Implementation of STL Vector Class in Reproducing Results from "An Improved Algorithm for Generation of Attack Graph Based on Virtual Performance Node" through Random IP Determination

Sparisoma Viridi[1,*] and Tito Waloyo Purboyo[2]

[1,2]Institut Teknologi Bandung, Jalan Ganesha 10, Bandung 40132, Indonesia

[1,*]dudung@gmail.com, [2]titowaluyo@gmail.com

Abstract. Solving problem using C++ language requiring dynamic size variable can be easier performed using STL vector class. How to reproduces statuses from the article "An Improved Algorithm for Generation of Attack Graph Based on Virtual Performance Node" is traced back in this work by implementing the vector class. A random function in C++ rand() is also used in determining IP for attacker and also the target, imitating guessing from attacker.

Keywords: C++, STL, vector class.

Introduction

Even though the advantages in using C++ Standard Template Library (STL) also accompanied by some problems [1], but it is still very helpful in programming something related to dynamic size variable as used in registering particles in a grid cell [2] and also registering cells in a colony [3]. The vector class from C++ STL [4] is used in this work to reproduce the statuses used in generating attack graph based on Virtual Performance Node (VPN) [5].

Concept and model

As explained in details in [5] there are some matrices should be considered in allowing an attacker to enter the system. The first is links matrix **L**, which describes the interconnections between network hosts

$$\mathbf{L} = \begin{bmatrix} l_{11} & l_{12} & l_{13} & \dots & l_{1N} \\ l_{21} & l_{22} & l_{23} & \dots & l_{2N} \\ l_{31} & l_{31} & l_{32} & \dots & l_{3N} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ l_{N1} & l_{N2} & l_{N3} & \dots & l_{NN} \end{bmatrix}, \quad (1)$$

where l_{ij} follows

$$l_{ij} = \begin{cases} 0, & \text{host } i \text{ does not connect to host } j, \\ 1, & \text{host } i \text{ connects to host } j. \end{cases} \quad (2)$$

Indices i and j are simply representing IP address of a host in the network. In Equation (1) there are N hosts, with IP ranging from 0 to N-1. Then, the diagonal of matrix **L** or the elements l_{ii} will be always 1 since it represents self-connection. In each host there will be different privilege for the attacker, which is described by a privilege one-row matrix **P**

$$\mathbf{P} = \begin{bmatrix} p_1 & p_2 & p_3 & \cdots & p_N \end{bmatrix}, \quad (3)$$

with value of p_i is given in Table 1. The elements v_{ij} has special meaning according to its position in matrix **V** as it is usually used in analyzing networks, e.g. adjacency matrix relating vertexes [6].

$$\mathbf{V} = \begin{bmatrix} v_{11} & v_{12} & \cdots & v_{1K} \\ v_{21} & v_{22} & \cdots & v_{2K} \\ \vdots & \vdots & \ddots & \vdots \\ v_{N1} & v_{N2} & \cdots & v_{NK} \end{bmatrix}. \quad (4)$$

TABLE 1. Constants and their value representing attacker privilege in a host.

Constant	Value	Privilege
P_DISC	-1	disconnection
P_ACCS	0	access
P_USER	1	user
P_ROOT	2	root

TABLE 2. Meaning of elements v_{ij} according to their indices i and j.

row i	column j	Meaning
i	1	target IP_i
i	≥ 2	(j - 1)th vulnerability at host with IP_i

In [5] matrix **V** is represented in two-column matrix only, while the second column could have a set of vulnerabilities. In this work, in order to give clearer illustration, number of vulnerability types is determined by target IP_i with the most number of vulnerabilities K. For host with fewer types of vulnerability it repeats its previous vulnerability to fill the available space. Attack rules is described in matrix **R**

$$\mathbf{R} = \begin{bmatrix} r_{11} & r_{12} & r_{13} & r_{14} & r_{15} & r_{16} & \cdots & r_{1M} \\ r_{21} & r_{22} & r_{23} & r_{24} & r_{25} & r_{26} & \cdots & r_{2M} \\ r_{31} & r_{23} & r_{33} & r_{34} & r_{35} & r_{36} & \cdots & r_{3M} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ r_{V1} & r & r & r & r & r & \cdots & r_{VM} \end{bmatrix} \quad (5)$$

with $M = 5 + N$. Value of V in Equation (5) is number of distinct vulnerabilities in Table 2. Interpretation of the role of each elements r_{ij} is shown in Table 3. There is new indicator named vulnerability performance node (VPN) which has value from 1 to 4, which indicate (1) system response time, (2) system recovery time, (3) throughput rate, and (4) message trans_delay, as defined [5].

TABLE 3. Meaning of elements r_{ij} according to their indices i and j.

row i	column j	Meaning
i	1	vulnerability of type i
i	2	minimal privilege of attacker in source IP
i	3	privilege of attacker in destination IP
i	4	updated privilege of attacker in destination IP
i	5	success probability of the attack
i	$j > 5$	value change of virtual performance node (VPN) at host with IP_j

TABLE 4. Meaning of elements s_{ij} according to their indices i and j.

row i	column j	Meaning
i	1	status i
i	$2 < j < V+1$	value of (j-1)th VPN
i	$V+2 < j < V+N+1$	privilege of attacker in IP (j-V-1)
i	$V+N+2$	source IP
i	$V+N+3$	destination IP
i	$V+N+4$	attack rule type
i	$V+N+5$	previous state of state i

Initial value of VPN is set to -1. Change of VPN on an attacked host could be 0 or 1. Value 1 means the performance loss to reach the higher level.

The last matrix is status matrix **S** which record the state of each attack

$$\mathbf{S} = \begin{bmatrix} s_{11} & s_{12} & \cdots & s_{1,V+1} & s_{1,V+2} & \cdots & s_{1,V+N+1} & s_{1,V+N+2} & s_{1,V+N+3} & s_{1,V+N+4} & s_{1,V+N+5} \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \vdots & \vdots \\ s_{S1} & s_{S2} & \cdots & s_{S,V+1} & s_{S,V+2} & v & s_{S,V+N+1} & s_{S,V+N+2} & s_{S,V+N+3} & s_{S,V+N+4} & s_{S,V+N+5} \end{bmatrix}, \quad (6)$$

which is defined slightly different than in [5], but with improvement containing source and destination IPs, attack rule, and related statuses. Explanation of meaning of element s_{ij} is shown in Table 4.

Flow chart

There are two sections in the flow chart. The first is preparation section, where all matrices are defined as shown in Figure 1. The second is the section where implementation of attack rule is conducted, where this section is given in Figure 2.

The rand(i, j) function will produce integer from i to j, which is normally used to produce IP (both source src and target dest) and to choose rule when a vulnerability has more than one attack rules.

Results and discussion

In this work $N = 5$ with IP begins with 0, $V = 5$, $K = 2$, steps of iteration 10^4 and step for not finding any IP before getting random IP is 10^2. Statuses obtained in this work are given in Table 5.

The most attacked hosts status is with privileges [2 1 2 1 2] which is obtained for status 5, 14, 18, 30, 34, and 39. This state is also observed previously [5]. Statuses in Table 5 are still having duplicates, which is introduce to the results due to the function rand(i, j). A procedure to remove the duplicates should be built in the future.

Summary

A program agvpn.cpp written in C++ implementing STL vector class has been developed and tested to get statuses reported in a work [5]. Similar statuses can be obtained.

Acknowledgements

This work is partially supported by Riset Inovasi Kelompok-Keahlian ITB (RIK-ITB) batch II in year 2015.

References

1. D. Gregor, S. Schupp, "Making the Usage of STL Safe", Generic Programming 115, 127-140 (2003).
2. S. Viridi, Novitrian, "Implementasi Metode Multi-Particle Collision Dynamics untuk Partikel Berukuran Berhingga pada Simulasi Aliran Hagen-Poiseuille dalam Saluran Persegi Panjang Dua Dimensi", Prosiding Simposium Nasional Inovasi dan Pembelajaran Sains 2014 (SNIPS 2014), Eds. S. Pramuditya et al., Bandung, Indonesia, 10-11 Juni 2014, pp. 106-109.
3. D. Aprianti, S. N. Khotimah, S. Viridi, "Budding Yeast Colony Growth Study Based on Circular Granular Cell", the 6th Asian Physics Symposium, Bandung, Indonesia, 19-20 August 2015, COM-03 (presented).
4. A. Alain, "The STL Vector Class", Cprogramming .com, Your resource for C and C++, URL http://www.cprogramming.com/tutorial/stl/vector.html [20150823].
5. Y. Zhao, Z. Wang, X. Zhang, J. Zheng, "An Improved Algorithm for Generation of Attack Graph Based on Virtual Performance Node", Proceedings of 2009 International Conference on Multimedia Information Networking and Security (MINES '09), Hubei, People's Republic of China, 18-20 November 2009, pp. 466-469, DOI 10.1109/MINES.2009.43.
6. M. E. J. Newman, "Networks An Introduction", Oxford University Press, Oxford, Reprinted, 2015, pp. 110-112.

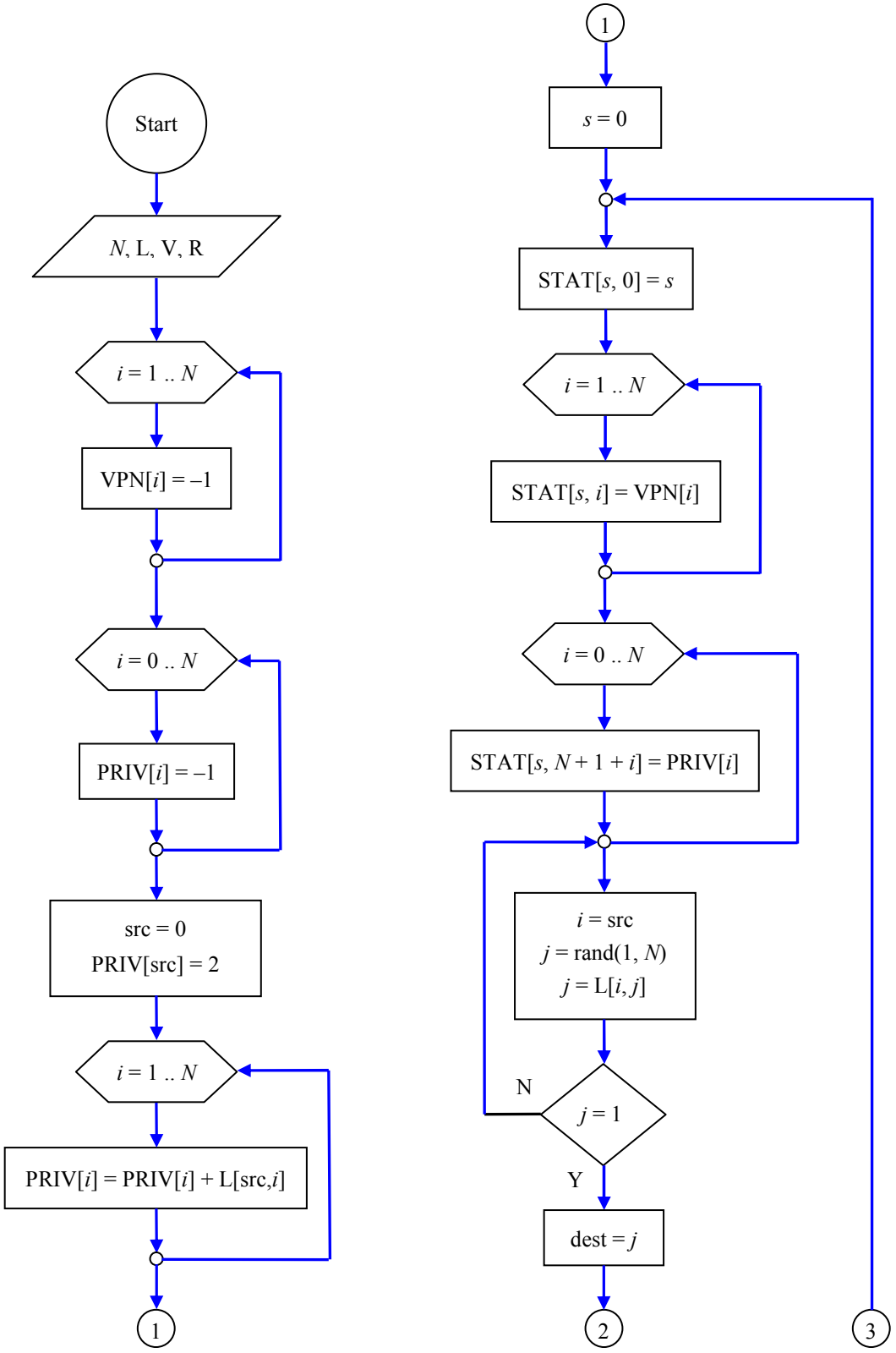

FIGURE 1. Program flow chart for preparation section. Left: Initialization of privilege array PRIV and virtual performance node array VPN. Right: Initialization of status in attack graph matrix STAT and determination of first target IP (or dest) based on links matrix L.

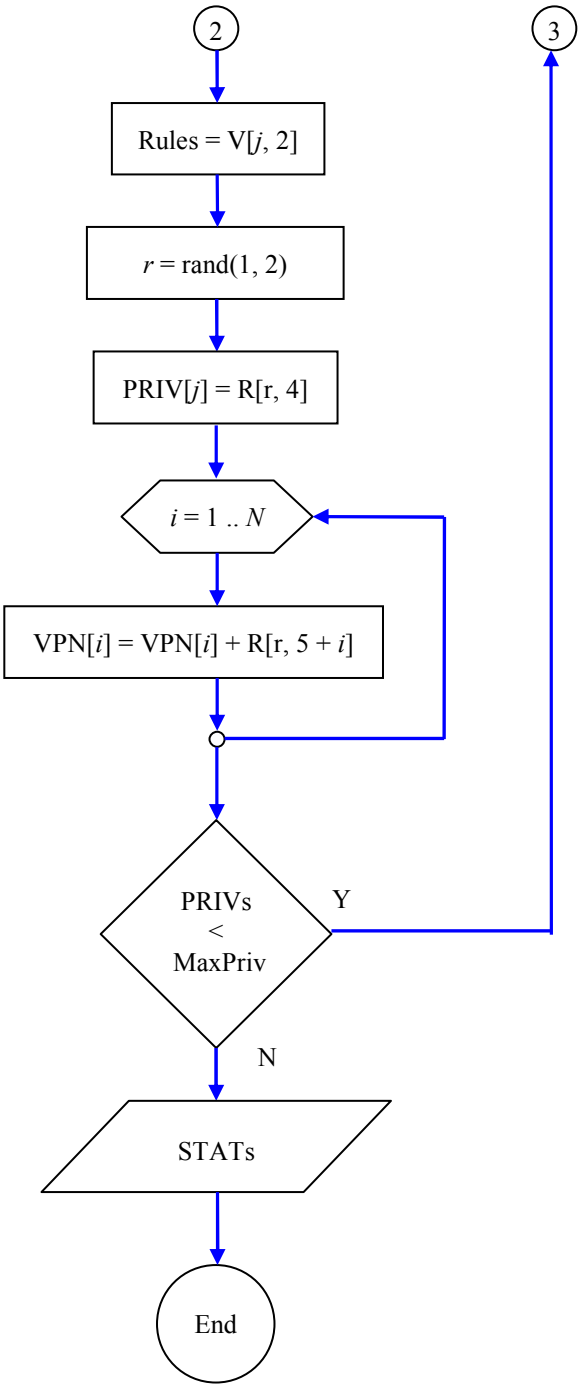

FIGURE 2. Program flow chart for implementation of attack rules.

TABLE 5. Output of the program agvpn.cpp implementing flow chart from Figure 2 and 3*.

# s	vpn1	vpn2	vpn3	vpn4	ip0	ip1	ip2	ip3	ip4	S	D	R	sprev
0	-1	-1	-1	-1	2	0	-1	-1	-1	-1	-1	0	-1
1	0	-1	-1	-1	2	1	0	0	0	0	1	6	0
2	1	0	-1	-1	2	1	0	0	2	1	4	5	1
3	1	0	0	0	2	1	0	1	2	1	3	3	2
4	2	0	1	0	2	1	1	1	2	1	2	1	3
5	2	1	1	0	2	1	2	1	2	1	2	2	4
6	0	-1	0	0	2	1	0	1	0	1	3	3	1
7	1	0	0	0	2	1	0	1	2	1	4	5	6
8	2	0	1	0	2	1	1	1	2	1	2	1	7
9	0	-1	0	0	2	1	0	1	0	1	3	4	1
10	1	-1	1	0	2	1	1	1	0	1	2	1	9
11	1	0	-1	-1	2	1	0	0	2	1	4	5	1
12	2	0	0	-1	2	1	1	0	2	1	2	1	11
13	2	1	0	-1	2	1	2	0	2	1	2	2	12
14	2	1	1	0	2	1	2	1	2	1	3	3	13
15	0	-1	0	0	2	1	0	1	0	1	3	4	1
16	1	0	0	0	2	1	0	1	2	1	4	5	15
17	2	0	1	0	2	1	1	1	2	1	2	1	16
18	2	1	1	0	2	1	2	1	2	1	2	2	17
19	1	0	0	0	2	1	0	1	2	3	4	5	6
20	2	0	1	0	2	1	1	1	2	3	2	1	19
21	1	0	-1	-1	2	1	0	0	2	1	4	5	1
22	2	0	0	-1	2	1	1	0	2	1	2	1	21
23	2	0	1	0	2	1	1	1	2	1	3	4	22
24	1	0	-1	-1	2	1	0	0	2	1	4	5	1
25	2	0	0	-1	2	1	1	0	2	1	2	1	24
26	2	0	1	0	2	1	1	1	2	1	3	3	25
27	1	-1	0	-1	2	1	1	0	0	1	2	1	1
28	1	0	0	-1	2	1	2	0	0	1	2	2	27
29	1	0	1	0	2	1	2	1	0	1	3	3	28
30	2	1	1	0	2	1	2	1	2	1	4	5	29
31	1	-1	0	-1	2	1	1	0	0	1	2	1	1
32	2	0	0	-1	2	1	1	0	2	1	4	5	31
33	2	1	0	-1	2	1	2	0	2	1	2	2	32
34	2	1	1	0	2	1	2	1	2	1	3	4	33
35	1	0	0	0	2	1	0	1	2	4	3	4	2
36	1	-1	0	-1	2	1	1	0	0	1	2	1	1
37	2	0	0	-1	2	1	1	0	2	1	4	5	36
38	2	1	0	-1	2	1	2	0	2	1	2	2	37
39	2	1	1	0	2	1	2	1	2	1	3	4	38
40	2	0	1	0	2	1	1	1	2	2	3	3	22
41	1	0	-1	-1	2	1	0	0	2	1	4	5	1
42	1	0	0	0	2	1	0	1	2	1	3	4	41
43	2	0	1	0	2	1	1	1	2	1	2	1	42
44	1	0	0	0	2	1	0	1	2	4	3	3	21

*Some statuses are still duplicates from other status. Removing the duplicated status should also reorder the status number s and its reference s_{prev}. The duplications are leaving here to show the link between statuses.

Appendix A. Source code

```
/*
	agvpn.cpp
	Attack Graph based on Virtual Performance Node

	Sparisoma Viridi | dudung@gmail.com
	Tito Waluyo Purboyo | titowaluyo@gmail.com

	Compile: g++ agvpn.cpp -o agvpn
	Execute: ./agvpn
	Note: Following files are required:
		attack-rules.txt
		vpns.txt
		links.txt
		ip-vuls.txt
		vulnerabilities.txt

	History:
	20150627
		Create this program (struct, array, vector)
	20150628
		Add vulnerability 0 for attacker.
		Change program name from vpn-simul to agvpn
		Notification where *.txt not available is
		not yet implemented
		Simulation results stuck if there is no
		available IP. It still can not go back in
		the search tree.
	20150822
		Try to understand (again) the code :-p.
		Found the problem, that using iteration of ip, always
		the last ip chosen. Try to change it with random ip.
		Simplify output style, comments uncecessary verbose
		output.
		Solve poblem finding dest_ip using random function [1604].
		Problem by loosing ip to IP_NONE and randomly choose there
		new one, vpn and priv should be loaded from status.
		Create overload operator == for status struct.
*/

# include <iostream>
# include <cmath>
# include <cstring>
# include <cstdlib>
# include <fstream>
# include <sstream>
# include <vector>
# include <ctime>

using  namespace std;

# define V_ROOT 0

# define P_DISC -1
```

```
# define P_ACCS 0
# define P_USER 1
# define P_ROOT 2

# define IP_NONE -1

# define VERBOSE false

struct vulnerability {
public:
	int id;
	string text;
	vulnerability(void);
	~vulnerability(void);
};

vulnerability::vulnerability(void) {
	id = -1;
	text = "";
}

vulnerability::~vulnerability(void) {
	id = -1;
	text = "";
}


struct vpn {
public:
	int id;
	string text;
	vpn(void);
	~vpn(void);
};

vpn::vpn(void) {
	id = -1;
	text = "";
}

vpn::~vpn(void) {
	id = -1;
	text = "";
}


struct ipvuls {
	int ip;
	vector<int> vuls;
	ipvuls(void);
	~ipvuls(void);
};

ipvuls::ipvuls(void) {
	int ip = -1;
```

```
	vuls.clear();
}

ipvuls::~ipvuls(void) {
	int ip = -1;
	vuls.clear();
}


struct attackrule {
	int vid;
	int src;
	int desti;
	int destf;
	double prob;
	vector<int> vpn;
	attackrule(void);
	~attackrule(void);
	string strval(void);
};

attackrule::attackrule(void) {
	vid = -1;
	src = -1;
	desti = -1;
	destf = -1;
	prob = 0.0;
	vpn.clear();
}

attackrule::~attackrule(void) {
	vid = -1;
	src = -1;
	desti = -1;
	destf = -1;
	prob = 0.0;
	vpn.clear();
}

string attackrule::strval(void) {
	stringstream ss;
	ss << vid << "\t";
	ss << src << "\t";
	ss << desti << "\t";
	ss << destf << "\t";
	ss << prob << "\t";
	for(int iv = 0; iv < vpn.size(); iv++) {
		ss << vpn[iv];
		if(iv < vpn.size() - 1) ss << "\t";
	}
	return ss.str();
}


struct status {
```

```
	int cur;
	int prev;
	vector<int> vpn;
	vector<int> priv;
	int src;
	int dest;
	int rule;
	status(void);
	~status(void);
	string strval(void);
};

bool operator ==(const status si, const status sj) {
	bool value = true;
	bool e_src = (si.src == sj.src);
	bool e_dest = (si.dest == sj.dest);
	bool e_rule = (si.rule == sj.rule);
	value = value && e_src && e_dest && e_rule;
	for(int i = 0; i < si.vpn.size(); i++) {
		bool e_vpn = (si.vpn[i] == sj.vpn[i]);
		value = value && e_vpn;
	}
	for(int i = 0; i < si.priv.size(); i++) {
		bool e_priv = (si.priv[i] == sj.priv[i]);
		value = value && e_priv;
	}
	return value;
}

status::status(void) {
	cur = -1;
	prev = -1;
	vpn.clear();
	priv.clear();
	src = -1;
	dest = -1;
	rule = 0;
}

status::~status(void) {
	cur = -1;
	prev = -1;
	vpn.clear();
	priv.clear();
	src = -1;
	dest = -1;
	rule = 0;
}

string status::strval(void) {
	stringstream ss;
	//ss << prev << "\t";
	for(int i = 0; i < vpn.size(); i++) {
		ss << vpn[i];
		ss << "\t";
```

```
	}
	for(int i = 0; i < priv.size(); i++) {
		ss << priv[i];
		ss << "\t";
	}
	ss << src << "\t";
	ss << dest << "\t";
	ss << rule << "\t";
	ss << prev;
	return ss.str();
}


struct sag {
	vector<status> list;
	sag(void);
	~sag(void);
	string content(void);
};

sag::sag(void) {
	list.clear();
}

sag::~sag(void) {
	list.clear();
}

string sag::content(void) {
	stringstream ss;
	//ss << "# cur\tprev\t";
	ss << "# s\t";
	for(int i = 0; i < list[0].vpn.size(); i++) {
		int j = i + 1;
		ss << "vpn" << j << "\t";
	}
	for(int i = 0; i < list[0].priv.size(); i++) {
		ss << "ip" << i;
		ss << "\t";
	}
	ss << "S\tD\tR\tsprev" << endl;
	for(int i = 0; i < list.size(); i++) {
		ss << i << "\t";
		ss << list[i].strval();
		ss << endl;
	}
	return ss.str();
}


int main(int argc, char *argv[]) {
	srand(time(NULL));

	if(VERBOSE) {
		cout << "# agvpn" << endl;
```

```
}

// Read vulnerabilities description file
const char *vulsfn = "vulnerabilities.txt";
ifstream vulsf;
vulsf.open(vulsfn);
int numvul = 0;
while(vulsf.good()) {
	string line;
	getline(vulsf, line);
	if(line.length() > 0) numvul++;
}
vulsf.close();
vulsf.clear();
numvul /= 2;
vulnerability vuls[numvul];
vulsf.open(vulsfn);
int i = 0;
while(vulsf.good()) {
	string line;
	getline(vulsf, line);
	if(line.length() > 0) {
		stringstream ss1;
		ss1 << line << endl;
		ss1 >> vuls[i].id;
		stringstream ss2;
		getline(vulsf, line);
		ss2 << line;
		vuls[i].text = ss2.str();
		i++;
	}
}
vulsf.close();
if(VERBOSE) {
	cout << "Number of vulnerability types = ";
	cout << numvul << endl;
	for(int i = 0; i < numvul; i++) {
		cout << vuls[i].id << endl;
		cout << vuls[i].text << endl;
	}
	cout << endl;
}

// Read VPN description file
const char *vpnfn = "vpns.txt";
ifstream vpnf;
vpnf.open(vpnfn);
int numvpn = 0;
while(vpnf.good()) {
	string line;
	getline(vpnf, line);
	if(line.length() > 0) numvpn++;
}
vpnf.close();
vpnf.clear();
```

```
numvpn /= 2;
vpn vpns[numvpn];
vpnf.open(vpnfn);
i = 0;
while(vpnf.good()) {
	string line;
	getline(vpnf, line);
	if(line.length() > 0) {
		stringstream ss1;
		ss1 << line << endl;
		ss1 >> vpns[i].id;
		stringstream ss2;
		getline(vpnf, line);
		ss2 << line;
		vpns[i].text = ss2.str();
		i++;
	}
}
vpnf.close();
if(VERBOSE) {
	cout << "Number of VPN types = ";
	cout << numvpn << endl;
	for(int i = 0; i < numvpn; i++) {
		cout << vpns[i].id << endl;
		cout << vpns[i].text << endl;
	}
	cout << endl;
}

// Read IP vulnerabilities rule file
const char *ivfn = "ip-vuls.txt";
ifstream ivf;
ivf.open(ivfn);
int numip = 0;
while(ivf.good()) {
	string line;
	getline(ivf, line);
	if(line.length() > 0) numip++;
}
ivf.close();
ivf.clear();
ipvuls V[numip];
ivf.open(ivfn);
i = 0;
while(ivf.good()) {
	string line;
	getline(ivf, line);
	if(line.length() > 0) {
		stringstream ss;
		ss << line;
		int ip;
		ss >> ip;
		V[i].ip = ip;
		ss >> line;
		char str[256];
```

```
                strcpy(str, line.c_str());
                char *vrule = strtok(str, ",");
                while(vrule != NULL) {
                        stringstream st;
                        st << vrule;
                        int rule;
                        st >> rule;
                        V[i].vuls.push_back(rule);
                        vrule = strtok(NULL, ",");
                }
                i++;
        }
}
ivf.close();
if(VERBOSE) {
        cout << "Number of involved IPs = ";
        cout << numip << endl;
        for(int i = 0; i < numip; i++) {
                cout << V[i].ip << "\t{";
                for(int j = 0; j < V[i].vuls.size(); j++){
                        cout << V[i].vuls[j];
                        if(j < V[i].vuls.size() - 1)
                                cout << ",";
                }
                cout << "}" << endl;
        }
        cout << endl;
}

// Read link matrix file
const char *lfn = "links.txt";
ifstream lf;
lf.open(lfn);
int links = 0;
while(lf.good()) {
        string line;
        getline(lf, line);
        if(line.length() > 0) links++;
}
lf.close();
lf.clear();
const int N = links;
int L[N][N];
lf.open(lfn);
i = 0;
while(lf.good()) {
        string line;
        getline(lf, line);
        if(line.length() > 0) {
                stringstream ss;
                ss << line;
                for(int j = 0; j < N; j++) {
                        ss >> L[i][j];
                }
                i++;
```

```
		}
	}
	lf.close();
	if(VERBOSE) {
		cout << "Number of possible links = ";
		cout << N * N << endl;
		for(int i = 0; i < N; i++) {
			for(int j = 0; j < N; j++) {
				cout << L[i][j] << "\t";
			}
			cout << endl;
		}
		cout << endl;
	}

	// Read attack rules file
	const char *atfn = "attack-rules.txt";
	ifstream atf;
	atf.open(atfn);
	int numatr = 0;
	while(atf.good()) {
		string line;
		getline(atf, line);
		if(line.length() > 0) numatr++;
	}
	atf.close();
	atf.clear();
	attackrule R[numatr];
	atf.open(atfn);
	i = 0;
	while(atf.good()) {
		string line;
		getline(atf, line);
		if(line.length() > 0) {
			stringstream ss;
			ss << line;
			ss >> R[i].vid;
			ss >> R[i].src;
			ss >> R[i].desti;
			ss >> R[i].destf;
			ss >> R[i].prob;
			for(int iv = 0; iv < numvpn; iv++) {
				int avpn;
				ss >> avpn;
				R[i].vpn.push_back(avpn);
			}
			i++;
		}
	}
	atf.close();
	if(VERBOSE) {
		cout << "Number of attack rules = ";
		cout << numatr << endl;
		for(int i = 0; i < numatr; i++) {
			cout << R[i].strval() << endl;
```

```
	}
	cout << endl;
}

// Define array attacker privilege on all IPs
int priv[numip];
for(int i = 0; i < numip; i++) {
	// Initialize with default privilege
	priv[i] = P_DISC;
}

// Find attacker IP
for(int ip = 0; ip < numip; ip++) {
	for(int iv = 0; iv < V[ip].vuls.size(); iv++) {
		int vid = V[ip].vuls[iv];
		if(vid == V_ROOT) {
			priv[ip] = P_ROOT;
		}
	}
}

// Find available destination IP using link matrix
for(int ip = 0; ip < numip; ip++) {
	int src_ip = IP_NONE;
	for(int i = 0; i < numip; i++) {
		if(priv[i] == P_ROOT) {
			src_ip = i;
		}
	}
	int link = L[src_ip][ip];
	if(link == 1 && ip != src_ip) {
		priv[ip] = P_ACCS;
	}
}

// Initiate first status
status s;
s.cur = 1;
s.prev = -1;
for(int i = 0; i < numvpn; i++) {
	s.vpn.push_back(-1);
}
for(int ip = 0; ip < numip; ip++) {
	s.priv.push_back(priv[ip]);
}
sag table;
table.list.push_back(s);
int prev = 0;

// Status
status ss = s;

// Initial IP
int src_ip = IP_NONE;
int dest_ip = IP_NONE;
```

```
// Maximum can not finding target IP
int max_no_ip = 100;
int no_ip = 0;

// Iterating attack process
int tmax = 10000;
for(int t = 0; t < tmax; t++) {
	while(src_ip == IP_NONE) {
		int ip = rand() % numip;
		if(priv[ip] > P_ACCS) {
			src_ip = ip;
		}
	}

	int ip = rand() % numip;
	int link = L[src_ip][ip];
	while(dest_ip == IP_NONE
	|| dest_ip == src_ip
	|| link == 0) {
		link = L[src_ip][ip];
		if(priv[ip] > P_DISC
		&& priv[ip] <= priv[src_ip]
		&& ip != src_ip
		&& link == 1) {
			dest_ip = ip;
		}
		no_ip++;
		if(no_ip > max_no_ip) {
			no_ip = 0;
			break;
		}
		ip = rand() % numip;
		link = L[src_ip][ip];
	}

	if(src_ip == 0 && dest_ip > 1) {
		cout << prev << " ";
		cout << src_ip << " " << dest_ip << endl;
		cout << L[src_ip][dest_ip] << endl;
	}


	vector<int> rules;
	rules.clear();
	for(int ip = 0; ip < numip; ip++) {
		if(V[ip].ip == dest_ip) {
			for(int iv = 0; iv < V[ip].vuls.size();
				iv++) {
				rules.push_back(V[ip].vuls[iv]);
			}
		}
	}

	// Choose among multipe rules using random
```

```
int Nir = rules.size();
int ir = rand() % Nir;
int rulenum = rules[ir];

// Get appropriate attack rule
attackrule rule;
for(int ir = 0; ir < numatr; ir++) {
	if(R[ir].vid == rulenum) {
		rule = R[ir];
	}
}

// Update status
bool SRC_OK = (priv[src_ip] >= rule.src);
bool DEST_OK = (priv[dest_ip] == rule.desti);
if(SRC_OK && DEST_OK) {

	ss.src = src_ip;
	ss.dest = dest_ip;
	ss.rule = rules[ir];
	ss.prev = prev;

	// Change VPN in status
	for(int iv = 0; iv < ss.vpn.size(); iv++) {
		ss.vpn[iv] += rule.vpn[iv];
	}
	priv[dest_ip] = rule.destf;

	if(t == 0) {
		src_ip = dest_ip;
		for(int ip = 0; ip < numip; ip++) {
			int link = L[src_ip][ip];
			if(link == 1
				&& ip != src_ip
				&& priv[ip] < P_ACCS) {
				priv[ip] = P_ACCS;
			}
		}
	}

	// Change PRIV in status
	ss.priv.clear();
	for(int ip = 0; ip < numip; ip++) {
		ss.priv.push_back(priv[ip]);
	}

	// Check whether it is already have the same or not
	bool NEW_STATUS = true;
	for(int is = 0; is < table.list.size(); is++) {
		status ssi = table.list[is];
		if(ss == ssi) {
			NEW_STATUS = false;
		}
	}
```

```
                //if(NEW_STATUS) {
                        ss.cur = table.list.size();
                        table.list.push_back(ss);
                        prev = ss.cur;
                //} else {
                //      cout << "duplicate" << endl;
                //}

                // Loose destination ip
                dest_ip = IP_NONE;
        } else {
                no_ip++;

                // Loose source ip
                if(no_ip > max_no_ip) {
                        src_ip = IP_NONE;
                        no_ip = 0;

                        while(src_ip == IP_NONE) {
                                int Ns = table.list.size();
                                int is = 1 + rand() % (Ns - 2);
                                ss = table.list[is];
                                int ip = ss.dest;
                                if(priv[ip] > P_ACCS) {
                                        src_ip = ip;

                                        // Get a status from table
                                        for(int ip = 0; ip < numip; ip++) {
                                                priv[ip] = ss.priv[ip];
                                        }
                                        prev = ss.cur;
                                }
                        }
                }
        }
    }

    // Removing duplicate status


    // Show the results
    cout << table.content();
    cout << endl;

    return 0;
}
```