Implementing Vulnerability Scanning in a Large Organisation

Richard Grime

GIAC Security Essentials Certification (GSEC) Practical Assignment
Version 1.4b (Option 2)

Implementing Vulnerability Scanning in a Large Organisation

Richard Grime
June 2003

Abstract

This paper describes how the security group in our organisation uses Vulnerability Scanning to demonstrably improve our security posture. This covers the reasons and requirements for scanning, how this fits with our current business structure and how we used a web interface to distribute the collected data to our system custodians.

Also covered are our techniques for dealing with false-positives, an explanation of the chosen solution and how the system was tailored to operate from an end-user perspective. Finally, we discuss the impact that the system has had on our organisation.

Introduction

Vulnerability scanning in itself is now very much a "point and click" affair. Scans are available freely on the web [1] and the majority of tools are provided with a graphical interface. However, implementing vulnerability scanning within a large organisation is about much more that installing the software and running it.

This paper describes how our security group now uses vulnerability scanning to demonstrably improve the security posture of our organisation.

The need for Vulnerability Scanning

We are a large academic institution where large chunks of our internal network can be considered "untrusted" – as we allow students to use their own laptops in the library and within halls of residence. Some departments are now also operating wireless network access for their researchers. We also offer VPN and dial-up services to our users from their homes. In addition, we offer internet access to academic visitors and to those using our conference facilities.

Even without these other attack vectors, perimeter security is not enough. An academic firewall must support the very varied activities of our researchers – video conferencing, GRID programmes [2] and collaborative research all require firewall exceptions. Hence, we must apply the principle of Defence in Depth and protect our host systems.

Scanning in itself is not a policy enforcement tool, but it does provide us with the necessary information to ensure that we can keep our hosts safe from known attacks. Other security problems, like weak passwords or poorly coded web sites are a matter for different tools – we do not attempt to address them using this system.

In a world of metrics and performance reports, vulnerability scanning is also a useful way of tracking our internal security posture over time. We can use "percentage of vulnerable systems" type statistics to compare departments, demonstrate need for budget / staff and show management that the security group is achieving its mandate.

The lack of any internal security data meant we had no way of knowing the risks we faced from internal attack. This is a problem faced by many companies – as illustrated by Paul Simmons of ICI; "Our biggest problem is knowing what we do not know." [3]

As stated by CERT-CC in their best practices, a system administrator is required to do the following:

> "Whenever an update is released, you need to evaluate it, determine if it is applicable to your organization's computers, and, if so, install it." [4]

We do not wish to rely solely on inexperienced administrators making this kind of decision which could affect the entire organisation. Vulnerability scanning provides a way of double-checking the decisions of our administrators.

By scanning IP ranges, rather than known hosts, the scanning engine also provides an asset discovery mechanism. This is a key part of managing risk, as this will identify the risk for *every* system, not just the ones we know about.

Leveraging the existing business structure

As part of our organisational security policy, we already had in place a tiered structure of personnel who had been made responsible for the security of their systems, known as "custodians". The custodians are, in the main, of basic technical competency. They report to departmental security liaisons who are in themselves responsible for annual departmental risk assessments.

There are also our central support teams, from first to third line. These teams interact with the users and custodians on a regular basis.

Vulnerability scanning some 10,000 hosts on a regular basis generates a lot of data. No central resource has the time or personnel to sift this information and then distribute it out to each custodian.

To utilise this valuable resource, we needed a way of supplying vulnerability scan data to the custodians in a simple and cross-platform fashion.

Only by demonstrating that the data collected can be of real benefit will our user community accept the concept of vulnerability scanning.

Defence in Depth and Risk Assessments

As mentioned above, vulnerability scanning itself is an important part of any Defence in Depth strategy. Our network perimeter is policed by firewalls and IDS systems and key internal groups are protected by internal firewall solutions.

The next step in this chain is host security. Identifying vulnerable hosts through scanning is an obvious benefit when attempting to secure them. So how does vulnerability scanning fit with the SANS four step plan [5] to risk management?

Step 1: Identify Risks
A vulnerability report is an excellent way of identifying the risks and attack vectors open on each host.

Step 2: Communicate your findings
This is the critical step: we must be able to pass the information down to the people who can actually effect changes – i.e. our custodians.

Step 3: Update (create) policy as needed
Policy already exists granting our group permission to vulnerability scan systems where the users have been properly notified beforehand. The policy covering the requirements of each custodian (their "job description") may need updating in response to data collected – e.g. we may have to change the frequency with which the custodian checks the machine if it turns out that patches are being released faster than they are being applied.

Step 4: Develop metrics to measure compliance
Using vulnerability scanning, it is easy to tell whether the custodian is doing their job properly. By scanning regularly, the overall security posture of the organisation can be measured. We can also look for successful custodians and use their experience to train others.

Getting Started

We can now summarise the requirements for our organisation

Requirements
1. A vulnerability scanning engine capable of scanning multiple OS platforms (our organisation uses a mix of Windows, Linux, Unix, MacOS, etc.)
2. A low cost solution
3. Ability to disseminate data to the authorised parties
4. Ability to secure data from unauthorised parties (who may be authenticated)
5. Ability to track data over time
6. Ability to generate reports
7. Tie in to our existing databases – e.g. computer registration database

Table 1 - Requirements

Vulnerability scanners are very expensive – with the notable exception of Nessus. As many parties consider Nessus to be amongst the best of the vulnerability scanners [6], and fits very well with the "low cost" requirement, it is the obvious choice.

The storage, distribution and reporting of data requirements immediately suggests "database". We have a large Windows infrastructure within our organisation – hence it was decided to use MS-SQL to backend the data collected during the scans. MySQL may seem a more logical choice (as Nessus connects directly to MySQL), but the Microsoft solution was chosen for two main reasons:

1) We already have existing MS-SQL database servers suitable for our use; these are regularly backed up and maintained by our database team.

2) Our experience with MySQL is limited. We needed to make sure that security on this database was absolute – the data contained within it is practically the keys to the kingdom.

MS-SQL also offers data transformation services (DTS), which makes integration with existing data sources much easier.

Setting up the scanner

Using the excellent documentation at the Nessus site [7], the daemon portion of Nessus was setup on a dual processor Linux server. NessusWX [8] was installed on a Windows server running MS-SQL.

NessusWX is used to initiate scans by department. We defined groups of subnets to cover each organisational unit so that the scans can be broken down easily. Hence, if we are asked to manually rescan a unit, we are not forced to cover the whole department again.

A database was created to hold the Nessus data. NessusWX can be used to generate SQL scripts containing all the data collected during a scan. Running these scripts in the Microsoft SQL Query Analyser imports all the data into the database where it is ready for distribution.

Both the Windows and Linux boxes are hardened using the tips provided in the SANS material - which includes Centre for Internet Security (CIS) [9] guidelines for each OS. In addition, the Windows machine runs BlackICE PC Protection by ISS [10]. The Linux machine runs IPTables as its host firewall.

Integration with existing da ta sources

Our computer registration database holds the IP, MAC address and location of each computer we have registered on our network. More importantly for this case, it also holds the registered owner and registered custodian for those systems. DTS is used to pull this data on a regular basis, so that the user and custodian fields are kept in sync between the two databases.

Building a user interface

We now had a database with some initial scan data in. Next, a user interface to that data needed to be created.

Many security products use monolithic management interfaces that would be unsuitable for our users. The interface needed to be quick and accessible for non security trained personnel.

The obvious choice (bearing in mind the cross-platform nature of our organisation) was a web based interface. As my background is in ASP web development, this was the language chosen. Our organisation also operates an Active Directory infrastructure, which means that authentication can be handled centrally.

We needed now to define the types of users who would be accessing the interface and what level of access they would require – following the principle of least privilege. All access to the data is, by design, read only

User Type	Access Required
System Owner	Systems who have the owner field set to that person's logon name*
System Custodian	Systems who have the custodian field set to that person's logon name*
Departmental Support Staff	Systems within that department, access to departmental reports.
Central 3rd Tier Support Staff	Access to all systems under their remit, access to reports covering only those systems
Security Group	Access to all systems, access to database directly, access to all reports

Table 2 - Access Control Types

*i.e. where the field from the computer registration database shows the currentl y logged on person as an owner / custodian. See "Integration with existing data sources".

The IIS server hosting the web interface is a separate machine, which means the ASP ADO connection string must be set to connect through the SQL server's host firewall [11,12]

Making the user aware

The interface also needed to promote the role of custodian to owners who weren't aware of the custodian scheme. Some of our systems do not have custodians assigned. However, the system owners would be able to see this and chase up getting a custodian assigned. Hence, the interface was designed to break down into three main sections as follows:

- *A list of machines that the current user is the registered owner of.* This provided basic details, such as who the custodian was, the IP address and name of the machine and the last recorded OS for that machine.

- *A list of machines that the current user is the registered custodian for.* The same basic details were provided as above, including who the owner of each system was.

- *An "additional access" dialog.* For the users who had access to additional data (i.e. departmental support staff, etc.) a box was supplied into which either an IP or hostname could be supplied. Provided the user had access to that particular machine, its vulnerability scan data would be displayed.

Screen shots demonstrating this system are given in Appendix A.

The first two sections are lists for a very specific reason. It highlights to the user *exactly* which systems they are responsible for - if our information regarding who the owners / custodians of particular systems are wrong, the user will be able to tell immediately. This helps us keep the databases current.

The additional access section caters for the support staff who are assigned access to specific subnets, etc.

The interface also includes a "Traffic Light Report". Systems with major vulnerabilities are marked with a red light; those with no major holes are given a green light. As the overall security posture improves over time, we can tune the traffic lights to cover less serious vulnerabilities.

This helps the busy user see, at a glance, which systems need immediate attention without having to check each system in detail.

The database is also used to log the actions of all the users accessing the web interface. This helps us track abuse of the system, as well as troubleshoot problems.

False Positives

It is an artefact of any scanning solution that "false positives" will be produced. An obvious example is that of UNIX system administrators who run PortSentry (David Sarmanian has a paper on what PortSentry is and how to deploy it [13]). In other cases false positives may also be generated because of misleading banners.

Our Nessus scanner is configured to use safe tests* (Non-DoS plug-ins in Nessus parlance). This means that in many cases, Nessus relies on the "banner" returned when it connects to a specific port. For example, it may use a test similar to this telnet connection:

```
[root@localhost root]# telnet localhost 22
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
SSH-2.0-3.2.2 SSH Secure Shell (non-commercial)
^]
telnet> qu
Connection closed.
```

In this case the banner would be "SSH-2.0-3.2.2 SSH Secure Shell (non-commercial)". The Nessus plug-in knows which versions of SSH are insecure, and so can flag problematic versions. However, patches released by some vendors may not modify this banner string.

Given the likelihood of false positives occurring, we had to implement a way that users could report them to make sure they were filtered out of future scan reports. As the system is based on a database this is fairly easy to do and is explained in the next section – "Database Model".

Why is handling false positives correctly so important? Any data we present to a user must be as accurate as possible. Imagine a user faced with a long list of false positives, which are showing up every time we scan their system. After a while, they will begin to take only cursory glances at the information as they "know" that they are all false. It would be very easy in this scenario to miss the one, new, true vulnerability that has appeared in the list.

But also a word of caution – we cannot simply remove false positives on a whim. Each false positive must be confirmed as such by someone with experience in checking for vulnerabilities. Hence, our users must email us directly to get vulnerabilities marked as false. We ask that they submit:

- The system(s) affected
- The ID of the plug-in causing the false positive
- Tests they have conducted to ensure the report is false

* Nessus can be configured to actively check for a vulnerability by attempting to exploit it rather than just grab the banner, but, this checking may cause a Denial of Service condition on the machine being tested. Although a more reliable method of detecting actual vulnerabilities, this is a trade off we had to make. Crashing our users' machines in the name of better security would not be popular!

We will then manually check their machine to ensure that we never mark vulnerabilities as false incorrectly.

Database Model

As mentioned above, the Nessus scan results are imported via an SQL script into an MS-SQL database. Information from the computer registration database is imported via DTS.

When a user brings up the web interface, SQL stored procedures are used to return the information they have requested. The database checks permissions for the requested IP address by converting the IP into a numeric form, e.g. given the IP address 192.168.10.12, we convert as;

$$\left(192\times256^{3}\right)+\left(168\times256^{2}\right)+\left(10\times256^{1}\right)+\left(12\times256^{0}\right)=3232238092$$

Unlike MySQL, MS-SQL has no built in IP address capability so this formula is written into a User Defined Function for efficiency. SQL is much better at dealing with numeric values – so having converted the IP address into a number we can check whether the user has permission for the requested IP address using a simple `SELECT` statement (using the keyword `BETWEEN`).

Looking carefully at the screenshots in Appendix A, we can see that this "encoded" IP address is passed as a URL parameter. So can anyone manipulate this number to view anyone else's data? No, because the permissions are checked by the database – and not just blindly followed by the ASP. So, in order, the user clicks to view a system's data and:

- IP Address is encoded by the ASP and passed to the database
- Database looks up the connected user, and checks whether this IP lies within a range permitted to that user (this is the authorisation step; authentication is taken care of by via trusted connections to the database).
- If so, the database checks the table of known positives for that system.
- The database prepares a record set of all discovered vulnerabilities, minus those listed in the false positives table.

So why use a false positive lookup table, rather than just turning off the relevant plug-ins in Nessus? After all, if we know a certain plug-in generates false positives, then why bother enabling it?

The table is there to allow us more flexibility in controlling false positives. If a certain system is not vulnerable to a certain version specific SSH vulnerability (though vendor patching), does this mean that this version is safe across the rest of our organisation? Definitely not. The false positives table ties plug-in ID numbers to IP, or range of IP, addresses. So, if we know a cluster of machines is running a vendor patched copy of an FTP server, we can mark those vulnerabilities as false for those specific systems.

Supporting Initiatives

This project relies on several other initiatives within our group. We had already established a precedent for scanning using our security policy (as mentioned above). In addition, we have deployed Microsoft Software Update Services (SUS) [14] to provide approved patches to our users (should they request it). SUS can be controlled via Active Directory Group Policy, or using a registry patch. We used another web page to generate the registry patch based on the users' choices.

When answering users' question regarding security issues, we also needed a document repository containing patching advice, information on vulnerabilities, etc. This repository is again made available via the web, so that we can quickly point users to answer documents and relevant sections of the security policy.

Most queries are currently handled via email, but we are working with different content management systems to hopefully provide a more intuitive web page that users can locate their own information on.

Improved Security?

The critical question after investing the time and effort in developing any security solution is "have I demonstrably improved the security of my organisation?"

Simply conducting a vulnerability scan will not fix anything. The scan itself is only a means to an end – the massive amount of data it produces needs proper analysis to improve the security posture of the organisation.

Although the vulnerability scanning in our organisation is a relatively young project it has already had some tangible benefits to security:

Improved level of patching over the departments who have opted into the project
During its initial rollout, we chose departments who wished to take part in a trial run. Since then, we have noticed a significant drop in the number of systems showing as "red" in the traffic light reports. Using web site logs and our own custom database reports, we can use the system to demonstrate that security of that particular department has improved. These types of quantifiable results are invaluable in management meetings.

Improved awareness of the Security Group and its role
The appearance of available vulnerability data made a lot of our more computer savvy users write asking questions about the function of our security group. Many have now started requesting advice and help with their security issues. Vulnerability scanning has effectively been used to demonstrate our presence within the organisation, which can only be a good thing.

Improved awareness and development of Security policy
We have used our security policy as an effective tool in shutting down unauthorised web, ftp and mail servers. Using it in this way has led some users to ask questions regarding other activities they are performing on their machines. As a result of these questions and feedback, we have been able to produce policy guidance notes the policy is more accessible to the end user.

Asset Management

The scan also performs a basic audit of hardware connected to our network. We have met Paul Simmons' challenge [3] by discovering those hosts which are not correctly registered. We also know what people are using as various FTP servers, web servers, etc. based on the banners supplied. This information is invaluable when we get news of a new vulnerability – we can now send targeted emails to our users. In a similar way to the false positive handling, this prevents "information overload" on our users. As opposed to "yet another email" from the security group, the system administrators can now receive only the relevant information. We can also catch people who have neglected to sign up for our internal issue-specific security mailing lists.

Documentation and Mainte nance

The system we have implemented will need to grow and change with time. It is essential that all the code is fully commented and that the system is documented, otherwise the system will only last for the period of my employment.

Other members of staff may also wish to contribute code ideas or suggestions for improvement. By employing a clear separation between data collection (Nessus), data storage (MS-SQL) and presentation (ASP) it is simple for a specialist in any of these fields to add their knowledge to the system.

This separation also allows us to change any of the technologies as required. If development were to stop on Nessus, we could swap in any other scanning engine capable of generating SQL output. We can also use other engines to perform sanity checking against Nessus; because as noted in the Network Computing article [6], none of the scanners tested detected 100% of vulnerabilities.

Storing the information in a database has another benefit. We are not tied to just using a vulnerability scanner such as Nessus – we can add in custom scripts which generate data, e.g. checking for open shares. We can then provide this extra information to the user without having to change the interface significantly. The other benefit of Nessus (and other scanners) is the ability to write your own plug-ins so we can test for organisation specific issues.

Conclusion

This project has demonstrated to use that deployment of a vulnerability scanner, especially in a large organisation, is about much more that just installing and running the scanner. Collation and distribution of the very valuable data collected is essential if the scan is to server its purpose.

We aimed to deliver a cost effective solution that would give our users the information they needed to ensure the security of our organisation. In this, we have delivered.

The project in itself does not mitigate the risk of an attack, but it does quantify the risk. But by providing the gathered information quickly and easily to the people on the ground, our generic "risk of successful attack" is greatly reduced.

Appendix A - Screen shots

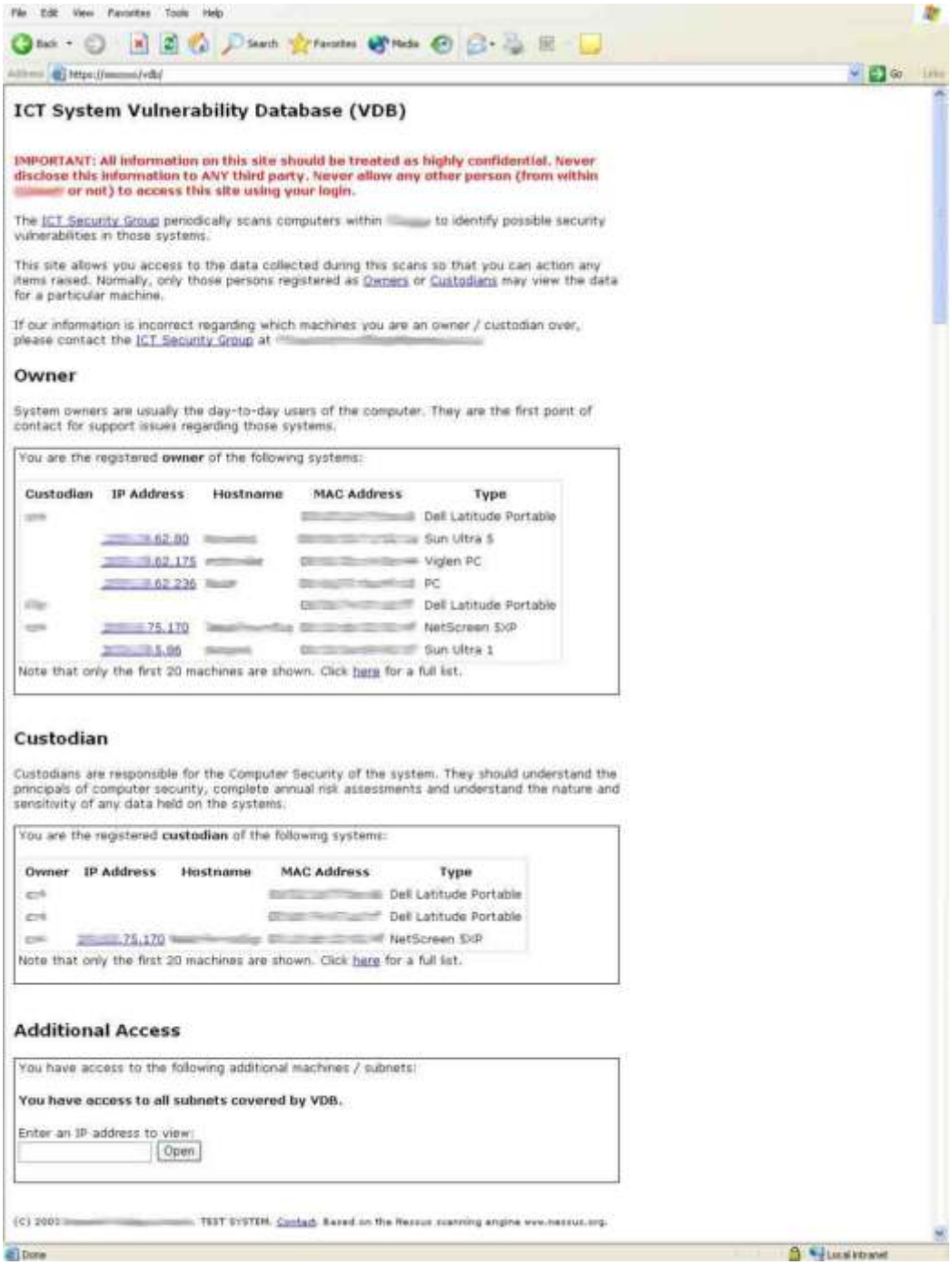

Figure 1 - Front page of the scanning results web interface

Figure 2 - Sample of a machine report

References

1. ScannerX,LLC. “Online Vulnerability Scanner” URL: http://www.vulnerabilities.org
2. UK Department of Trade and Industry. “e-Science Programme” URL: http://www.escience-grid.org.uk
3. Ward, Mark. “Closing the holes on hackers” BBC News Online, URL: http://news.bbc.co.uk/1/hi/technology/2989033.stm
4. CERT CC. “Keep operating systems and applications software up to date”, 30th April 2001. URL: http://www.cert.org/security-improvement/practices/p067.html
5. Austin, Brice, Dinehard et al. “Basic Security Policy” 1.2 SANS Security Essentials II: Network Security Overview. SANS 2003. pp2-20.
6. Shipley, Greg; Foristal, Jeff. “Vulnerability Assessment Scanners”. Network Computing. URL: http://www.nwc.com/1201/1201f1b1.html
7. Anon. “Nessus Documentation”. URL: http://www.nessus.org/documentation.html
8. victor@securityproject.org. “NessusWX”. URL: http://nessuswx.nessus.org/
9. Various. “Centre for Internet Security”. URL: http://www.cisecurity.org
10. ISS. “BlackICE PC Protection”. URL: http://blackice.iss.net/product_pc_protection.php
11. Microsoft Corp. “HOWTO: Use ADO to Connect to a SQL Server That Is Behind a Firewall”. URL: http://support.microsoft.com/default.aspx?scid=kb;en-us;269882
12. Microsoft Corp. “HOWTO: Set the SQL Server Network Library in an ADO Connection String”. URL: http://support.microsoft.com/default.aspx?scid=kb;EN-US;238949
13. Sarmanian, David. “Deploying PortSentry – A Simple and Free Barrier From Inside Hackers”. 5th January 2001. URL: http://www.giac.org/practical/gsec/David_Sarmanian_GSEC.pdf
14. Microsoft Corp. “Software Update Services”. URL: http://www.microsoft.com/windows2000/windowsupdate/sus/default.asp