CS442 NETWORK SECURITY
TERM PROJECT REPORT

TOPIC: Port Scan Detector
DATE: 22/05/2018

1.SUMMARY

A port scan attack, occurs when an attacker sends packets to machine, varying the destination port. The attacker can use this to find out what services running, open ports and to get a pretty good idea of the operating system have. Port scan detectors are tools that informs administrator or the user of the machine. Port scan detectors have signatures that are very good at detecting port scan. Obviously it is always possible for an attacker to make his/her attack either very unlikely to be noticed, or very unlikely to be traced to its real origin, while still being able to obtain the port number information. To obsecure the attack, attacker could do it very slowly and hide itself.
This project aims to catch the port scan happening on system by an attacker. **scanlogd** is minimal, reliable port scan detection tool that is used in this project and made following modification for our objectives as descripted below.

2. OBJECTIVES

- Two port detection signatures. Approach triggers then system logs the attack.
- When scanlogd executed it works as daemon on background. Creates a new process then starts logging incoming TCP based port scan attacks silently.
- System logs every source IP that are making attack with the triggered signatures.
- System has the principle of least privileges (PoLP)
- Every user has access to log file. Only user with minimal permissions and root can modify it.

3. REPORT

There are two different approach to signatures used in this project to detect the port scan attack happening on system. Each trigger has its own advantages and disadvantages.

1. Packets following each other coming from same source address to different destination ports within short period of time.
2. Record header of the each packet to a file. Initiate analysis.

3.1 ADVANTAGES & DISADVANTAGES OF SIGNATURES

SIG1 = Packets following each other coming from same source address to different destination ports within short period of time.

Using this signature as trigger is efficient. System has to keep record of the each packet for a short period of time. Storing only the source address and destination port is enough. When we consider the data sizes and amount of packets collected in short amount time, total data size will not be too much that cannot be fit into memory. Storing the data in memory will be very efficient.
This signature is very useful if attacker conducts port scan in very short period of time. Here the real problem is if attacker initiates port scan in very long period of time then this signature most likely will fail because port scan won't be triggered as designed.

```
        …
        // Start timer
        // Store received packets with source address IP and destination port
        // Take the packets in defined time frame and do analysis
        // Group packets that are coming from same source address
        // If same source address is sending too many packets in this time frame then
log it
        …
```

SIG2 = Record header of the each packet to a file. Initiate analysis.

True performance killing solution yet effective. Recording the header of the each packet to file is very costly in terms of performance. Even though writing packets to disk inefficient, conducting analysis on them is can be made in ease and of course analysis also takes some of the resources i.e. writing disk & CPU usage. Port scan will begin from some port that attacker decides then it will go to up or down (random is possible) as attacker wants to (and vice versa) or attacker will try the list of most of the known ports such as vsftp: 5 and ssh: 22. In each case system writes the header of the received packets to file then we do analysis on that file which contains header information. If system sees a pattern that someone from source x trying every known ports or sending packets to many different ports.
Another advantage is that if port scan happens in short or long period of time doesn't matter. In both case system will see the pattern and detect the port scan then log it. Considering the efficiency of this approach, it will be better to use.

```
        …
        // Create file for storing received packets then do analysis on file
        // If same source address is sending many packets to different ports in
ascending, descending or random order then log it
        // If same source address is sending packets to pre-defined most known ports
then log it
        …
```

3.2 BACKGROUND PROCESS – DAEMON

Execution of scanlogd results with creating parent process, forking itself to a child process then exits. After fork and exit operations of parent process, **init** adopts the child process and scanlogd becomes a daemon process that is running on background.

```
if (!fork()){       // scanlogd process executed and it is not daemon (parent)
        fork();  // fork
…
        exit();
}else{          // scanlogd is running as daemon (child)
        …
// act as daemon, initiate port scan detection
        …
}
```

3.3 PRINCIPLE OF LEAST PRIVILEGES (PoLP)

Principle of least privilege's main objective is give user, program or a process minimum amount of resource as possible as that exists on the system for their purpose. For instance in this case scanlogd's required file access can be limited to where it writes the log file. Also process needs access to ports because daemon will listen them.Root permissions will be needed at some point.System will require for root permissions than drop them.

```
…
#define SCANLOGD_CHROOT   "/var/scanlogd-log"     // Directory with limited
access
…
drop_root();   // Drop root privileges after granting required root permissions
…
```

3.4 LOG WITH PERMISSIONS

scanlogd executed by user with normal privileges. Executed program forks to child process with the same privileges.
While daemon process is in execution if system signature triggered then logging occurs. scanlogd creates the log file with the permissions defined below:

- Every user can read the file
- User with minimal permissions can modify
- Root can modify

Daemon running chmod system call changes the created log file's access permissions and it can be as above:

```
        chmod 644 or rw-r--r-- <log.file>
```

```
        #include <sys/stat.h>
        int chmod (const char *filename, mode_t mode);
```

Create log file:

```
        #include <stdio.h>
        #include <sys/stat.h>
        #define FILE_NAME "log.txt"

        // syslog is also an option to log on file
        FILE *f = fopen(FILE_NAME, "w");
        fprintf(f, "TIME \tSOURCE IP \tSIGNATURE");     // Header part of the log file.
        …
        fclose(file_ptr);
        …
        int chmod (const char *f, mode_t mode)          // Set log file's access
permissions
        // or

        …
```

Write to log file when signature(s) triggered:

```
        #include <time.h>
        …
        time_t rawtime;
        struct tm * timeinfo;
        time ( &rawtime );
        timeinfo = localtime ( &rawtime );
        const char *time_now = "[%d %d %d %d:%d:%d]",timeinfo->tm_mday, timeinfo-
>tm_mon + 1, timeinfo->tm_year + 1900, timeinfo->tm_hour, timeinfo->tm_min,
timeinfo->tm_sec;

        // if signature(s) triggered then log to file
        …
        fprintf(f, "%s \t %s  \t %s", time_now, attacker_ip, signatures);
        …
```

i.e. log file:

```
TIME                       SOURCE IP           SIGNATURE
2015-07-16 23:56:17        88.255.245.246      SIG1,
2015-07-16 23:56:17        10.40.4.82          SIG1, SIG2
```

```
2015-07-16 23:56:17          172.221.159          SIG2
…
```

5. CONCLUSION

scanlogd modified to meet the objectives of this project. Principle of least privileges, running as daemon on background, logging when signature(s) triggered and protecting the log file with minimal access privileges. Two different signature triggers implemented. First one is effective if attack is done in short period of time and the second one done well if attacker initiates slow port scanning on system in very long time but very expensive in terms of performance. Each signature has its own advantages and disadvantages.

6. REFERENCES

scanlogd
http://www.openwall.com/scanlogd/

Scanlogd: Port Detection Made Easy
http://pfsensesetup.com/scanlogd-port-detection-made-easy/

Phrack Magazine Volume 8, Issue 53 July 8, 1998, article 13 of 15
http://phrack.org/issues/53/13.html

libpcap
http://www.tcpdump.org/release/

libnet (required for libnids)
http://packetfactory.openwall.net/projects/libnet/

libnids
http://libnids.sourceforge.net/