Global Information Assurance Certification Paper

Design and Deployment of a Rapid Response Security Vulnerability Scanning Infrastructure

Eliot Lim
Submitted for SANS GIAC GSEC Practical
Assignment version: 1.4b, option 2
24 May 2004

1. Abstract

A large research university presents a formidable challenge to computer security professionals. Among the hazards are a completely porous, non firewalled border and decentralized administration of computers.

Considerable emphasis and effort is hence placed on proactive vulnerability scanning in an attempt to reduce risk. This paper will discuss the evolution of a software infrastructure designed to support that effort using mostly open source tools.

Additionally, it will be shown how this infrastructure is used to rapidly respond to emerging threats. A real life success story will be described that will underscore the value of the investment made in this effort.

2. Before snapshot

2.1. Brief history of computer security at a large research university

> No environment is more challenging for security professionals than the modern university where the free interchange of ideas and information is among the most cherished of ideals.
>
> - Quote from SANS '99 brochure[1]

Broadly speaking, the need for the "free interchange of ideas and information" in a type of organization that has existed for approximately a thousand years is currently taking precedence over the need to protect data on a technology medium that has existed for just 30 or 40 years.

In practical terms this represents a nightmare scenario where most of the generally acceptable security concepts and practices are absent. For example:

- A lack of a well defined comprehensive computer security policy
- Lack of enforcement of existing policies
- No risk analysis performed on information assets

[1] http://www.sans.org/sf99/smu.htm

- No security in depth defenses – i.e. No border firewall
- Decentralized administration of networked computers

The following passage is from a paper[2] published by a network administrator at a large research university. It presents a fairly typical view of how firewalls are regarded at these facilities. It is beyond the scope of this paper to discuss the merits of the arguments presented. However, it aptly summarizes the reasons why many large universities do not deploy border firewalls.

> Our thesis is not that all firewalls are evil; rather, it is that all firewalls have significant disadvantages, often ignored, and that their advantages are often overstated. This is especially true of enterprise border firewalls, which are the focus of today's debate.
>
> Can systems be made secure (*network safe*) without using external firewalls? Clearly yes. We have many examples of this. But that seems to be more the exception than the rule, both because most operating systems are not network-safe "out of the box", and because a large number of those systems are essentially unmanaged.
>
> There is unanimous agreement that evil packets should not be permitted to reach a place where they can do harm, so the debate is not over whether to block, but rather *where* the blocking should be implemented, and how to deal with the fact that different people want different things blocked. (*"One person's secure network is another's broken network."*)

The result of this absence of a well planned and executed security strategy is what most professionals in the information security field already know – that modern universities are easy hacking targets. All manner of intrusions are routinely encountered. The sheer volume of incidents frequently overwhelms computer security personnel. For the calendar year 2003, the security operations team at my university recorded a staggering 13,022 security incidents. For confidentiality reasons, details of these incidents will not be discussed.

The biggest risk faced by the University was generally regarded to be a widespread network outage caused by compromised computers overloading the network, typically as virus or worm carriers or distributed denial of service attack zombies. Without the benefit of a defense in depth approach to implementing security, attention was focused on other means to reduce risk.

[2] http://www.educause.edu/ir/library/pdf/erm0319.pdf

2.2. Proactive vulnerability scanning project

The proactive vulnerability scanning project originally started with the premise that security specialists would use well-known open source scanning tools to scan the entire campus and track the vulnerability history of each and every computer. We intended that upon detection of a vulnerability the appropriate owners would be contacted. Later, when new policies were in place the team would compel system owners to patch their systems or face the prospect of being disconnected from the network.

The scanning team consisted of the security operations manager, myself and another security specialist.

2.2.1. Nessus and Nmap

The open source vulnerability scanner *nessus*[3] is a key component of this effort. A lot of the attraction of nessus is the fact that it is open source, not just its operating code but also its ever-expanding database of all known vulnerabilities.

The "nessus vulnerability database" is essentially a collection of individual vulnerability tests called "plugins", written by many individual contributors from the open source community. With this architecture nessus not only keeps track of all known vulnerabilities; it also provides a test for a newly discovered vulnerability in a very short amount of time - typically in a day or less. At the time of writing the nessus vulnerability database contained over 2000 vulnerability tests.

The other tool we frequently used to complement nessus is the well-known port scanning tool *nmap*[4]. Though much of nmap's functionality is already provided within nessus, we found that it was still advantageous to run nmap in isolation. For example, we would run nmap with the -O (OS detect) option and save the results separately into a file before running nessus. In that way we avoid the extra computation of looking for platform specific vulnerabilities in a non-matching platform. We found that almost all vulnerabilities are platform specific and thus by initially pruning the target list we were able to generate scan results faster. Nessus by itself could not avoid looking for a Windows specific vulnerability in a UNIX computer. It would simply attempt the test and return a negative result. Running nmap as a preliminary step allows for nessus to scan a more focused target list.

The maintenance of these open source software tools, including building and upgrading them was my responsibility. I was also responsible for evaluating the

[3] http://www.nessus.org/

[4] http://www.insecure.org/nmap/index.html/

tools and acquiring expertise in using them effectively for our specific needs. I provided feedback and suggested improvements to the various authors of these tools. Additionally, I was responsible for the hardware and the Linux operating system on which they ran. My other team member took on the responsibility of being the public face of this effort while my responsibility was more in the back room designing and building of the scanning software infrastructure.

2.2.2. Early results (part 1)

Early runs using this strategy brought mixed results for the scanning team. One scan in particular involved an attempt to locate ssh daemons vulnerable to a buffer overflow exploit, (CERT® Advisory CA-1999-15)[5]. The team performed a scan to specifically locate this vulnerability across a large university campus of approximately 60,000 computers.

Nmap was run to locate all computers listening on port 22, the traditional ssh port. We then fed this list to two different implementations of the same test.

The test itself was nominally verified against a small number of workstations that were available to the team within the organization. The results of the campus wide scan were naively taken at face value. My team member proceeded to send out advisories warning of potential root compromise to individuals listed in DNS records corresponding to the IP addresses of computers that our test indicated had vulnerable ssh daemons.

A significant amount of negative feedback was received in return, broadly divided into two categories: False positive and incorrect contact person

False positive

Post mortem analysis of this event revealed that the tests for this particular vulnerability involved simply connecting to the ssh daemon, reading its header for the version number and comparing it against a list of predefined vulnerable version numbers. For example, a sshd header announcing itself to be version 1.2.27 would be flagged as vulnerable while version 1.2.28 would not.

It turned out that some highly skilled administrators had simply patched the vulnerability by hand, stitching in the source code changes and leaving the original version number intact. This was done for a good reason: Local enhancements were made to the ssh source to provide additional functionalities. To simply download a newer version of ssh would have meant reapplying all the local modifications to the new version. It was thus a lot easier to simply extract

[5] http://www.cert.org/advisories/CA-1999-15.html

the fixed portion of source code from the new version and apply it to the source code of the old version.

The other groups of people affected were those running packaged ssh. For vendors such as Redhat Linux, the patched executables supplied by the vendor also left the ssh header information intact. Redhat refers to this practice as "backporting[6]"

Incorrect contact person

Many warnings were sent to individuals who were:

- Not the real owners of the computer tested, or
- The correct owner of a computer that had a recycled DHCP address belonging to a vulnerable computer that had previously used the same address.

2.2.3. Early results (part 2)

This early strategy was also used with somewhat better results on the Code Red Worm[7] (CERT® Advisory CA-2001-19) outbreak. My team member supplied me with a list of computers on campus that were listening on the traditional http/https ports and I proceeded to run nessus on them. A total of 502 systems were scanned and 1649 security holes were found. However at this point we were still not completely confident with the reliability of the results. We were also concerned about computers crashing when they were scanned.

2.2.4. Lessons learned from early forays

- Sending out warnings based on inadequately verified scanning results was a bad idea. False alerts rapidly undermined the team's credibility.
- Due to the decentralized nature of a university campus it became clear that maintaining accurate information on computers was a crucial requirement of this project. In particular, we needed owner contact information and its physical location.

---

[6] http://www.redhat.com/advice/speaks_backport.html

[7] http://www.cert.org/advisories/CA-2001-19.html

- On my large university campus there are over 60,000 registered DNS entries. Given this size it would be computationally infeasible to run scans on an as-needed basis whenever a new vulnerability was announced. The results would simply take too long to generate, even with the availability of powerful computers. Target computers often take their time to respond fully to nmap, and nmap makes many different attempts at discovering open ports[8] and resorts to even more trickery (hence increasing the time taken) to do OS detection[9]. We simply would not have enough of a time cushion to run a vulnerability scan, verify the results and alert potential victims.

- Identifying a computer by its IP address is usually adequate when reporting an incident originating from a foreign site. However, due to the use of DHCP, which allows for multiple computers to use the same IP address at different times, it became problematic to identify local computers by their IP addresses. The growing proliferation of laptop computers and their great mobility made this a serious problem. A worm infected laptop could appear in multiple locations across campus at different times of the day and could use multiple IP addresses, depending on where it was plugged into the campus network.

- Results from older scans still had value. For example, we could compare a list of open ports a computer had over time. If a new port suddenly appeared in a fresh scan that information could be used to signal an alert for more scrutiny. The popular scanning tools that we used were handy and very powerful in terms of providing snapshots of a target computer's security status from an attacker's viewpoint, but their ability to archive and correlate historical data was very limited or non-existent.

3. During snapshot

3.1. Solving the DHCP identification problem

The solution that offered the most promise to solving the DHCP identification problem was to tie each IP address at the time of the nmap scan to the computer's MAC address. A pairing of IP and MAC addresses would offer a

[8] http://www.insecure.org/nmap/nmap_doc.html

[9] http://www.insecure.org/nmap/nmap-fingerprinting-article.html

much more positive identification of a particular computer than just IP address alone.

3.1.1. Getting ARP cache data

Of course this would not be possible if we did not have ready access to the ARP cache data containing the IP to MAC address mappings of all the routers on campus. Fortunately we were able to secure the cooperation of the university's network engineers, who were responsible for the smooth delivery of packets throughout the university campus and who had a vested interest in not seeing the network performance deteriorate due to security related incidents. While many individual computers tended to be individually administered (or non-administered!), the main networking infrastructure was centrally administered. Therefore, there was a completely streamlined process of collection, storage and processing of ARP cache data already in place. A trusted server polled all routers four times a day and collected ARP cache data. This data was then processed and organized into human readable files.

3.1.2. Potential complications

There was some discussion in our meetings concerning the potential pitfalls of this scheme. Some departments had elected to deploy a "logical firewall[10]", essentially a stripped down linux computer running a set of iptables[11] rules. A fully ready to install CD of this freeware product had been distributed to and deployed by interested parties around the campus. All hosts behind this firewall implementation have internal non routable RFC1918[12] addresses. The firewall maintained unique world accessible IP addresses for inbound connections that mapped to each host but they would all have the MAC address of the firewall's NIC.

We decided that departments who deployed even rudimentary firewalls were already more proactive and secure than systems that were unpatched and wide open to the world. ***I reminded the team and our managers that the focus of the scanning effort was not to attempt to break the hardest defenses but rather to locate the least protected and most vulnerable computers and to alert their owners***. With the assumption that attackers were typically not targeting specific boxes but rather those that offered the weakest defenses, we decided that MAC addresses that had multiple IP addresses associated with them at a single given moment in time were very likely to be logical firewall hosts

[10] http://staff.washington.edu/corey/fw/

[11] http://www.netfilter.org/

[12] http://www.faqs.org/rfcs/rfc1918.html

and hence we would discard this data on the assumption that these were protected systems that we did not need to worry about.

I performed some data analysis of the ARP cache data to get a feel for how much spurious and bogus data we would encounter. I was also intent in learning about the nature of the data, like how static or dynamic MAC to IP mappings tended to be. I found that approximately 10% of all MAC addresses collected had multiple IP addresses associated with it. These were discarded. The remaining addresses tended to be very static and thus became suitable for our need to uniquely identify computers.

It should be mentioned that nessus provides a means to identify computers by MAC addresses but this feature only works for computers on the same network segment as the scanning computer. - i.e. only if the computer is reachable without traversing a router.

3.2. The need for a computer security database

It became clear that the running of regular periodic nmap scans and the cataloging of the results in a relational database would be a very powerful tool. In addition the database would contain freshly updated contact information and ARP cache information. When an emergency arose a query to the database would yield results much more quickly. This turned out to be a major contributing factor to the success of the overall scanning project.

Though newer releases of nessus do provide a means to keep track of older scan data, it does not provide enough flexibility to incorporate standalone nmap scan data and to allow for the radical step of tying MAC addresses to IP addresses. Basically it was a closed off database that did not allow for external data to be integrated with it. Hence it was quickly rejected for being unsuitable for our purposes.

A key design feature of our database was its ability to track historical data. For each network device scanned, the database would have the ability to compare the list of open ports discovered now with that discovered at the previous scan. Since MAC addresses were recorded, this database also had the ability to track the physical movement of a computer as well as its OS history.

3.2.1. Database design and implementation

A commercial database, IBM's DB2[13] was used for the backend. This choice is simply due to the local availability of expertise and existing backup infrastructure in my environment. There is nothing inherent in this scanning infrastructure

[13] http://www.ibm.com/software/data/db2

design that prevents the use of open source DBMS products like *MySQL*[14] or *postgreSQL*[15].

I installed and configured IBM DB2 version 7.1 on our Linux system. I enlisted the help of another engineer in my team who had extensive experience in database design and operation. He was also quite experienced with the IBM DB2 product. Our team held several database design meetings with our guest engineer in attendance. We briefed him on our data management needs and he in turn guided us in the design and specification of the database schema. I did the eventual coding of the actual schema.

Our database contained these tables:

- The `device` table holds information pertaining to the network device, tying its `MAC` address with a SQL generated unique device id `dev_id` that serves as a key to the other tables. The device table allows for the device to have different IP addresses associated with it, and stores the last seen IP address in `last_ip`. A `restrictions` field is defined to store scanning restrictions if applicable. The `created` field is used to record when the particular device entry was first created. The SQL definition is listed below:

```
CREATE TABLE DEVICE (
dev_id INT NOT NULL GENERATED ALWAYS AS IDENTITY PRIMARY
KEY,
mac VARCHAR(32) NOT NULL,
last_ip VARCHAR(32) NOT NULL,
isup SMALLINT DEFAULT 0,
lastseen TIMESTAMP DEFAULT CURRENT TIMESTAMP,
created TIMESTAMP DEFAULT CURRENT TIMESTAMP,
restrictions SMALLINT DEFAULT 0
);
```

- The `openports` table contains open port information for each port open on `dev_id`. The `instance` field is incremented each time the port is seen, hence recording port history information. `port` and `protocol` would be used to record the port number and its protocol (tcp or udp). `scannum` is used to record the scan event number. The SQL definition is:

```
CREATE TABLE OPENPORTS (
dev_id INT NOT NULL,
scannum SMALLINT NOT NULL DEFAULT 0,
port SMALLINT NOT NULL,
instance SMALLINT NOT NULL DEFAULT 0,
protocol VARCHAR(4) NOT NULL,
PRIMARY KEY (dev_id,scannum,port,instance,protocol)
```

[14] http://www.mysql.com/

[15] http://www.postgresql.org/

```
);
```

- The `idscan` table contains identity information for the particular `dev_id`. There is an `ip` field to record the IP address of the device at the time the nmap scan was performed. Additionally its fully qualified domain name, `hostname`, is recorded. Nmap provides verbose OS information from its –O option. I decided that we needed a condensed summary of OS such as "linux" or "MS". This summary OS type is stored in `os_short` while the full OS identity string returned by nmap is recorded in `os_full`. Physical location and contact person information is stored in `location` and `contact` respectively. The SQL definition is listed below:

```
CREATE TABLE IDSCAN (
dev_id INT NOT NULL,
scannum INT NOT NULL,
ip VARCHAR(32) NOT NULL,
hostname VARCHAR(128),
scandatetime TIMESTAMP DEFAULT CURRENT TIMESTAMP,
os_short VARCHAR(8),
os_full VARCHAR(256),
created TIMESTAMP DEFAULT CURRENT TIMESTAMP,
location VARCHAR(128) DEFAULT NULL,
contact VARCHAR(128) DEFAULT NULL,
PRIMARY KEY (dev_id,scannum)
);
```

3.2.2. Database tools development

Following the design of the database, we wrote software to interact with it.

1. Distributed nmap scanner

My team member wrote a distributed nmap wrapper script whereby nmap is invoked with a certain set of options to scan a single subnet. The wrapper script dispatched multiple instances of nmap to run at the same time to fully utilize processor cycles. The other main feature that the wrapper script performed was to direct nmap output into individual files named after its target subnet. All of these files were stored in a directory named after the date of the nmap scan. For example, a directory named "101503" would contain nmap output files called "10.2.227".

2. load_nmap_data

I wrote this program that essentially takes raw nmap data, processes it and loads it into the database. Data is processed in the following phases:

- Read nmap output files created by the nmap wrapper program above;

- Download ARP cache data;
- Associate MAC addresses from the ARP cache data with IP addresses and hostnames collected by the nmap scan;
- Condense nmap OS identification strings into a simpler form like “linux”, “MS” and “BSD” compared to `”Linux 2.5.25 - 2.5.59 or Gentoo 1.2 Linux 2.4.19 rc1-rc7)”, “MS Windows XP Pro Version 5.1 Build 2600”` and `“OpenBSD 3.0 (x86 or SPARC)”` respectively. This allowed for coarser grained queries and the ability to handle the somewhat chaotic list that nmap sometimes generate; and
- Generate SQL commands to directly load the processed data into the database

3. Simple database query programs

I wrote the following simple text based programs to query the database for specific information and display that information.

- `get_dev_id_by_ip` display all dev_ids matching an IP address
- `get_macip_by_os_port` display all mac and IP addresses matching a particular OS with a particular port open
- `put_contact_by_ip` Input contact information associated with the particular IP address.
- `put_contact_by_subnet` Input contact information associated with the particular subnet.

3.3 Database population

With all these tools in place we then proceeded to run weekly nmap scans and have its output automatically processed loaded into the database. If necessary we could run this scan and store infrastructure daily but given the intrusiveness of running nmap in such a widespread fashion we decided on weekly intervals.

The IP and subnet contact information were filled in independently when more accurate information were available compared to those maintained by the DNS administrators.

3.4. Nessus tuning

Very early on it became clear that unless nessus scanning was performed in a highly distributed and independent manner (i.e. run on multiple computers with a divided target space) the computing requirements would exceed the resources available to the team. It would take typically 30 minutes or more to do a full nessus scan of a single /24 subnet with approximately 100 live hosts. There were over 3,000 subnets of varying sizes in the entire university campus. The total number of unique MAC addresses that had been recorded in the scan database was approximately 80,000.

I explored options at fully optimizing nessus for maximum performance. I felt that the parts of nessus that made it user friendly to the casual scanner also created performance and reliability bottlenecks. I attempted to strip away as much of the user-friendly interface as much as possible in order to gain maximum performance. The following is the evolution of the steps I took to improve nessus performance:

- **Scan only for the vulnerability of the day.** Though there are thousands of known vulnerabilities, it was typical that only a single vulnerability would cause mayhem at any one time, rather than many at once. Given that the purpose of this effort was to rapidly respond to an imminent threat, we decided that the effort should also focus on a single "hot" vulnerability though the strategy was open ended enough to handle multiple vulnerabilities, but at a cost of reduced responsiveness. Thus instead of unleashing 2,000 vulnerability tests on a single computer we would only do a small handful, typically no more than 5, but on a very large number of computers.

- **Completely bypass the X windows based user interface.** Early versions of nessus would occasionally crash in the middle of a scan and lose all of the scan results. Many crashes were traced to X11 server problems. Later versions of nessus featured running nessus in "batch" mode and the ability to save intermediate scan results[16]. I explored these new features but I found them to still not be completely satisfactory in terms of reliability, flexibility and performance.

- **Completely bypass the client/server architecture, authentication mechanism.** Nessus is built on a client/server model intended to allow one main scanning server engine to serve multiple scanning clients running their own user interfaces. To support this a security infrastructure is provided to encrypt the traffic as well as to properly authenticate clients. We found that if we used nessus in command line mode or "batch" mode[17], client authentication was still required. Furthermore the password had to be typed

[16] http://www.nessus.org/doc/session_saving.html

[17] http://www.nessus.org/doc/nessus.html

out in the clear. I felt that this was unnecessary and cumbersome. Hence I bypassed this scheme altogether.

- **Completely bypass the long complicated configuration file with its complex options.** Nessus documentation is constantly improving but its structure and layout of options has always been complex and confusing. In order to specify just one specific test to be run, one had to either use the user interface, which has improved tremendously in recent releases, or one had to manually edit the long *nessusd.conf* configuration file. I found this to be cumbersome to use and bypassed it.

- **Run the nasl interpreter directly** on the nessusd host, and run it in parallel. At the heart of nessus is the *nasl* [18](nessus attack scripting language) interpreter and the thousands of open source vulnerability tests written in nasl and contributed by programmers all over the world. Eventually all the steps of stripping away unneeded features of nessus described above led to this innermost core where the potential for maximum performance lay. Having stripped off nessus features to this point can be considered to have departed significantly from the conventional way of running nessus. The next section will be used to describe this deployment.

3.5. Design of a nessus based high speed vulnerability scanner

To fully utilize powerful CPUs and wide bandwidth to scan large numbers of computers parallelism is required. This is because target computers may not respond instantly to a test, and the time taken to wait for a test to complete can also be used to test additional computers or to launch different tests. The following are design highlights and issues of this scheme.

- **Parallel scanning algorithm**

[18] http://www.nessus.org/doc/nasl2_reference.pdf

The following pseudo-code is based on real code that I wrote using the scripting language *rexx*[19]. The pseudo-code presented here, however, can be implemented in just about any programming language of choice.

```
Max = max # of scans to run in parallel
Threads = max  /*initially */
While list still contains hostnames
      Do threads
            Get the next hostname in the list
            Run_single_scan using nasl on hostname &
            /* run it in the background! */
      End
      Sleep 2 seconds
      threads = Max – count_threads_still_running()
End
Exit


Function count_threads_still_running()

Return the result of: ps –ef | fgrep "nasl" | wc –l
```

From the pseudo-code it can be seen that an algorithm to devise a self-tuning parallel scanning program is fairly trivial. The code has a predefined upper bound on the number of scanning threads. The individual scans are launched in the background so each call returns immediately after a scan is launched. The inner loop thus exits quickly after launching its quota of individual scans. After a short time the operating system is queried via the unix *ps* command to determine how many individual scans are still running.

This number of uncompleted scans is deducted from the next batch to be launched so that we do not overwhelm the scanning computer with excessive processes. For example, in the first batch 200 parallel scans are launched and after 2 seconds 130 have completed, leaving 70 still running. Therefore for the next batch only 130 are launched to bring the total number of simultaneously running scans close to the predetermined maximum of 200.

After the 2 second interval a query is submitted again and perhaps this time 160 scans would have completed and so forth. In this way the program can be tuned to keep the CPUs 100% busy yet at the same time not overwhelm the system.

The maximum number of threads and the polling interval was initially randomly selected and then fine-tuned via trial and error by observing system load using the unix *top* command. The numbers used in the example are the actual numbers of 200 threads and 2 second polling intervals used on our scanning

[19] http://regina-rexx.sourceforge.net/doc/fregina-00.html

computer, an IBM Netfinity, with twin 1GHz Intel Pentium P3 processors, 512MB of memory and the Redhat Linux version 7.3 operating system.

- **Output to file**

Each thread would either terminate quietly if the host tested negative or would create a file named after the target host if it tested positive. By writing the results of a parallel program to individual files, I could avoid the tricky timing and synchronization problems that typically plague the parallel programmer. Because each thread ran independently and did not care or depend on the results of other threads the problem was tremendously simplified compared to traditional multithreaded and parallel programs. I simply had to wait until I saw no more threads running to know that the program had finished running.

Having hosts that tested positive represented by a file named after them made it easy to create lists of vulnerable hosts, simply by using the unix *ls* command.

To have the scan results written to a file named after the target box, it was necessary to wrap the nasl command in another script. The pseudo-code is presented here.

```
/* read hostname in from the command line */
hostname = read_command_line_arg()

/* define output filename as /tmp/<hostname> */
output_filename = /tmp/ || hostname

/* perform the actual test on hostname */
/* negative result will result in no output */
/usr/local/bin/nasl -t hostname \
/usr/local/lib/nessus/plugins/msrpc_dcom.nasl > output_filename

if empty_file(output_filename) then
      delete output_filename
```

- **Additional tweaks**

A little bit of tweaking was required on the nessus plugin before it would work correctly if invoked directly by the nasl interpreter. This involved removing code that interacted with the nessus “knowledge base” which is the mechanism that nessus uses to track historical scan information. By running nasl directly, the knowledge base is not properly initialized and calls to it within the plugin would generate error messages.

- **Unreachable targets**

A further advantage to this scheme was that the nasl interpreter eliminated hosts that were unreachable because they were firewalled, powered down, or unplugged. Hosts that did not respond would simply return a test result of "negative". We took an attacker's perspective that a potentially vulnerable computer that is powered down is not considered vulnerable. We did not think that it would be possible to discover every single vulnerable system; rather we were aiming to minimize the number that got attacked.

- **Real world performance**

We were extremely pleased with the efficiency gains of the multithreaded script compared to stock nessus. In the real life scenario to be described in the "after snapshot" section, it was able to scan almost 20,000 computers in just 20 minutes for a single vulnerability.

4. After snapshot

4.1. Deployment example in responding to a serious incident (MS03-026 DCOM "Blaster" worm outbreak)

Our scanning infrastructure was deployed with great success for this particular incident. Compared to older vulnerabilities, this incident evolved very rapidly from the discovery of the vulnerability to the development of a rapid spreading worm in just 3 weeks. In the past many months could go by before a discovered vulnerability was exploited. Fortunately our team managed to make a significant impact in preventing many computers from being infected. The chronology of events is as follows:

7/16/03 The "research" group "Last stage of delirium"[20] announced the discovery of a vulnerability in Microsoft's RPC DCOM Interface. On the same day Microsoft issued advisory MS03-026 [21] and a patch labeled "critical".

7/17/03 The nessus site published a plugin[22] test for this vulnerability. This plugin checks the Windows registry for the presence of the MS03-

[20] http://lsd-pl.net/

[21] http://www.microsoft.com/security/security_bulletins/ms03-026.asp

[22] http://cgi.nessus.org/plugins/dump.php3?id=11790

026 patch and requires Administrator privileges. Unfortunately due to the decentralized nature of university computers this test was of very limited use to us.

7/25/03 Another hacker group Xfocus.org[23] published the first exploit in its web site. This exploit required the attacker to specify the version of Windows of its target (XP or 2000) and which patch level was applied. A total of 2 XP service pack levels and 4 of W2K were exploitable.

This initial exploit had a limited impact because specifying the wrong Windows version and/or the wrong service pack level caused the target system to simply crash. That attacker had no better than a 1 in 6 chance of a successful exploit. However this was an alarming development and it led to a high state of alert for the security team.

7/26/03 The security products company eeye[24] released the first vulnerability scanner for MS03-026. The free version that was available only allowed scanning a subnet at a time and only ran on the Windows platform. I tested this scanner and concluded that it was not sufficiently powerful or efficient to scan the entire university campus in a reasonable amount of time.

7/29/03 The nessus site released a new test[25] based on the eeye scanner. Unlike the test released earlier this test did not require Administrator privileges. I tested it immediately on those computers that belonged to our organization where we had the ability to login, verify the patch status and thus verify the accuracy of the test. The initial results were very disappointing, with a 50% false positive rate. I decided that this test would not be sufficiently accurate to be deployed for the entire campus.

7/31/03 A strongly worded advisory was sent to 75,000 email recipients across the entire campus by our computing directors, warning all computer users of this problem and urging them to patch their systems immediately. Many administrators heed the warning and patch their systems. Based on our previous experiences, however, we felt that there would be many more systems that would have been unpatched.

8/2/03 Several popular TCP ports for Windows services, including the MS

[23] http://xfocus.org/advisories/200307/4.html

[24] http://www.eeye.com/html/

[25] http://cgi.nessus.org/plugins/dump.php3?id=11808

RPC DCOM port 135 were blocked bidirectionally at the border routers.

8/5/03 The first worm appeared but it was based on the proof of concept Xfocus.org code, which required prior knowledge of the patch level and OS version of its target. I spent time studying the Xfocus.org exploit code, the nessus plugin test and the vulnerability paper in an attempt to better understand the risks and the technicalities of the exploit.

8/6/03 The hacking group "oc192"[26] released a significantly improved exploit. This exploit[27] was now patch level independent. The code was simply called "oc192.c". It still required the attacker to specify the target as either Windows XP or W2K. I tested this exploit on a test system and found that it was very effective.

At the same time the author of the nessus test had been working hard to further refine the accuracy of his test. Many revisions were issued and I watched the nessus site daily and downloaded each fresh release of the test and tried it. I found the rapidly evolving test to show steady improvement but it still did not give us the accuracy that we needed.

8/7/03 It occurred to me that I could create a very high confidence test from the oc192.c exploit code by removing the portion of code whereby a shell on the exploited computer was spawned after it was compromised. I explained to my managers that this amounted to "opening the door but not entering, and then shutting it". Given the requirement of a high confidence test, my managers instructed me to proceed. By removing a small portion of code, the oc192.c exploit was converted into a vulnerability test that was virtually 100% accurate.

The remaining problem that I had to solve was to properly identify the version of Windows that the target computers ran: XP, W2K or some other variant like NT4. I did not have the luxury of improperly identifying a target system and crashing it. The nmap OS identification fingerprinting did not have sufficient granularity to resolve between XP and W2K. After some discussion with other engineers, I arrived at the following ad hoc algorithm:

- Use *netcat* to determine if port 5000/tcp is open, if so identify it as XP. The syntax used was: `nc -w 5 -v -z <hostname> 5000`.

[26] http://www.oc192.us/security.html

[27] http://www.oc192.us/projects/downloads/oc192-dcom.c

This was somewhat simplistic but it turned out to be a very good test.

- Use the *smbclient* command to establish a null session with the target and parse the output for Windows version. The syntax used was: `smbclient -L <hostname> -N -p 445`. Output from `smbclient` would typically look like:

```
# smbclient -L target.com -N
added interface ip=10.12.13.46 bcast=10.12.13.63
nmask=255.255.255.224
session request to TARGET.COM failed (Called name not
present)
Anonymous login successful
Domain=[XXX] OS=[Windows 5.1] Server=[Windows 2000 LAN
Manager]

Sharename Type Comment
--------- ---- -------
Error returning browse list: NT_STATUS_ACCESS_DENIED
```

Enough information was returned in many cases to determine the OS version even though full access privileges were not available. In the above example the token that my script looked for was "`OS=[Windows 5.1]`". Windows 5.1 is XP, while 5.0 is W2K and 5.2 is Windows 2003 Server.

Some systems that were properly secured would not answer to this query, though these were in the minority. Typical output would look like:

```
# smbclient -L fort.knox.gov -N
added interface ip=10.12.13.46 bcast=10.12.13.63
nmask=255.255.255.224
timeout connecting to 10.12.17.23:445
Error connecting to 10.142.17.23 (Operation already in
progress)
Connection to fort.knox.gov failed
```

I felt once again that we should concentrate on unmanaged, unpatched and weakly secured systems rather than those whose owners were more proactive. We decided that a Windows system where the null session was properly secured would also have a good probability of having a capable administrator. Therefore, if I was unable to make a determination of the exact OS version, my script would abandon the target.

The two step approach was taken once again for speed. It was faster to use netcat to check for port 5000 than it was to use smbclient.

Lastly, I was also not able to reliably test Windows NT for the vulnerability using the oc192 code. Since the most dangerous exploit code that had thus far been released did not have an ability to exploit Windows NT I also decided to disregard potentially vulnerable NT systems for the time being and concentrate on just XP and W2K. We found out later that even though Windows NT was vulnerable to the same bug, the exploit code written for W2K and XP would not work at all on NT because their internals were too different.

8/8/03 to 8/9/03

With the final technical challenges resolved I was now ready to scan the entire campus. From our database of nmap data, I obtained a list of about 20,000 computers that met the following criteria:

- Running some version of Microsoft Windows
- TCP port 135 open to internet

The vulnerability scan was divided into 2 phases. In the first phase nessus was used to do the screening. This initial screening accomplished multiple objectives:

- Since it was a safe test that did not crash the target system it could be deployed using the parallel scanning engine described in the previous section. It took no more than 20 minutes to do the first phase scan of 20,000 Windows computers.

- The first phase scan eliminated systems that were unavailable, powered down, or simply were not listening on port 135, such as older versions of Windows like ME and 98.

Scan results:

The first phase narrowed down the list of targets to 5205 listening on port 135 during the time of the scan. 1102 were found to be "potentially vulnerable". These I subjected them to the much more dangerous oc192 code after identifying them as either XP or W2K. They were not scanned if a positive identification was not obtained.

- 112 systems were positively identified as XP. Out of these, 102 were found to be vulnerable with virtually 100% confidence, 10 were found to be false positives that were identified by the nessus test.

- 484 systems were positively identified as W2K from which 442 were found to be vulnerable with virtually 100% confidence while 23 were nessus false positives.

- The remainder was NT4 or systems that no longer answered on port 135.

I ran another test to verify that the first phase nessus test did not produce false negatives. i.e. that an unpatched system did not test negative by nessus. I ran the dangerous oc192 test on these systems and found that nessus did not produce any false negatives. That meant that the two-phase approach to scanning was sound.

Post scanning notes:

- While it was possible to also scan the 1102 systems in parallel, I decided to only scan them serially (one at a time) because of the danger involved in overflowing buffers. If something went wrong, it would have been much easier to terminate a sequential test than if several hundred were launched to run together. The conservative approach meant that not a single system tested crashed, and every system that was identified as vulnerable was indeed found to have been vulnerable.

- In a post incident presentation I was asked why for a campus of 60,000+ registered DNS names and 80,000+ unique MAC addresses that had been recorded in our scan database only 19,850 computers were scanned. The reason is that the 80,000+ MAC addresses were recorded over a period of months by weekly scans. When the time came to scan, only a fraction of the systems that had been seen in prior months were up and running, and out of these only another fraction were Windows computers. We believed that since this outbreak occurred during the summer break, many students were away and hence did not have their computers connected to the Internet from campus.

- The oc192.c modified exploit was by far the most accurate test, since the blaster worm is actually based on a scanner wrapped around the oc192 exploit. However, the two-phase test would be unnecessary if we had a higher tolerance for false positives such as in an environment where we have Administrator access to all systems tested. Running just the first phase using the safe nessus test to generate a list of potentially vulnerable systems and then verifying them with Administrator access would have been a less risky and easier strategy if it was possible.

- This infrastructure was deployed again to scan for the related vulnerability MS03-039. Doing the first phase nessus scan in parallel took 45 minutes for approximately 27,000 hosts.

5. Conclusions

Proactive vulnerability scanning on a large scale is a much more challenging endeavor than it first appears. However given the difficult security challenges in our environment, we pursued this solution which we felt had the greatest potential for risk reduction. We tackled the technical challenges head on and eventually became successful in achieving our goal.

6. Bibliographic References

1. Gray, Terry. “Firewalls: friend or foe.” Educase Review Jan 2003 16 Feb 2004
   URL: http://www.educase.edu/ir/library/pdf/erm0319.pdf

2. Anderson, Harry. “Introduction to Nessus.” Security Focus Infocus 28 Oct 2003. Feb 16, 2004
   URL: http://www.securityfocus.com/printable/infocus/1741

3. *fyodor*. “Remote OS detection via TCP/IP stack fingerprinting.” 11 Jun 2002. 16 Feb 2004
   URL: http://www.insecure.org/nmap/nmap-fingerprinting-article.html

4. Hoffman, Forrest. "An Introduction to Parallel Programming." Linux Magazine Mar 2002. 16 Feb 2004
   URL: http://www.linux-mag.com/2002-03/extreme_01.html

5. Arboi, Michel. "The NASL2 reference manual." 6 Feb 2002. 16 Feb 2004.
   URL: http://www.nessus.org/doc/nasl2_reference.pdf

6. Freeman, Wayne J. "An Analysis of the Microsoft RPC/DCOM Vulnerability MS03-026." 22 Sep 2003. 16 Feb 2004
   URL: http://www.inetsecurity.info/downloads/papers/MSRPCDCOM.pdf

7. *author unknown.* “Information Security in the Modern University: Is it mission impossible?” SANS Security Conference San Francisco 99 brochure 1999
   URL: http://www.sans.org/sf99/smu.htm