Towards a Taxonomy of Network Security Assessment Techniques

Adam Shostack, Scott Blake
BindView Development
{adam|blake}@netect.com

July 2, 1999

Abstract

Vulnerability assessment tools are coming into widespread use, but the methods that they use are not well understood. We present a taxonomy of methods for testing if a target is vulnerable to a particular attack.

1 Introduction

Network security testing tools, such as SATAN, have existed for several years, and are coming into wider use as an expected component of a penetration test or security audit. However, the capabilities and limitations of these tools are poorly understood outside of the tiny, separated groups working on the tools.

As such, the field of testing for network vulnerabilities is one that has not received much attention from the scientific community. A recent mailing list discussion [Bugtraq] shows that even in a highly skilled group of security practitioners, the workings of these tools are not widely understood.

This paper attempts to bring order to the methods for algorithmicly determining vulnerability to known problems. We use vulnerability in a loose sense which includes not only software design errors and implementation flaws, but also misconfigurations and questionable user decisions (such as using a weak password). We provide an understanding of the issues and difficulties that arise in a devising and deploying a test or set of tests for a given vulnerability, and also offer up a set of terms that can be used to unambiguously describe a given test, allowing practitioners to more easily debate and discuss their merits. We limit our investigations to that set of problems which can be identified without credentials.

Note that we do not offer a taxonomy of attacks or vulnerabilities here; we are concerned with means of finding vulnerabilities, not the vulnerabilities themselves. We do use a few specific vulnerabilities to illustrate various limitations of methods of testing. We include an appendix giving an overview of a few vulnerabilities used in examples in the text.

In section N, we offer some terminology. In section N+1, we discuss means of finding vulnerabilities by testing for them. In section (N+2), we discuss the evidence for/evidence against connundrum. In section (N + 3), we discuss means of finding vulnerabilities that do not involve actual testing for the problem. The paper closes with some comments on credentialed vs. non-credentialed assessment methods.

We will present the broad areas of assessment by exploitation and by inference. Assessment by exploitation consists of fully or partially exploiting the problem, and drawing conclusions from observations made. We show that there are two ways to make observations about an exploit, and how they are useful in different situations. We also discuss issues of attempting to identify software before or during an exploit, the need to completely exploit a problem, and denial of service issues in exploit testing. In the inference analysis section, we will explain the use of banners and other identifying characteristics that allow assessment to be performed without exploiting the problem, as well as the use of timing and statistics.

2 Terminology

The field of testing for vulnerabilities has recieved little formal attention, and lacks a precise set of terminology. We offer some definitions of terms as we use them at Bindview.

Vulnerability A design flaw, defect or misconfiguration which can be exploited by an attacker.

Problem A synonym for vulnerability, less loaded with pre-conceptions about its meaning.

Test An algorithm for determining if a vulnerability is present in the tested system by taking advantage of the vulnerability.

Inference An algorithm for determining if a vulnerability is present in the examined system without taking advantage of the vulnerability.

Check A check is a test or an inference that finds a vulnerability. Checks, without a modifier, are the union of the set of tests and the set of inferences.

Banner check A banner check is a check which relies on a banner produced by a server daemon in its algorithm.

Credentials A username or password needed to access a service is considered a credential. This can include UNIX login access or the ability to call NT APIs over the network.

3 Testing Methods

The most obvious and apparent way to go about finding if a system has a particular problem is to attempt to exploit it. However, this is only one possible method for finding problems, and there are often difficulties with it, ranging from the visibility or parsability of the result to the effect of actually running an exploit against the target system.

3.1 Testing by Exploit

Testing by exploit involves using a script or program designed to take advantage of the vulnerability. This test code is often similar to exploit code that demonstrates the presense of a vulnerability, however, it will usually simply return a "RISK" result to its caller, rather than a root shell. There are exploits that do not return anything, but instead leave the targeted system in a vulnerable state, by, e.g., putting a plus in the rhosts file. The user must take a seperate action to determine if a vulnerability was exploited. This differentiator leads to two noticably different sets of tests; those whose result can be directly observed have different reliability characteristics than those which must be indirectly observed.

3.2 Possible distinctions

We considered distinguishing between *directly observed exploitation*, and *indirectly observed exploitation*. The directly observation set find their risk conditions within a the network connection or connections needed to exploit the problem, while indirectly observed exploits use additional connections or listeners for their data. This distinction turns out to be very hard to use in practice, as there is no simple defining line that we have found to make sense; there is, rather a continuum with ICMP and UDP checks being hard to define as direct or indirect.

3.2.1 Parsing Results

There is a set of cases in exploit tests where the result of the exploit is available, but hard to automatically parse; this happens with many CGI problems, since the web server is partially in the loop, and different web servers interpret output differently, for example, possibly requiring a "Content-Type:" line before sending any data to the client. There are also cases where evidence can be found directly or indirectly; an excellent example is the Sun telnet DOS attack, which can be indirectly observed through a new connection, and can also be directly observed by timing analysis of the ACK packets sent. Which is preferable for a deployed test typically depends on implementation related issues.

If the evidence of an indirectly observed exploit is unreliable, it may benefit from being run several times against a target, so we can use statistical techniques to increase our confidence that our observations are accurate. There is also the issue of when an attack takes effect; some tests for the majordomo bug referenced will go through the mail queue, which may be quite backed up. Thus, the commands may not execute for several hours or days while a mail queue empties, leading to a test that takes an extended period to complete. If our host happens to receive mail, it may be possible to test the time it takes for a message to cycle through the queue, but this time may vary substantially.

Lastly, there are exploits for which evidence can only be found through timing analysis; many denial of service attacks against NT services which absorb CPU time require .

can be tested for reasonably reliably through the use of timing analysis.

3.3 The Robustness Principle

The number of poor or non-compliant implementations of various services makes testing a tricky business.

Many systems will respond in unusual ways to standard requests, failing to be conservative in what they send. (Other, robust implementations lead to these problems not being noticed in normal use.) For example, the CERN HTTPD 3.0 will return a "500 Internal Error" message to mean access denied. [1]

Some tests will unexpectedly crash systems; HP printers are fairly notorious for being unable to survive a portscan. These problems are in fact the opposite of the previous paragraph: These systems are not liberal in what they accept. may well relate to the failure to comply with the robustness principle problems found in [Belovin].

3.4 Banners in Exploit Tests

For those tests which actually perform an exploit, there is often a temptation to improve its accuracy also invoking inference methods. This may improve a check's accuracy, but it reduces its ability to find related vulnerabilities. Take for example the asp-dot problem. This problem was first found in an early version of Microsoft's Internet Information Server. It was later re-invented in the Windows version of the Apache web server. A test that attempts to improve its accuracy by checking that it has received a 'Server: IIS' header may do so at the expense of reliability, since its test will not discover that Apache gives up files in the same way. This difficulty is based on a lack of clarity in defining what a vulnerability is. If the vulnerability being looked for is "the willingness of the server to give up files ..." versus "the willingness of the *IIS* web server to give up files, ..." then which test is appropriate becomes clearer. Unfortunately, its not clear which of those statements of vulnerability is more appropriate.

On the other hand, if a test only runs against targets against which it has been checked, then the probability of unintentionally denying service is much lower.

3.5 Denial Of Service Testing

For assessing vulnerability to denial of service attack in daemons which run on their own, vulnerability can be detected by crashing them, and observing that the port that they were bound to is no longer accepting connections. This has a number of potential difficulties, including but not limited to the possibility that an intrusion detection system has instrcuted an firewall between the tester and the target to block the connections. This class of problem actually affects many indirect observation techniques, but is most problematic when attempting to reconnect, since the defensive software may be blocking your re-connection attempt, which may be difficult to distinguish from the system not responding.

[1] While concealing the existence of private files is an admirable goal, the HTTP specifications define a number of responses specifically to indicate that access is not allowed.

Much preferable is careful examination of the network level behavior which terminates a connection. Is it done with a RST or a FIN? The RST generally indicates a program failure on the remote side. RST + push is often seen when a daemon dies, even one from inetd or with other monitoring service. It is more reliable to directly detect the abnormal behavior we cause than its secondary effects, such as an ICMP port unreachable message in a re-connection attempt.

Note that there are problems which are not traditionally classed as denial of service attacks, but whose tests have that effect on the service they exploit. An example of this can be seen by sending an HDS X terminal an empty UDP packet on port 161 [Shostack], which can crash the xterminal. For these cases, a check may choose to use a less reliable method in the interests of reducing the effect on the examined system.

Other tests, particularly those using a direct exploit technique, may crash a daemon or service as a by-product of the check (e.g., many buffer overflow attacks). Clearly, the impact of a test must be considered when selecting a testing technique, particularly when production systems may go down or become unresponsive. In practice, this is done on a case-by-case basis.

Some tests may also result in disabling the entire system rather than just a single service. These have the difficulty that the system under test may crash or become unresponsive for reasons unrelated to the test. Such reasons may include a router crash, a sudden and extended burst of network traffic causing loss of UDP or other unreliable traffic, a machine being rebooted, etc. This is a different matter when the crash of the machine is a confounding variable and an expected result of a test. When it is a confounding variable, it may be possible to build other tests into a check to isolate it. However, when the goal is to crash the target, only repeated testing of the attack can assure accuracy. However, there are substantial practical difficulties in doing this.

4 Coming to Conclusions

For a check to be useful, there must be some sort of reporting at its end. The check algorithm should assess the evidence, and provide results. (It would be possible to build a system where all the evidence is presented to the user for assessment, but it seems a rather mechanical task which can reasonably well be automated.)

For any check which we perform, we can either work from a hypothesis and attempt to disprove it, or we can look for a preponderance of evidence.

When disproving a hypothesis, there are essentially two types that can be used; that of *risk assumed* , and that of *safety assumed.* For either, we must seek to disprove the hypothesis. The disproving safety-assumed hypothesis is preferable when testing a directly observed exploit, because the exploit can easily be shown to be successful.

If the exploit is observed indirectly, we should assume risk, and seek ways to disprove it. One route to disproving is to send the commands that should exploit the hole, and to see an error message. Since we have been shown the error, we can not have succeeded (except in the face of a deceptive system, such as the DTK [Cohen]). For example, we can disprove the existence of the majordomo reply-to hole if we can demonstrate that majordomo is not installed. We can show that majordomo is not installed by showing that there is no mail server on the host. If there is a mail server, we may be able to show that it will not accept mail for majordomo. For the glimpse cgi, we can disprove the existence of the hole by proving that the cgi is not installed, or that there is no web server present. The assumed risk model may be contrary to user expectations, since if a check reports a vulnerability, the user seems to expect that the check has found evidence of that.[Bugtraq]

Some words about preponderance of evidence, and multiple-method tests tk here.

5 Inference Methods

Several of the early examples of network testing programs, including SATAN, made extensive use of *version checks*. Version checks are those checks which rely on a version identifier provided by a server, possibly in response to a request. The version is compared to a known safe or known vulnerable version number. Version checking is the simplest member of the inference checking family. These methods do not actually exploit problems, but look for evidence that they may be exploited. Methods for examination by inference include, but are not limited to, versioning, program behavior, OS stack fingerprinting and timing.

Behavioral analysis can shine when, for whatever reason, we want to disprove a risk hypothesis. In these cases, it is reliable to show that a program behaves in ways which would require substantial changes to the source of the vulnerable version. This allows us to infer that the software we are looking at is not the software in which the risk is present. This is not quite as reliable as exploit techniques, but it is less intrusive.

This technique can be useful when checking for a new problem whose details have not been made public; the benefits of keeping exploitation information private may outweigh the reliability issue. In addition, these techniques are useful for checking vulnerability to denial of service attacks, since they are less intrusive, and one can check a number of issues between reboots.

Lastly, these techniqu1es are very useful for quickly scanning a large number of targets, since the computational and network requirements for inference checking tend to be very low.

5.1 Versions

Versions are an extremely simple sort of identifying behavior. Many programs will offer a banner to identify themselves. Sendmail and wu-ftpd are good examples of this. Sendmail will, by default, reveal not only its version, but its configuration file information, as well as the time. Wu-ftpd reveals its version in gory detail. Other programs, such as many web servers, will reveal their version information in response to a simple anonymous query.

Version checks are a low impact way to find evidence of a vulnerability, but it will fail to function if the banner has been changed, causing a false negative. The ease of changing banner information varies from program to program. With sendmail, one must merely brave the configuration file, whereas with ssh, one must change the code.

The use of banners as a means to infer a problem will, like most inferential methods, fail to catch programs that, unknown to the check writer, fail in similar ways across programs.

5.2 Port Status

Knowing what ports are open on a system can be useful in and of itself. A large number of portscanning tools have been written (nmap, firewalk). We note that techniques for distinguishing between a firewalled and a closed port have been developed. In addition, it is feasable to discover when a port has access control on it.

There are certain vulnerabilities which can be reliably infered from the status of a port. Many of these vulnerabilities relate to policy violations. For example, if there is a policy that only approved servers can offer web service, then the presence of an open port 80 is a vulnerability. Alternately, if all systems are required to run SNMP network management software, then the SNMP port being closed is a problem. These checks are refered to as *port status* checks.

5.3 Protocol Compliance

If a port being open is not quite enough to determine a vulnerability exists, then it may be desirable to simply excersize the server, for example with a request for status, or a simple information request. This may be useful with finger, where a banner that announces "Finger is not available" may be acceptable, but actually serving up information is not.

5.4 Behavioral Analysis

There are *behavioral analysis* methods which are more complex than versioning or determining port status.

There are many commands in a protocol whose output is distinguishing. Optional or new commands that are outside the requirements for interoperability are a fertile ground for this. For example the response to the SMTP help command is distinctive for families of mail server code, although it usually will not allow a test to distinguish versions.

Behavioral analysis can also reveal information about the configuration of a system. Some of these ways, such as tcp_wrappers delaying the response to a connection for a DNS double reverse lookup, are quickly apparent. Others are more subtle, such as web servers revealing information about index files being on disk or generated on-the-fly by the headers that they produce.

5.5 OS Identification

An outgrowth of this is the observations from the authors of Queso and others [Fyodor] that host operating systems can be identified by their behavior in responding to out of spec or simply bizarre combinations of TCP packets. ((This section to be expanded.))

5.6 Timing Data

The use of timing data to detect vulnerabilities is relatively unexplored currently. However, there is a set of denial of service attacks that consume CPU which can be effectively tested for in this way. Timing analysis can also be used to determine if the su command in Solaris (before 2.5.1) has been given the correct password. It can also be used in many cases to discover that account lockout has been invoked, that certain files do not exist, that tcp wrappers are in use, etc etc.

I suspect it will be a fascinating area of research into understanding the behavior of remote programs, if not as reliable as a direct exploit. Note that it may be used to reveal RSA keys over the network, [Kocher] but that this is not a fast technique.

5.7 Reliability and Accuracy of Inference

Deciding when to use inference is a matter of strong and ongoing debate, with various authors and readers in strong disagreement. In practice, these disagreements are settled by research and testing into the particular vulnerability, and seeing how reliable the inference method seems to be on a variety of test platforms.

Assuming the designer of a test has done proper research to determine if an inference method is appropriate, these methods are still unlikely to discover that someone has used a non-standard patch to close a problem, potentially reducing the accuracy of a check. Further, as noted, the check will not find similar problems in other implementations; this may or may not be a reliability issue, depending largely on how the vulnerability being checked for is defined. As the definition becomes more precise and narrower in scope, the inference becomes more reliable, as other implementations are likely to be defined as out of scope.

6 Credentials and Tests

The area of testing for vulnerabilities by sending packets is a useful one, however, it excludes two large classes of vulnerability tests — those that can be done with credentials, and password cracking.

The vulnerabilities and methods discussed above generally do not require any more access than IP connectivity to the target machine. However, many attacks are launched by users with shell access (UNIX) or resource access (NT) and aim to promote the user to root or Administrator either permanently or for the purpose of running a command. Such vulnerabilities are often more difficult or impossible to check for from a remote, unprivileged location.

Furthermore, testing is greatly aided by having root or administrative access to the target system. This allows the testing processes to examine the system in the most unobtrusive and reliable way. Unfortunately, if implemented, this storing of credentials and passwords may make the testing platform itself a fat target since it holds the credentials for accessing the target machines. Even storing user credentials may draw attackers. Compromising the credential data would allow an attacker to masquerade as a different user, at least, possibly as a user known to be held by the testing system and possibly as a root or administrative user.

Clearly, the security ramifications of the testing process itself and storing the results of the process must be carefully considered by system designers and end-users alike. When the testing system holds the keys to other systems, the security of the testing system becomes very important. A well-designed testing system will include integrity tests of the system itself. Otherwise, testing for vulnerabilities may be more dangerous than leaving the systems open.

Despite the advantages to the vulnerability assessment and in addition to the risks of storing credentials, organizational factors may play a role in the difficulties of testing with credentials. Particularly in large organizations, the persons charged with assessing the security of a network are often not the same people who have control of the network. Obtaining the credentials for the persons controlling them may pose its own challenges.

We do not consider such things as anonymous (for FTP), null or guest (for SMB) or public to be proper credentials, but well known tokens.

6.1 NT

On NT systems, many native calls can be used either on the local or a remote system. Some of these calls require some user credentials and the results of the call may be highly dependent on the credentials used. It can be very useful for vulnerability analysis to compare the results of a call made with anonymous, user, and administrative privileges. Access to a resource by anonymous persons may not be acceptable, but user access may be a reasonable or necessary risk. Administrators can make more informed decisions about appropriate configurations when this information is available.

6.2 UNIX

Most vulnerabilities on UNIX systems that are better assessed with credentials than without fall into the category of promotion attacks. These are vulnerabilities that allow users to gain root access. On multi-user systems where user access typically includes the ability to run code, user promotion can be a very important class of attack. User access to NT across a network typically does not include the ability to run code.

6.3 Password Assessment

Obtaining encrypted/hashed passwords for use in password cracking operations is also an area where user or administrative access is highly desirable. To thoroughly analyze the strength of user passwords (which are extremely important to the overall security of the network), the assessor, whether human or software, must be able to obtain the encrypted/hashed passwords. This case also illustrates the extreme importance of the security of the assessment system itself. In addition to storing credentials used during the analysis itself, we now also need to store the results in a secure manner.

6.4 Reliability and Accuracy

Using credentials, particularly administrative or root credentials, greatly enhances both the reliability and accuracy of tests providing that access with the credentials can be obtained. For example, if we have the Administrator username and password to an NT system, but no access to a service that accepts these credentials, there is not advantage to having these credentials. However, if we additionally have code present (or sufficient access to execute code) on the machine which has the credentials, we gain nearly perfect reliability and accuracy. The disadvantage to this approach is that it may provide a distorted view of the vulnerabilities on a machine. For example, ports or services may only be open to the localhost. Thus, credentialed checks may be answering a different question than is typically asked by a network testing tool. In other words, using credentials tells us what vulnerabilities are available to attackers with credentials, which is likely different from the vulnerabilities present for attackers without credentials.

7 Future Directions

This is a large, and still open, area of research, that is being driven largely by commercial needs. There is much work to be done in the area of multi-host testing, sniffer augmented testing, credentialed testing, etc. There is room for additional rigor in defining when various testing methods are appropriate. There is much that can still be done in defining the role of behavioral analysis. There likely exist other taxonomies that can be constructed; some may offer more benefits than this one.

8 Conclusions

We have explained some of the myriad difficulties in creating tests to find vulnerabilities on IP connected networks. We have presented a taxonomy that allows classification and discussion of these techniques. Additionally, we have presented and justified a higher level classification for checking, that of credentialed vs. non-credentialed. We hope that these advances will stimulate and encourage public research into the area of automated assessment of vulnerabilities.

9 Acknowledgements

Jordan Ritter, Todd Sabin and Izar Tarandach provided useful and stimulating discussion and feedback on drafts of this paper.

References

[Belovin] S. Bellovin, "Packets Found on an Internet," *Computer Communications Review*, Vol 23, No 3, July 1993, pp. 26–31.

[Bugtraq] Bugtraq Mailing list discussion, Feb 7, 1999-Feb 18, 1999.

[B94] J. P. Rouillard, Bugtraq posting, 8 Jun 1994, archived at http://geek-girl.com/bugtraq/1994_2/0360.html.

[Cohen] F. Cohen, "The Deception Toolkit," posted at http://www.all.net.

[Cowan] C. Cowan, C. Pu, et al, "StackGuard: Automatic Adaptive Detection and Prevention of Buffer-Overflow Attacks", Proceedings of the 7th USENIX Security Conference.

[Farmer] D. Farmer, "Shall We Dust Moscow," 18 December, 1996, available at http://www.fish.com/survey/.

[Fyodor] "Fyodor," "Remote OS detection via TCP/IP Stack FingerPrinting," *Phrack*, Vol 54, No 9, 25 December, 1998.

[Kocher] P. Kocher, "Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems," *Adavances In Cryptology – CRYPTO '96*, pp 104-113.

[PHF] CERT Advisory CA-96.06, March 20, 1996

[SATAN] D. Farmer, W. Venema, "SATAN 1.1.1 Documentation"

[Shostack] A. Shostack, Bugtraq mailing list post, Jan 12, 1999, *http://geek-girl.com/bugtraq/1999_1/0160.html*.

Enumeration of Vulnerabilities The vulnerabilities listed here are selected because each is useful to illustrate some point of distinction when compared against the others. Some of these are interesting because they are difficult to test for, and we are not aware of a reliable method for finding them.

phf CGI The *phf* CGI was a C language example CGI included with pre-version 1.2 NCSA web servers [PHF] It took no care to ensure that its arguments were correct before passing them to the system() call. This allowed an attacker to send an HTTP get request which included a ;command which would be executed with the UID of the web server.

Glimpse CGI The *aglimpse* CGI search tool offered a remote command execution method similar to that in phf, with the exception that due to the complexity of the Perl code surrounding the backticks in the code, the result needed to be sent out via some mechanism other than display to the web browser.

Majordomo Reply-To The *Majordomo* mail processing package failed to check its input well, and up to version 1.90 was willing to execute commands enclosed in backticks in the reply-to address if commands in a mail message were somewhat invalid.[B94]

Sun telnet DOS The *Sun Telnet DOS* consists of sending a stream of D̂ characters to the telnet port of a Solaris machine. Done with the right telnet option negotiation, this causes the target to slow substantially.

Smurf A *smurf* attack, named after the first published exploit, uses a forged ICMP packet sent to the broadcast address of a network to cause a storm of packets to be sent to the source address in the packet. This allows a very small pipeline to the internet to produce a large amplification, resulting in many more packets arriving at the target than the attacker sends.

ASP dot This is an attack against some Microsoft web servers that allows an attacker to see the contents of a program file. It works by appending a '.' to the filename of an executable. When the web server parses the request, it sees that it does not end in ".asp" and does not execute the asp, but instead sends the asp file to the requester.

IP Spoofing We include *IP Spoofing* as a reminder that there are aspects of insecurity to which you are inherently vulnerable, and short of protocol revision, there is not a lot you can do about them.

Mail Flood *Mail flooding*, unlike IP Spoofing, can be defended against. However, due to the impact of testing, and the fact that a test is not binary, we choose not to test for mail flooding.