes (nodes in V_1) are the following: *user(3)* (attacker privilege) and *OfficeWebAppsServer(3,1)*, *OfficeWebAppsServer(3,2)*, *OfficeWebAppsServer(1,2)* (network conditions).

user(3) and *OfficeWebAppsServer(3,1)* make it possible the exploit: *use-after-free (3,1)*. Similarly, *user(3)* and *OfficeWebAppsServer(3,2)* make it possible the exploit: *use-after-free(3,2)*. Thus, the graph should have *use-after-free(3,1)* and *use-after-free(3,2) exploit/attack* nodes (nodes in V_2).

This part of attack graph is shown in Figure 3. Oval-nodes correspond to nodes in V_1; and the rectangle-nodes correspond to nodes in V_2.

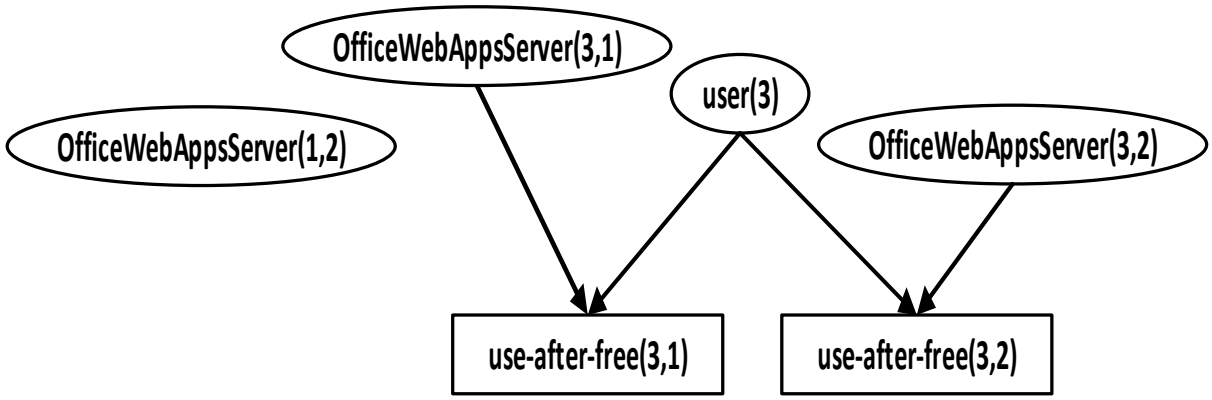

Figure 3. Part of Attack Graph

Exploiting the *use-after-free(3,1),* an attacker obtains *user* privilege on host1 *and* exploiting the use-after-free*(3,2)* an attacker obtains *user* privilege on host2 (*postcondition nodes user(1) and user(2))* (Figure 4).

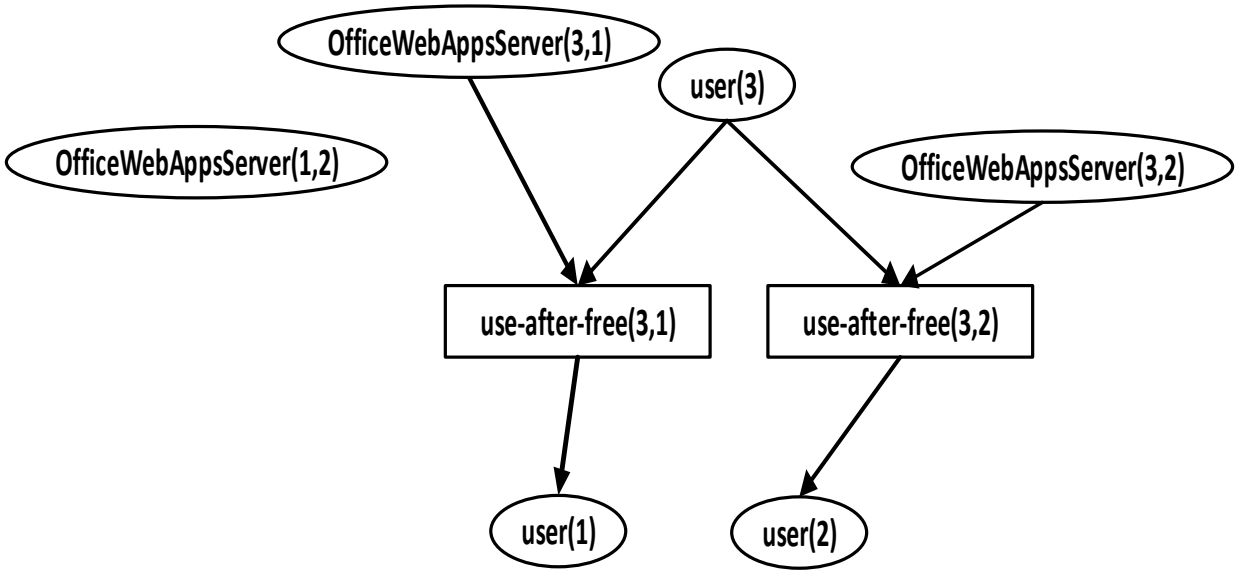

Figure 4. Part of Attack Graph in the next step.

user(1) and OfficeWebAppsServer(1,2) make it possible the exploit use-after-free(1,2), which in its turn give the attacker user privilege on host2. Figure 5 demonstrates the whole graph.

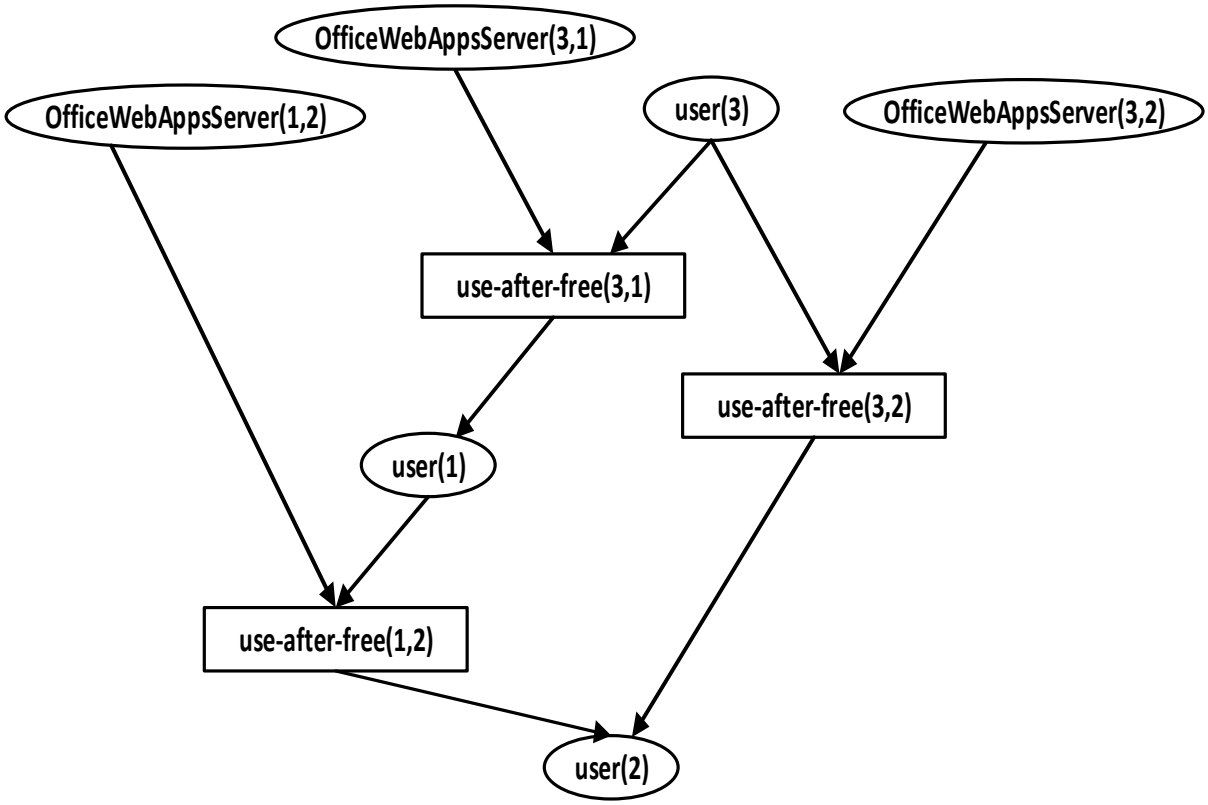

Figure 5. An Attack Graph example

Indeed the real practical networks are quite large and an automated Attach Graph generation is an objective. It is to be composed by the collected descriptions of the network and it is supposed that the data collection is also automated. Let us formulate our Attack Graph generation algorithm.

Algorithm A_G

Input of the algorithm A_G.

Let $H = \{h_1, \cdots, h_n\}$ denote the set of *hosts* in a network that can potentially be targeted by an attacker. Let $V_i = \{v_{i,1}, v_{i,2}, \cdots, v_{i,m_i}\}$ denote the set of vulnerable services running at host $h_{\mathrm{i}} \in H$, $i = 1,2, \cdots, n$. We suppose that vulnerabilities descriptions are known (for example, from CVE, CVSS). Let $C_{i,j} = \{c_{i,j,1}, c_{i,j,2}, \cdots, c_{i,j,m_j}\}$ denote the set of connectivity relations from the host h_i to the vulnerable services running at host h_j, $i = 1,2, \cdots, n$, $j = 1,2, \cdots, n, i \neq j$.

Input of the algorithm A_G

H, V_i for $i = 1,2,\cdots,n$, and $C_{i,j}$ for $i = 1,2,\cdots,n, j = 1,2,\cdots,n$

Output of the algorithm A_G is the corresponding attack graph.

```
begin
source_H=dest_H=H;
    while (source_H is not empty)
    {
    source:=take_host_from(source_H);
    if user(source) then
    {
    source_H=current(source_H,source)
    add_node_vuln_type_with_label("user(source)");
    dest_H=current(dest_H,source);
    while (dest_H is not empty)
    {
     dest= take_host_from(dest_H);
     dest_H=current(dest_H,dest);
     while (V_dest is not empty)
     {
      vuln=take_vuln_from(V_dest);
      V_dest=current(V_dest,vuln);
      if (connect(source,dest,vuln)) then
      {
       add_node_vuln_type_with_label(vuln);
       Precond = add(Precond)
       if (find_attack(Precond)) then
          {attack(attack_precond, attack_postcond); add_node_attack_type_with_label(attack_name);
          add_Connection(attack_precond, attack_name); Postcond=attack_postcond;
          add_node_vuln_type_with_label(Postcond); add_Connection(attack_name,attack_Postcond);
          Precond = add(Postcond }
          }
       }
      }
     }
    }
end
```

Let us explain the terms and definitions used:

take_host_from(H) – returns the current host from the set H;

user(h) – returns 1 if attacker has “user” privilege on host “h”, and 0 – otherwise;

current(H,h) – returns the set H\{h};

add_node_vuln_type_with_label(l) – adds a node of type “vulnerability” with the label “l”;

add_node_attack_type_with_label(l) – adds a node of type “attack” with the label “l”;

take_vuln_from(V) – returns the current vulnerable service name from the set V;

connect(h1,h2,v) – returns 1 if the vulnerability “v” at the host “h2” can be accessed from the host “h1”;

P=add(p) – adds into P all vuln_type node labels, created during the current cycle;

find_attack(P) – returns 1 if there is a known attack, which has preconditions that are in P;

attack(name,P1,P2) – returns the attack name, the set of attack preconditions and the set attack postconditions;

add_Connection(F,T) - adds connection from all elements of the set F to the all elements of the set T.

4. Graph-theoretical tasks

In this section we consider graph-theoretical local actions related to the tasks of network hardening.

Let us start with definitions.

Definition: A *directed graph* is a graph, where the edges have a direction associated with them. In formal terms, a directed graph is a pair $D = (V, A)$, where V is the set of vertices, and A is a set of ordered pairs of vertices, called arcs, or directed edges.

An ark $a = (v_1, v_2)$, is considered to be directed from v_1 to v_2. v_2 is called the *head* and v_1 is called the *tail* of the arc.

For a vertex v, the number of head endpoints adjacent to v is called the *in-degree* of the vertex and the number of tail endpoints adjacent to v is its *out-degree*. The in-degree of v is denoted as $deg^-(v)$, and the out-degree as $deg^+(v)$.

Definition: A directed graph $D = (V, A)$ is called a directed bipartite graph if there exists a partition of the vertex set: $V = V_1 \cup V_2$ such that $D[V_1]$ and $D[V_2]$, the two induced directed subgraphs of D, contain no arcs of A (Undefined terms can be found in [Jorgen et al, 2007]).

Tasks related to the vertex degrees in attack graph.

Let $D = (V_1 \cup V_2, A)$ is a directed bipartite graph corresponding to attack graph model, described in the previous section. V_1 consists of *attacker privilege* or *network condition* type nodes, and V_2 consists of *Attack* type nodes.

If a vertex $v_1 \in V_1$ has large number of tail endpoints, then network conditions corresponding to v_1 may allow large number of possible attacks.

If a vertex $v_2 \in V_2$ has small number of head endpoints, then the corresponding attack/exploit has small number of preconditions, and thus, is easy to execute. If v_2 has large number of tail endpoints, then the execution of the corresponding to v_2 attack will create large number of conditions for further attacks. Removal of the corresponding services will reduce the number of possible attacks. Thus, the vertex degree is a managing parameter to be computed.

Thus we formulate:

Task1. Find all vertices in V_1, which have out-degree greater than the given threshold.

Find all vertices in V_2 which have in-degrees less than the given threshold.

These are easy tasks, and simple algorithms can be used (complexity is $|V_1| \times |V_2|$).

Similar task is to find subsets of V_1, such that the summary out-degrees is greater than the given threshold.

Tasks related to the covering problems.

One of the main tasks in network hardening is to identify minimal number of vulnerable services that should be removed such that no attack will be possible.

If we pay no attention to the fact that vertices of V_1 are not homogeneous (meaning that the corresponding vulnerabilities can be created step by step, as results of attack exploits), then we deal with D as undirected graph, and thus consider the following problem.

Find minimal number of nodes in V_1 which "cover" all nodes in V_2.

Now consider an equivalent form of the task – formulated as the set cover problem.

Set cover

Given a finite set S, a collection C of subsets of S, and a positive integer $K \leq |C|$. Does there exist a cover $C' \subseteq C$ of S such that $|C'| \leq K$, i.e. does there exist a subset $C' \subseteq C$ such that $|C'| \leq K$ and every element of S is in at least one subset of C'. This is the *decision version* of the set cover problem.

In the set covering *optimization version*, the task is to find minimal cover. The decision version of the set covering is NP-complete, and the optimization version is NP-hard ([Garey,Johnson, 1979]).

If we associate with each vertex $v \in V_1$ the subset of vertices of V_2 connected to v, then in this manner, V_1 can be considered as collection of subsets of V_2.

Initially, V_1 is a cover for V_2, since each attack node in V_2 is connected with some vulnerability in V_1. Thus, we formulate the task as optimization version of set covering:

Task2. Given a finite set V_2, a collection V_1 of subsets of V_2. Find a minimal cover of V_1.

Many algorithms have been developed for solving the set cover problem. The exact algorithms are mostly based on branch-and-bound and branch-and-cut (e.g. [Balas et al, 1996]). Since exact methods require substantial computational effort to solve large-scale instances of the problem, heuristic algorithms are often used to find a good or near-optimal solution in a reasonable time [Sahakyan, 2014]. Greedy algorithms may be the most natural heuristic approach for quickly solving large combinatorial problems ([Vazirani, 2001], [Bendorz, 2008]).

Greedy Approximation Algorithm for Set cover.

Consider the greedy approximation algorithm for *Task2* :

Algorithm G

Input: Undirected bipartite graph D with parts V_1 and V_2.

Output: cover C.

begin

$C = \emptyset$;

while (V_2 is not empty)

{ take $v \in V_1$ which is connected to maximum number of vertices of V_2;

remove those vertices from V_2;

add v into C

}

end

The algorithm G finds cover whose size is at most O($ln|V_2|$) times the size of minimal set cover.

A similar task is when neutralizing of certain attacks is the interest, which requires that certain vertices are to be covered in V_2.

Task3. Find minimal number of nodes in V_1 that "cover" given vertices $v_1, \cdots, v_k$ in V_2.

Observe that we have simplified the task when consider all vertices of V_1, whilst only those vertices, which are starting points of attack paths may be satisfactory.

Algorithm G' below is a modified version of G.

Algorithm G'.

Input: Directed bipartite graph D with parts V_1 and V_2.

Output: cover C.

begin

$C = \emptyset$;

while (V_2 is not empty)

{ take $v \in V_1$ such that $deg^-(v) = 0$ and $deg^+(v)$ is maximal (V' denotes the set of heads for arcs starting in v);

for each $v' \in V'$

{

if ($deg^+(v') \neq 0$)

Removing_procedure(v');

}

remove V' from V_2;

add v into C;

}

end

Removing_procedure(v') - removes all arc heads having v' as head or tail; and continue this process while the current node has out-degree greater than 0.

Conclusion

Attack graphs as a useful model and tool in the areas of network security can be identifying vulnerabilities that should be removed. We have proposed a general model of representing and generating attack graphs. A simple algorithm of generating attack graph has been described for the attack graph model. Some graph-theoretical problems and algorithms are investigated related to particular network hardening tasks. For future work we plan to improve the algorithm and to create an implementation using real-world network data in realistic situations.

Bibliography

[Aslanyan et al, 2013] L. Aslanyan, D.Alipour and M.Heidari, Comparative Analysis of Attack Graphs. Mathematical Problems of Computer Science, 40, 2013, pp. 85-95.

[Balas et al, 1996] E. Balas, M. Carrera, A dynamic subgradient-based branch-and-bound procedure for set covering. Operations Research 44, 1996, pp. 875–890.

[Barik et al, 2014] M.S. Barik, Ch. Mazumdar, A Graph Data Model for Attack Graph Generation and Analysis, Recent Trends in Computer Networks and Distributed Systems Security, Communications in Computer and Information Science Volume 420, 2014, pp 239-250.

[Bendorz, 2008], Greedy algorithms, edited by W. Bednorz, Publisher: InTech, 2008, 586 pages.

[CVE] Common Vulnerabilities and Exposures (CVE®), the standard for Information security Vulnerability Names, [Online]. Available: http://cve.mitre.org.

[CVSS] Common Vulnerability Scoring System (CVSS-SIG), [Online], Available: http://www.first.org/cvss

[CWE] Common Weakness Enumeration, [Online], Available: http://cwe.mitre.org/data/definitions/416.html.

[Jorgen et al, 2007] Jørgen Bang-Jensen, Gregory Gutin, Digraphs: Theory, Algorithms and Applications, Springer-Verlag, 2002, 754 pages.

[Noel et al, 2010] S. Noel, L. Wang, A. Singhal and S. Jajodia, Measuring security risk of networks using attack graphs, International Journal of Next-Generation Computing, vol. 1, no. 1, 2010, pp. 135-147.

[NVD] National Vulnerability Database, [Online], Available: https://nvd.nist.gov/.

[OSVD] Open Sourced Vulnerability Database, [Online], Available: http://osvdb.org/.

[Sahakyan, 2014] H. Sahakyan, Constrained object-characterization tables and algorithms, International Journal "Information Content and Processing", Volume 1, Number 2, 2014, pp.136-144.

[Shey et al, 2002] O. Sheyner, J. Haines, S. Jha, R. Lippmann, J. M. Wing, Automated generation and analysis of attack graphs, Proceedings of the IEEE Symposium on Security and Privacy, 2002, pp. 254–265

[Vazirani, 2001], V. Vazirani,. Approximation Algorithms, Springer, 2001.

[Zhang et al, 2009] Zhang Lufeng, Tang Hong,Cui YiMing, Zhang JianBo, Network Security Evaluation through Attack Graph Generation, World Academy of Science, Engineering and Technology, 54, 2009.

Authors' Information

Hasmik Sahakyan *– Scientific Secretary, Institute for Informatics and Automation Problems, NAS RA, P. Sevak St. 1, Yerevan 14, Armenia, e-mail: hasmik@ipia.sci.am*

Daryoush Alipour *– PhD student, Institute for Informatics and Automation Problems, NAS RA, P. Sevak St. 1, Yerevan 14, Armenia, e-mail: computernano@gmail.com*