Software Engineering Institute | Carnegie Mellon University

Adapting Penetration Testing for Software Development Purposes

Kevin van Wyk

January 2007

ABSTRACT: This article provides background information on penetration testing processes and practices. It then discusses the issues related to integrating penetration testing into a software development life cycle by describing the pitfalls associated with traditional penetration testing practices as well as making recommendations for improving these practices. A related article describes types and examples of penetration testing tools.

Today's software penetration testing tools, practices, and (to some degree) staff have been developed and improved for an IT Security user base, primarily. However, to effectively make use of these elements in a software development environment takes careful thought and clear goals.

For example, most existing penetration testing tools and services offer a fairly rigid technology-centric perspective of their respective findings. This is in stark contrast with the software security touchpoints recommended here on the BSI portal, where a more business risk approach is stressed. The business and architectural risk analysis process should serve as a prioritization input to penetration (and other security) testing processes. However, that is not generally what happens in today's environment [Arkin 2005, Janardhanudu 2005, Michael 2005].

To get around this, and to get closer to the practices discussed here on BSI, this document provides a description of and recommendations for a penetration testing process and methodology that is more suited to the needs of software developers than is typically found today.

Additionally, the document provides both a conceptual as well as a more specific survey of the tools available today for conducting penetration testing. This tool survey is then balanced against the need for trained, skilled, and highly motivated testing staff. Staff training is addressed and compared against mentoring or apprenticeship types of on the job training processes.

Software Engineering Institute
Carnegie Mellon University
4500 Fifth Avenue
Pittsburgh, PA 15213-2612

Phone: 412-268-5800
Toll-free: 1-888-201-4479

www.sei.cmu.edu

INTRODUCTION

At a most basic level, the term "penetration testing" refers to testing the security of a computer system and/or software application by attempting to compromise

its security, and in particular the security of the underlying operating system and network component configurations. However, that broad definition encompasses many things and indeed means many different things to different people.

For the purpose of this document, we'll explore the practice of penetration testing from both an historical purpose as well as for how it can be specifically applied to the field of software development.

Background of Penetration Testing Processes

Penetration testing of computer systems and software applications has been around for decades. No doubt, a basic penetration test was among the first activities performed when the very first concerns of a computer's security were raised many years ago.

In the early 1990s, the practice of penetration testing began receiving widespread attention among the Internet community with the publication of a Georgia Institute of Technology student's research software project, the Internet Security Scanner, as well as an early paper on the subject [Farmer 1993].

The basic process used in penetration testing is a simple one: attempt to compromise the security of the mechanism undergoing the test. Computer operating systems, along with their access control mechanisms, are natural test candidates for penetration testing. In a networked computer context, operating system configurations, including their respective network services, are common test targets for penetration testers and attackers alike. These operating system components have offered countless opportunities for penetration testing over the years because they are so central to the security of the computers that they support.

However, as software has increased in complexity and attack methodologies have advanced in capability, there has been a corresponding push to turn penetration testing into a more general-purpose security testing process. This push has focused on applying penetration testing techniques "up" the software abstraction levels, beyond the operating system and network services, towards the application software itself. Naturally, this has brought with it a bevy of technical challenges and difficulties.

A major driving force behind the advances made in penetration testing technologies has been mainstream IT Security practitioners. They have quickly embraced penetration testing as a means for assessing the security of a deployed (or soon-to-be deployed) computer. Indeed, the software tools available to penetration testers have largely been written with the IT Security professional as its principal user.

The earliest penetration testing experiences involved arming an IT Security engineer with a handful of attack tools and pointing the engineer at the system to be tested. Not surprisingly, there was little in the way of repeatability and reliability in these tests. As the processes and the tools have matured, however, repeatable testing processes have been developed and adopted significantly.

Another area where much progress has been made has been in the tools' reporting capabilities. The earliest tools did nothing more than recite each discovered security defect. Newer tools, on the other hand, have added extensive hyperlinked information databases to their reporting in order to make things easier for the engineers that receive the reports. Additionally, some tool vendors have added enterprise-level views into their respective reporting capabilities, giving the chief information officer (CIO) or IT Security manager the ability to quickly see vulnerability issues across the corporation, improvements over time, and so on.

The de facto stewardship of penetration testing tools and processes by IT Security organizations is significant. Penetration testing has afforded IT Security people with the opportunity to test a computer and application's configurations from afar, as a sort of independent audit function. Tests conducted this way tend to be late life-cycle tests, shortly before an application is deployed, and tend to be outside --> in oriented.

Adopting this outside --> in methodology has served penetration testing well, at least in its early stages of development. Most such tests have been designed to "emulate a hacker" who would otherwise have zero or very little knowledge of the underlying software being tested. Since most of the traditional penetration testing tools available essentially automate a typical attacker's actions, this approach has been easy to justify without causing too much consternation by the software developers. "After all, since an attacker would use these same sorts of techniques, it is best to do the testing ourselves before someone else compromises our new system," or so the typical logic goes.

On the other hand, the outside --> in approach is most likely to blame for a significant shortcoming in test results: remediation. Because the penetration tester has little insight into the application's design itself, it is not likely to be able to produce recommendations for remediation beyond a superficial level, such as "turn off this network service" or "turn on a firewall rule to prevent external access to the network service." Admittedly, this practice of generating relatively superficial recommendations has served the CIO audience well, but will cause difficulties for a software development audience.

Today, penetration testing has indeed advanced significantly but is still not as useful in the software development process as it ought to be, for reasons that

we'll describe in greater detail below [Herzog 2005]. Indeed, most of the technical advances in penetration testing have resulted from better tools, integration of an ever-expanding set of known software security defects, and in reporting of findings. While these are important advances, they are largely independent of enabling penetration testing to be highly valued in a software development context.

Program Ownership

We noted above that penetration testing was largely fostered and developed among IT Security organizations. It is significant to note that this has resulted in penetration testing coming under the purview of many organizations' CIO system support organizations, as the CIO in most companies is responsible for the company's IT infrastructure as a whole.

Software development, on the other hand, generally is not the direct responsibility of the CIO. Instead, funding for internal software development projects in most companies is the responsibility of their respective business organizations, which ultimately are subordinate to either the chief operating officer (COO) and/or the chief executive officer (CEO). Even where the CIO is responsible for software development, the majority of his/her security staffs have been drawn from the systems support and operations staffs, most of whom have little experience with application software development. Therefore, the actual software development is often outsourced or staffed via temporary or assigned employees for the duration of the project itself.

As we've noted, the CIO's ownership—specifically, the IT security or system support staff ownership—of penetration testing has resulted in tools that are tailored to their purposes. That is to say that there has been very little integration into the software development process in most organizations. This situation has also promulgated the outside --> in methodologies that we noted above. That is, the CIO's penetration testing processes are largely used as security acceptance (or non-acceptance) tests prior to an application being approved for production purposes.

Another significant fact that comes as no surprise is that, because of its placement within the CIO organization, traditional penetration testing has also resulted in test reports that are tailored to the CIO audience. That is, the reports generally prescribe remediation at the firewall, network, and operating system configuration level. Although these are important to the overall security of an organization—including its software applications—the focus would need significant adjustment to be truly useful for software development purposes.

The Push for Increased Automation

The earliest penetration testing processes were highly ad hoc and manually intensive. (Some would argue that they weren't "processes" at all, at least not in any substantive sense.) Over time, as these processes matured, it became easier to automate the testing.

There has been a corresponding push for increasing the level of automation in the testing processes themselves. This could be due to CIOs' desire to get greater repeatability and predictability in the results, but it could also simply be to drive down costs. Whatever the reason, the automation push has affected not only the actual testing of targeted networks and computers but the reporting process in just about every tool set available, commercial as well as open source.

Although this has arguably served the CIO community well, it has made penetration testing at least somewhat less effective for general purpose software testing. This is due to the fact that rigorous test scenarios need to be unique and tailored for a particular software product. A "one size fits all" approach can never address all of the idiosyncrasies of every piece of software that it tests.

On the other hand, a highly automated testing tool can be very effective at testing a large base of homogenous networks and operating systems, which is of real value to a CIO audience.

Minimization of Cost

We said that a major contributing factor in the push for increased automation in penetration testing was cost reduction. This bears further consideration with respect to applicability to software development.

In order to minimize the cost of penetration testing, one essential thing that must be done is to reduce the amount of labor time associated with each test. That is, testing tools need to be more "fire and forget" in their nature so that the security engineers spend as little time as possible in running the software and analyzing the results.

This cost minimization, while necessary to the CIO audience, has also resulted in eroding the effectiveness of penetration testing for a software development audience. The reason for this, as we'll explore in more detail in this document, is that, in order for penetration testing to be truly useful to software developers, testing scenarios and test cases need to involve a great deal of in-depth knowledge of the application undergoing the testing. This means that more, not less, human time and energy needs to go into testing.

Of course, the same can be said for general software testing, so it is not unique to security testing. And driving security concerns further "back" in the development cycle—to include requirements and specifications—can go a long way to assisting testers in developing realistic and effective test scenarios.

CURRENT PRACTICES: STRENGTHS AND LIMITATIONS

As we've described in the introduction above, most of today's penetration testing tools and processes have been designed with the CIO's needs as the primary focus. In this section, we'll explore the types of tools and their uses that are most commonly used in practice today. Later, we'll dive deeper into the specific tools themselves.

We'll start by looking at the categories of tools available to the tester, followed by a discussion of what is meant by "penetration testing" as opposed to "vulnerability scanning." We'll then discuss many of the pitfalls associated with traditional penetration testing and then make recommendations for how one can find value in penetration testing, despite these potential pitfalls.

Categories of Penetration Testing Tools

Conventional penetration testing tools come in a variety of forms, depending on what sort of testing they can perform. A principal distinguishing factor is the perspective from which each type of tool operates, that is, whether a testing tool evaluates its target from afar or from relatively close up—at least within the same computer system. This section provides a description of each of the popular classes of tools used in penetration testing today [Fyodor 2006].

Host-Based Tools

Some of the first security assessment tools available were host-based vulnerability scanning programs. Perhaps the most well known of these was Dan Farmer's COPS program, which evaluated the security posture of a UNIX computer. Although the terminology "penetration testing" wasn't commonly used when COPS was released, it (and others like it) quickly became a mainstay of every penetration tester's toolkit.

Host-based testing tools typically run a series of tests on the local operating system to assess its technical strengths and weaknesses. For example, they evaluate file access control mechanisms for opportunities for attackers to affect the security of the host. Using automated processes that computers excel at, they are able to exhaustively examine every file, configuration data (including registry keys on Windows systems), installed patch inventory, and so on from the perspective

of every user ID, group ID, etc. configured on the system. (This, admittedly, is an area in which automation not only excels, but is absolutely called for.)

They are also able to look for other common operating system configuration mistakes and omissions such as dangerous setuid files, unnecessary network services enabled, and excessive privileges for user accounts.

To do these tests, host-based tools require a database of known problems to look for. In some cases, they can look for particular states, such as the ability of a process owned by a particular user ID to write to a file somewhere on the file system. However, even those states need to come from a predefined set of things that are known to be bad from a security perspective. This means that host-based security assessment tools operate on what is known as a black list method.

Host-based tools have a distinct advantage over the network-based tools covered in the next section. Their advantage is that they can directly interrogate each of the elements they're looking for without having to make informed guesses from afar. That said, a significant disadvantage that they have is one of scale: evaluating a large number of computer systems can require a great deal of manual effort in installing the tool on each target system. This is the case even for "enterprise" versions of host-based tools that enable a single system administrator to run the tool across several machines within an enterprise, as they normally require some form of software agent be installed in advance on each computer to be evaluated.

While these tests are useful and necessary, they do little if anything to address the security of one's application software, as their focus is entirely on the installation and deployment characteristics of the operating system they're running on. Therefore, they are of diminished value to someone seeking to assess the security of an arbitrary piece of application software.

Of course, it is still necessary to note that the security of application software can be adversely affected—perhaps catastrophically so—by the security of the underlying operating system and environment. Further, the security of an application can be greatly enhanced by careful and application-specific attention to matters of operating system configuration, such as file access control. To that end, this type of host-based tool is still useful for assessing the indirect aspects of an application's security.

Network-Based Tools

Not long after COPS started gaining popularity as a host-based security configuration assessment tool, system administrators (and, no doubt, attackers) began to want some of the same features that the CIOs were demanding—particularly increased automation. When one is faced with evaluating several hundred com-

puters, host-based tools begin to lose their charm very quickly. At this point, network-based security evaluation tools were born, starting chiefly with Chris Klaus's Internet Security Scanner (ISS) program (in its original freeware form).

Network-based testing tools attempt to assess the security configuration of a computer operating system from afar—across a network. They examine a target computer (or, more often, computers) for weaknesses that may be exploitable from a remote networked location. As with their host-based counterparts, they do this using a database of things that are known to be bad.

Network tools are generally able to detect a variety of security problems on the machines that they examine. These include (possibly) unnecessary network services being enabled, weak network services being enabled, the patch state of the network service software, and so on.

The principal advantage to network-based testing is scale. A huge number of computers can be evaluated across a network in a fraction of the time it would take to evaluate those same computers using host-based technologies alone. However, that advantage is also the cause of its principal disadvantage: coverage. Network-based testing is only capable of directly evaluating the externally accessible interfaces of the computers under evaluation. It is therefore incapable of assessing many of the criteria that host-based tools have done since their inception, such as file access controls and setuid files.

In the past few years, quite a few tools have come along that have combined host and network-based attributes. By doing this, they are able to leverage the strengths of each of these forms of penetration testing tool. Some require network-connected software agents to be connected, but others attempt to "break in" to targeted systems and run test tools on them from remote locations. And still others use administrative privileges to remotely log into the systems being evaluated and perform a similar assessment to host-based tools. This practice is not without risk and should be considered carefully before being permitted on production systems, but the benefits of being able to look more deeply into the systems being tested could well be worth the risks for many organizations.

Application Testing Proxies

Just as early penetration testing was largely a manual process, a popular type of manual application security testing tools has emerged. These tools, called application testing proxies, enable the security tester to look behind the graphical user interface when testing a web application or web service.

They accomplish this by interposing themselves between the application client (e.g., a web browser) and server (e.g., a web server or application server). Re-

quests to and responses from the server can then be intercepted, observed, and optionally manipulated.

These tools have rapidly become a mainstay of application penetration testers everywhere, primarily because they afford the maximum level of visibility and control over a web application. Unfortunately, as we noted, they are hugely manually intensive and not well suited to testing every aspect of every application.

Application Scanning Tools

The newest entry in the penetration testing tools category is so-called application scanning tools. These tools purport to be able to do penetration testing scans of general purpose (mostly web-based) software applications. To that end, they have the potential to take the network-based penetration testing tools to a new level.

Application scanning tools generally work in either (or both) of two ways. First, they connect to web applications and attempt a series of well-defined tests for each data field, cookie, and so on. Secondly, most application scanning tools are able to initiate a "learning mode" in which they observe the normal operation of a web application. From that collection of normative data, they attempt to exploit common web application defects such as data overruns, SQL injection, and cross-site scripting (XSS).

While all of this sounds okay in theory, there are several problems with it in practice. To start with, application scanners only perform a relatively small number of probes against the different data entry possibilities for each application tested. Secondly, they are only able to detect success or failure, when human judgment (if it could be applied here) might well lead the tester to try different sorts of tests, dig a bit deeper, and so on.

In essence, a web application that fails any of the tests put to it by an application scanning tool is truly one in bad shape. On the other hand, a web application that passes all of the tests still can't be declared "secure" in any significant sense.

Convergence with Other IT Security Technologies

As the various penetration testing tools and processes matured, it quickly became obvious that there would be benefit (to the CIOs' organizations) to tightly integrating the testing tools with other IT security technologies. These included firewalls as well as intrusion detection and prevention systems.

The theory, at least according to the many product vendors' marketing literature, was that penetration test results can be used to "proactively" make adjustments

to an organization's perimeter protection and attack detection capabilities. For example, a newly discovered security defect can be probed for using penetration testing tools and, when found, the results can be coordinated with the firewall and IDS teams to protect weak machines from being broken into. Similarly, if or when attempts to exploit known weaknesses on systems are spotted by an IDS, the priority of the reporting should be much higher than attempts to probe for weaknesses that have been verified to have been addressed.

With these additional security technologies thus included in the capabilities of the penetration testing tools, the logical next step for the product vendors was to provide a unified (sometimes referred to as dashboard) view into the reporting provided by all of each organization's deployed security products. This was made extraordinarily difficult by highly heterogeneous environments, such as those frequently found in large corporations today. However, it did drive a push towards more standardized reporting data formats. Although all of these were notable advances for the penetration testing tools, these new feature sets do not provide much value to the software developers that may consider using the tools. The lesson here for software developers is to not look for these types of features in any penetration testing products that they consider; instead, pay close attention to the products' technical testing capabilities.

The Role of the CIO Audience on Penetration Testing

Both host-based and network-based testing tools and techniques have their benefits as well as shortcomings, as we've noted. The primary shortcoming, from the perspective of software development, is that both classes of tools essentially work from a set of scripts of how to probe and (possibly) exploit a set of known security weaknesses. They're basically "one size fits all" approaches to testing. This model is far better suited to testing general purpose operating systems and deployed networks than it is to testing application software. Given that penetration testing was primarily developed within CIOs' organizations over the years, this should come as no surprise.

From the point of view of a software development organization, these shortcomings are not insurmountable, but they do mean that consumers of penetration testing technologies must be well informed in order to find the value in them. We'll explore that in more detail below.

Penetration Testing Processes Today

In addition to the tools available to the penetration tester, it is important to consider the processes that are used and to assess their applicability to software development.

As we've seen, early penetration testing tools, as well as processes, were highly ad hoc, but the push to get more predictability and repeatability in the results led to improved toolsets being created. The actual processes used, however, followed the improvements in the tools and continue to be highly dependent on the needs of the organizations doing the testing. At the heart of this disparity is the fact that the term penetration testing means so many different things to different people.

One of the first things that should be addressed is the administrative side of doing the penetration test. In particular, it is absolutely essential to establish a clear and explicit set of rules of engagement that govern what and how the testers will do during their test activities. It is also highly advisable to have an explicit and fault tolerant "call off the dogs" process whereby either party can immediately halt all testing. Although these may seem mundane, experience has shown us that the effort is entirely justified.

With the administrative matters out of the way, the fundamental process that most organizations follow in doing network-based penetration testing, along with a brief description, looks something like the following:

1. Target acquisition
2. Inventory
3. Probe
4. Penetrate
5. Host-based assessment
6. Continue

In the target acquisition step, the tester looks for legitimate targets for testing. This can be a manual process, even in the form of an explicit target list provided by the organization undergoing the testing. It can also be an automated process in which the tester scans a range of network addresses in search of potential targets. In practice, however, it is most often a combination of manual and automatic in which the testee provides a starting list of network addresses and the tester uses software tools to look for additional computers in the network vicinity.

Once a list of computers to be tested has been developed—and explicitly agreed to—the tester uses a set of tools to conduct an inventory of available network services to be tested. At the same time, the tester gathers whatever version information can be determined from across a network. This may include data such as operating system type and version and network server types and versions.

The next step is to probe the available targets to determine whether they are susceptible to compromise. Note that many testers and test tools proceed directly to

the next step and immediately attempt to penetrate each of the targets. In many production data processing environments, however, it is advisable to proceed with additional caution.

Next, each identified vulnerability (or potential vulnerability) is exploited in an attempt to penetrate the target systems. The outcome of exploited vulnerabilities varies widely from one system to the next. Also, the level of invasiveness involved in exploiting a vulnerability can heavily impact this step. For example, if a vulnerability can result in the attacker—in this case, the tester—having the ability to overwrite an arbitrary file on the target system, great care should be taken in how to exploit the vulnerability. On many systems, it would become possible to give oneself an account on the target system, but this may involve overwriting vital system information—perhaps even the /etc/passwd file on a UNIX system![1]

Depending on the goals and time frame of the testing, a host-based assessment is typically conducted of any system that is successfully compromised. This enables the tester to identify vulnerabilities that provide additional vectors of attack, including those that provide the ability to escalate privileges once the system is initially penetrated.

Again, depending on the nature of the testing and the underlying rules of engagement, the next step would be to "continue," that is, to obtain access on any of the systems where identified vulnerabilities were exploited and to continue the testing process from the network location(s) of each compromised system. Quite often, this can be done from behind a firewall and will yield very different results than the same testing performed from the original network location. This step may well require repeating steps 1-5 recursively until all options have been exhausted. Note that step 5 is the least automated by the tools available for penetration testing, although there are tools that claim to do some or much of this.

Vulnerability Scanning vs. Penetration Testing

As penetration testing tools and processes were maturing, the need for a "lighter" process became apparent. In this context, "lighter" means a process that finds the first tier of security weaknesses but does not exploit them or attempt to gain further access. Thus was born a process known as vulnerability scanning.

In the previous process nomenclature, vulnerability scanning refers to steps 1-3. That is, it glances across a network and takes an inventory of computers and services found. Then it attempts to ascertain whether conditions exist that typically indicate the presence of vulnerabilities that are exploitable, but without actually exercising them.

Its chief benefits are that it is the least invasive of any of the common penetration testing processes and that it can be very quick. Additionally, it is the easiest penetration testing type of test to automate.

From a software testing standpoint, vulnerability scanning is nearly useless. At the very most, it can be used as a quick (and dirty) inventory scan of a production network. Of the penetration testing sorts of processes, however, vulnerability scanning suffers the most from its extremely weak test coverage.

"Black Box" vs. "White Box" Approaches

Another concept that is commonly touched on in penetration testing is so-called "black box" and "white box" approaches—often referred to as uninformed and informed testing, respectively. In a black box test, the tester starts with no knowledge whatsoever of the target system, whereas in a white box test, the tester gets details about the system and can assess it from a knowledgeable insider's point of view.

The argument in favor of black box style penetration testing is that the tester sees the system as an outside attacker might. On the other hand, a white box tester wouldn't need to waste any time learning about the system prior to testing it. Or so the rationale goes.

About the only practical argument in support of pure black box testing2 methods is to answer the question of how difficult (or easy) it would be for an external attacker to break into the system. However, no commercial penetration test is truly representative of an actual attack. For economic reasons, the test is usually compressed in time, which reduces the likelihood of seeing and exploiting chance or cyclical exposures. Attempts to avoid detection also mean that thorough vulnerability assessment cannot be accomplished. So, the greater the degree of black box testing incorporated into the test method, the greater the chance that vulnerabilities will be missed. So, though black box testing will usually identify exposures and demonstrate them, if it isn't combined with more comprehensive vulnerability assessment, it will not identify all exposed weaknesses.

Black box testing can be useful in helping justify new or increased budget for the security program. However, for all practical risk management purposes, and in particular for the purpose of using penetration testing during software development, black box methods offer, at best, marginal utility.

Continuous or Recurring Services

A final category of penetration testing and vulnerability scanning services has appeared in the past few years. We refer to this category as continuous, as its

principle of operation is to run a testing tool (or tools) on a target population on a periodic or even continuous basis and have the tool report changes in the environment, at least once an initial baseline is established.

While these services may well appeal to the CIO who wants to keep an ever vigilant eye on the network and computing infrastructure in an organization, they have little if any applicability when it comes to software development.

Common Pitfalls

We've seen how penetration testing tools and processes have developed and matured over time. We've also seen how these same tools and processes have been developed primarily for CIOs' purposes. Our primary concern here is how penetration testing can be applied to software development. With that in mind, there are several pitfalls that should be avoided in using today's penetration testing tools and processes. We address those pitfalls below.

Too Late in the Life Cycle

Perhaps the most egregious mistake made in penetration testing processes, from the standpoint of their applicability to software development, is that they're almost always applied far too late in the life cycle. In many cases, they're run only on production systems that have been operational for a significant period of time, so they're essentially irrelevant to software developers. However, even when penetration testing is done on "new" systems, it is almost always at the tail end of the life cycle, sometimes within days of a product going into a production state.

We should emphasize that the practices described here on the BSI portal are themselves process agnostic. Similarly, we don't mean to imply here a particular focus on one or more software development processes. Irrespective of what development process is used, penetration testing has traditionally occurred too late to be substantially useful to those that need it the most—the developers.

Among other perils inherent in this approach, any problems found in the software are probably not adequately fixable in time for the product to go live. The mere analysis process involved in finding the root cause of the problems can be daunting. For example, was the problem introduced as a "mere" coding bug or was it due to an inherent architectural flaw in the product's design? If a design problem is detected in a penetration test—penetration testing, after all, does not distinguish between design and implementation shortcomings of the product being tested—the remediation process may require a significant engineering effort for the system to be adequately fixed. Failure to do so might well result in trickle down failures that cannot be properly anticipated by the remediation team.

Lack of Business Risk Perspective

Most penetration testing processes and tools do little, if anything, to substantively address the business risks associated with the software weaknesses that they uncover. This is largely due to the fact that the tools and the testers view the target systems with "technology blinders" on, metaphorically speaking.

Although many testing tools and services claim to rank vulnerabilities in terms of technical severity, they do not typically take business risk into account in any significant sense. At best, the test teams conduct interviews with the business owners of the applications and the application architects in an attempt to ascertain some degree of business impact, but that connection is tenuous.

More importantly, however, is that the business perspectives, however limited, that these processes can determine are all post facto. That is, they make their business impact rankings after the test is completed, instead of using business risk to focus time and attention. This is a key shortcoming of penetration testing practices today.

In all fairness, there is an implicit business risk prioritization done here in terms of how CIOs and CSOs engage their penetration testing services. That is, they typically point the testers at particular business application networks rather than do broad scope testing of an entire enterprise. (The latter practice was more popular in the late 1990s and early 2000s.) This may well be giving them more credit than is due, however. Their business system targeting may well be a simple pragmatic reaction in the way of cutting the costs, rather than actually using business risk to prioritize and focus the efforts of the penetration testing team.

Checklist of Repairs Approach

Another commonly seen problem in penetration testing processes today is the "checklist" approach in addressing the security shortcomings uncovered. That is, the list of test findings is considered a simple checklist of items to be fixed, and once they're fixed, they are forgotten.

Treating the results as a mere checklist carries with it several pitfalls. For one thing, there is a significant danger of missing the underlying causes of the problems. How did the vulnerabilities get entered into the code base? Were they introduced in the design, or were they due to a simple coding error? Should they have been picked up by a code review? These are the types of causal questions that need to be addressed, in addition to fixing the list of problems uncovered in the testing.

An additional danger here relates to the above shortcoming: if the checklist of things to fix is inadequately ranked by business risk as the most important factor,

then you may well be wasting time and effort fixing something that doesn't matter nearly as much as something else.

Failure to Integrate With Existing Bug Tracking

Most software development organizations have existing and mature bug tracking processes and tools. However, few penetration testers integrate their test results with these existing tools. Instead, a common practice among penetration testers is to report on their vulnerability findings in the complete absence of how those findings map to code defects and what fixes should be made to address them.

This failure often has a negative effect on the development team, who are left to figure out the test findings and their mappings to the code base.

Test Coverage

In traditional software quality assurance testing, the test scenarios are judged by, among other things, their test coverage. One of the test measurement criteria that test coverage addresses is how much of the software gets evaluated in the test scenarios [Kaner 1996]. The testers, naturally, shoot for test cases that achieve as high a level of coverage as possible by exercising every single line of code in the application, as well as other means.

From that standpoint alone, penetration testing can only be viewed as being woefully inadequate. This is because penetration testing typically only exercises the external network interfaces to the application being tested. Even this assumes that the testing goes well beyond merely testing the underlying operating system and network environment to actually testing the application software interfaces directly.

Achieving adequate test coverage requires the test team to go well beyond what traditionally constitutes a penetration test.

Outside --> In

Closely related to the problems associated with test coverage is penetration testing's inherent outside --> in methodology. Even the name penetration testing implies that the tester is on the outside and attempts to enter the inside of the application. This immediately affects how much of the application the testers are able to see and probe at. It also drives a black box way of thinking in many cases that is ill suited to obtaining test results that are useful to application developers.

Finding Value in Traditional Penetration Testing

With all of the pitfalls and inadequacies of traditional penetration testing that we've listed, one might reasonably ask whether it is worth doing at all. Well,

there is value to be found, but it can be elusive if the testing is not handled carefully.

The primary benefit to doing traditional penetration testing is to ensure that the deployment environment where the application is being installed is adequately locked down and maintained over time. Although these things are typically in the realm of the CIO, they without a doubt can deeply impact—negatively or positively—the security of an application.

When done well, network- and host-based testing can be done on a deployment environment to look for human errors in file and directory permissions, user account privilege management, network services configuration, and so on. Similarly, they can also be done periodically to verify the adequate upkeep of system-level patches.

At an application platform layer, traditional penetration testing can be moderately useful at looking for common installation and configuration flaws, at least in some cases.

Within an application itself, traditional penetration testing can do little to assist in assessing the level of security. At most, application scanning tools serve, as one author posits, as a "badnessometer," meaning they can only measure how bad one's application is. They're far less effective at measuring how good one is. That said, traditional penetration testing can help highlight certain security stress points in an application, in particular where implementation errors have been made during the coding of the application.

In the next section, we talk about adapting penetration testing techniques to serve the purposes of software development far more effectively than traditional penetration testing.

ADAPTING PENETRATION TESTING TECHNIQUES TO SOFTWARE DEVELOPMENT

Earlier in this document, we saw numerous reasons why traditional penetration testing methods are ill suited for the purposes of software developers. Although this is indeed the case, there remains significant value to including penetration testing concepts in the software development process. After all, quality assurance driven software testing focuses largely on testing the software to functional specifications. Therefore, testing for negative states and areas of security compromise can be difficult for QA testers to build scenarios for.

So, to enumerate some of the positive aspects of penetration testing that are relevant to software development, we offer the following list as a starting point for consideration:

- unhampered by functional specification
- testing focused on direct user interfaces
- automation of attack resistance analysis methodology
- excellent at spotting deployment environment configuration mistakes

Despite this list of positive aspects, there remain significant hurdles to clear before penetration testing can truly be considered a value added team player for software development purposes. The remainder of this section discusses these hurdles and provides insight into how they might be cleared.

At its core, however, penetration testing should be employed to stress test the security aspects of a software system. That is, other testing processes are intended to identify the stress points, whereas penetration testing should ideally exercise those stress points for signs of breakability.

Early Life-Cycle Integration of White Box Process

For penetration testing to be effective to software developers, it needs to be started at as early a stage as possible in the development life-cycle process [Arkin 2005]. Further, for maximum effect, the penetration testing process should ideally be a white box one in which the testers have full access to all available development artifacts, including requirements, specifications, design, source code, and deployment specifications.

Although this is contrary to the status quo of today's penetration testing practices, compelling justification exists to support this approach. The principal justification for doing the testing as early in the life cycle as possible is quite simple; the development team will need adequate time to respond to the test findings which, as pointed out above, may include changes to the product's design. Late cycle testing may well still be useful and valuable, particularly for detecting human errors in the application's deployed environment configuration, but that is not the primary focus of this document.

The major justification for doing white box testing instead of the more traditional black box method is so that the test coverage can be maximized. A black box approach, by its very nature, focuses entirely on an application's interfaces to the outside world. Although these interfaces must be thoroughly tested—indeed, mistakes made here are most likely to be discovered and exploited by attackers—a more thorough and rigorous process is called for here. To that end, a penetration test designed for software development purposes should employ a much broader focus. They need to examine and test every interface, process, module,

object, data element, and so on—all the time looking for ways to subvert the application.

In designing the test scenarios, particular attention should be paid not only to user interfaces but programmatic interfaces between modules, data flow, environmental boundaries, and so on. These are the areas that are typically the most likely to be compromised in an attack. Further, they should go beyond the external user interfaces and include the inside à out, looking for potential weak spots.

Speak to a Different Audience

In our software development context, the CIO is no longer the primary customer—or, more to the point, the software developer is now the primary customer. Although you might occasionally find a CIO with a background in software development, it's probably safe to say that you won't often find a software developer with a background as a CIO. Therefore, succeeding with penetration testing for software developers requires some fairly significant changes to be made. For starters, a software development audience has very different interests than a CIO one. The finished product should thus be tailored to software developers' needs.

Effectively communicating with software developers means speaking their languages—spoken as well as programming languages. It requires the tester to understand the product being tested and the technologies used, along with their respective nomenclatures and jargon.

This is particularly the case since the testing is being started earlier in the life cycle as a white box process. Since the "box" is thus open to the test team, the resulting report documentation needs to be as specific to the particular issues uncovered during the test process as possible.

For example, whereas a traditional penetration test report might speak to a particular network service being exposed, a development driven one needs to weigh the issues with regards to the specific needs of the application. What module(s) need that service? What do they need the service for? How might the communications through that network service be protected without hampering the application's ability to do its work? Should we include this in the application's bug tracking system? And so on.

Test Team Should be Multidisciplinary

Traditional penetration testing has, as we discussed above, been the purview of the information security organization. While that has served the CIO well, it is less likely to be adequate for software developers' purposes. This is generally because the information security staff lack the necessary software development background. On the other hand, it is also generally the case that the software de-

velopment team will not have the necessary security background to be effective. For example, while the information security team is likely to be current with knowledge of attack techniques and tools, they are less likely to know all the programming languages, frameworks, etc. used by the development team.

Ideally, these complementary knowledge and skill sets should drive a composite team to design and execute the testing. Although in many cases internal cultural and political constraints may make this difficult to accomplish, the end result is far more likely to meet the application's needs if these cultural and political difficulties can be put aside.

Indeed, some large commercial organizations have begun deploying software security teams to act as champions or internal consultants to the tasks of secure software development. These teams are frequently made up of individuals who share information security and software development backgrounds.

Risk-Based Prioritization of Focus

As one might expect, a full scale white box penetration test like the type described above could end up being a massive effort that would consume enormous amounts of time, energy, and money. A test effort, however, can be less extensive and still useful to software developers. The key to accomplishing this is to prioritize the testing focus and process based on business risk. See also [Michael 2005] for a discussion on formal risk-based testing.

A primary result from an effective architectural risk analysis process should be a prioritized list of risks posed by the application. This prioritized list of risks should be used as a map and guide for the penetration testing team. Naturally, the highest risk areas of the application should receive the greatest amounts of attention in planning and executing the testing. The test team should thus begin by looking for the "nightmare scenarios" described by the architectural risk analysis team.

By carefully planning a penetration testing regimen in this manner, the test team's efforts can be optimized and focused on the issues of greatest importance to the business's concerns.

As one might expect from this sort of approach, it's quite possible that at least some of the highest priority architectural risk cases involve testing scenarios that are not addressable by way of an outside --> in test methodology. In fact, many of the risk cases will involve test scenarios that will require the test team to do testing from an inside perspective, within the application's innards as it were. Although many will consider this sort of testing to be outside the scope of a tra-

ditional penetration testing regimen, it is likely to be vastly more effective and meaningful to application risks that mean the most to the application itself.

Thus, for each of the identified highest risk areas, test scenarios should be developed that accurately and rigorously test the system and its components to see whether the risks are realistic or not. Scenarios should address entry preconditions and exit conditions so that the test evaluators can make informed decisions on how to address each of the realizable risks. For example, if a particular attack requires the attacker to have access to a local unprivileged account in order to escalate his privileges and compromise an application component, that precondition must be clearly articulated in the scenario as well as in the resulting report.

The test planning and execution should proceed in descending priority order based on the level of time and effort allocated to the testing process.

Tools Should Provide Launch Points for Further Probing

Although it is quite acceptable for the penetration testing team to use appropriate commercial and open source tools for their testing, the tools should serve largely as guides for additional probing. The additional probing will often require a much more manual process and should always be performed by experienced and knowledgeable engineers.

From a more pragmatic perspective, off-the-shelf tools for penetration testing will be severely limited in their usefulness if the test team truly adopts a white box testing methodology like the one described above. Indeed, to thoroughly do white box penetration testing would require the test team to make use of custom written and/or tailored tools. This is because most penetration testing tools have been written with the traditional information security penetration tester in mind. They tend to focus on network and host level vulnerabilities and not application-specific defects. Thus, it will frequently be necessary to test application and other interfaces that go well beyond what an ordinary user (or penetration testing tool) can see from a remote network location perspective. The test tools would thus look more like those used by unit testers than by traditional penetration testers.

Human Judgment Calls Are Essential

For the purposes of developing software, penetration testing is by necessity anything but a “fire and forget” sort of activity. Quite the opposite, it needs to be done by humans who can exercise judgment.

As in the case of QA testing, the best test engineers are the ones that can “smell” a problem and don’t easily give up. These are the people that use their intuition to drive their testing. They’re tenacious and relentless in their pursuit of prob-

lems in the software. Indeed, these are the same sorts of qualities that the best penetration testers need to have, particularly when not solely relying on off-the-shelf tools to automate the process.

Note that this is not to say that QA testers will necessarily make good penetration testers. This is because they need to couple these qualities with a broad and deep knowledge base of software attacks, attack patterns, and weaknesses.

Additionally, since the penetration testing process should inherently be a white box one, the test team should have a thorough understanding of how the application works across all of its components. They need to be as knowledgeable of the application as the development team. Therefore, sufficient time needs to be allocated so that the test team can immerse themselves in the design and implementation of the application.

Interpretation of Results

Not surprisingly, the results from a development driven penetration test will by their nature be substantially different than those of their information security counterparts. Further, as we noted above, a commonly made mistake in penetration testing is to treat the results as a checklist of repairs to be made. That same pitfall should be avoided here, of course.

At a most basic level, the testing should provide additional validation or elimination to the results of the architectural risk analysis. Thus, previously theorized risks should now be better understood. The understanding should include issues such as ease of exploit, time required, preconditions required, symptoms displayed, ability to detect, and so on. These are all important issues associated with each of the risks.

At this point, each risk needs to be carefully considered in the business context. That is, a business decision needs to be made for each one based on a fundamental question—which is worse, the defect or the cost to repair? In other words, should the defect be repaired or not? If repairing the defect is not feasible for some reason, can steps be taken in the application's environment to mitigate the damages if the defect is discovered and exploited by an attacker?

Additionally, whenever possible, the testing team should provide direct mapping of test results to the affected code modules, right down to individual objects, processes, etc., that are implicated in each finding. The results should then be integrated, as feasible, into existing software bug tracking tools and/or processes.

These are the sorts of issues that should be carefully thought through so that a principled business decision and course of action can be decided for each of the risks tested.

Consider too, some of the indirect results of the testing process. These include addressing of systemic issues in the design and implementation process as well as training for the development team. That is, how did each defect end up in the product? Was it due to the design or implementation team not being aware of a new attack technique or tool? Was it due to a simple coding mistake made by a coder? Each of these issues should provide a feedback loop into the development process so that the process and the team itself can continue to improve over time. That is, rather than making the common mistake of simply fixing a defect, spend the time to learn from the failure itself.

Penetration Testing Compared to Other Security Testing

Compared against traditional penetration testing, the testing processes described in this section start to significantly resemble other software testing processes such as risk-based and white box testing [Arkin 2005, Janardhanudu 2005]. Indeed, there are similarities and overlaps in scope that would need to be addressed in designing the appropriate testing program for any particular project or organization. However, differences remain that warrant separate discussion of each of the testing goals and techniques.

For example, penetration testing does not seek to validate the security functionality of a test target, at least not in any direct sense. This is in sharp contrast to the security testing described in [Arkin 2005]. Further, although risk inputs should be used to prioritize and focus penetration testing, its primary intent is still to break in to a target. It is just that the target, in the context of penetration testing as outlined here, may well consist of subcomponents of a system and not just the deployment network and operating system components that traditional penetration testing primarily focuses on.

It differs, too, from white box testing [Janardhanudu 2005] in that it doesn't closely examine the data flow through an application at the source code level. That is, even to the extent that source code can be included as one of the artifacts examined in penetration testing, it does so for the expressed purpose of breaking "into" it. That is, inputs such as source code should help guide the penetration tester where to focus his efforts, but penetration testing is not a source code review per se.

Traditionally, too, penetration testing teams have consisted of disinterested third parties, often outside contractors with no bias with regards to the test outcomes. White box and risk-based test teams, on the other hand, may well include various

personnel from the design and development teams. This difference, however, is more derived from tradition than anything else.

As stated above, the somewhat subtle differences pointed out here may well lead to mission overlap. In some instances, they are also sufficient to justify separate consideration.

SUMMARY

In summary, the practice of penetration testing has traditionally benefited not the software developers, but the IT Security staff. In order to be used effectively for software developers, penetration testing needs to be planned and performed quite differently. Among other fundamental changes, penetration testing needs to be insideàout to truly benefit software developers.

Additionally, penetration testing tools and processes should be selected and refined to better suit software development. Emphasis should be placed on tools that are highly customizable and can easily be extended with application-specific probes and "attacks."

Personnel performing the testing, too, should be appropriately trained on software attack technologies and should have a deep understanding of the software being developed. The status quo of using penetration testers who do not have an adequate understanding of the software architecture or implementation is not sufficient for the purposes outlined here.

When these practices are followed, there is without a doubt significant value to be found in penetration testing. Traditional black box penetration testing, on the other hand, provides little value to the software developer.

BIBLIOGRAPHY

[Allen 2008]

Allen, J.; Barnum, S.; Ellison, R.; McGraw, G.; Mead, N. Software Security Engineering: A Guide for Project Managers. Boston, MA: Addison-Wesley, 2008 (ISBN 978-0-321-50917-8).

[Arkin 2005]

Arkin, B.; Stender, S.; & McGraw, G. "Software Penetration Testing." IEEE Security & Privacy Magazine 3, 1 (Jan-Feb 2005): 84–87.

[Farmer 1993]

Farmer, D. & Venema, W. "Improving the Security of Your Site by Breaking Into it." http://www.fish2.com/security/admin-guide-to-cracking.html

[Herzog 2006]

Herzog, P. "Open Source Security Testing Methodology Manual." Institute for Security and Open Methodologies, September 2006. http://www.isecom.org/osstmm.

[Foster 2006]

Foster, N. "Re: ddj: Beyond the badnessometer." Internet mailing list posting, SC-L@SecureCoding.org, July 2006.

[Fyodor 2006]

Fyodor. "Top 125 Network Security Tools." Insecure.org http://sectools.org/, 2012.

[Janardhanudu 2005]

Janardhanudu, G. "White Box Testing." Build Security In, 2005.

[Kaner 1996]

Kaner, C. "Software Negligence and Testing Coverage." Software QA Quarterly 2, 2 (1995): 18.

[McGraw 2006]

McGraw, Gary. Software Security: Building Security In. Boston, MA: Addison-Wesley Professional, 2006 (ISBN 0-321-35670-5).

[Michael 2005]

Michael, C. & Radosevich, W. “Risk-Based and Functional Security Testing.” Build Security In, 2005.

[Spafford 2003]

Garfinkel, S.; Schwartz A.; & Spafford, E. Practical Unix and Internet Security, 3rd edition. Sebastopol, CA: O’Reilly and Associates, 2003.

[Wysopal 2007]

Wysopal, C.; Nelson, L.; Zovi, D.; Dustin, E. The Art of Software Security Testing: Identifying Software Security Flaws. Boston, MA: Addison-Wesley, 2008 (ISBN 0-321-30486-1).

This material is based upon work funded and supported by Department of Homeland Security under Contract No. FA8721-05-C-0003 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center sponsored by the United States Department of Defense.

DM-0001120