Penetration Testing: A Duet

Dr. Daniel Geer and John Harthorne
@Stake
dgeer@atstake.com

Part I: Penetration Testing, Looking East from 50,000'

1 Art v. Science

Penetration testing is the art of finding an open door. It is not a science as science depends on falsifiable hypotheses. The most penetration testing can hope for is to be the science of insecurity - not the science of security - inasmuch as penetration testing can at most prove insecurity by falsifying the hypothesis that any system, network, or application is secure. To be a science of security would require falsifiable hypotheses that any given system, network, or application was insecure, something that could only be done if the number of potential insecurities were known and enumerated such that the penetration tester could thereby falsify (test) a known-to-be-complete list of vulnerabilities claimed to not be present. Because the list of potential insecurities is unknowable and hence unenumerable, no penetration tester can prove security, just as no doctor can prove that you are without occult disease. Putting it as Picasso did, "Art is a lie that shows the truth" and security by penetration testing is a lie in that on a good day can show the truth. These incompleteness and proof-by-demonstration characteristics of penetration testing ensure that it remains an art so long as high rates of technical advance remains brisk and hence enumeration of vulnerabilities an impossibility. Brisk technical advance equals productivity growth and thereby wealth creation, so it is forbidden to long for a day when penetration testing could achieve the status of science.

That penetration testing is an art means that there are artists. In deference to those artists, they range from virtuosos to mules. At the low end, automation (tractors) is replacing brute labor (mules). Automation is the handmaiden of commoditization, and there is little doubt that the penetration field is fully commoditized at the lower levels of art. At that low level, scanning systems steadily expand the scope and coverage of what they automate. That those same scanning tools can be deployed for evil purposes is irrelevant unless you are in the newspaper business. As Sherlock Holmes said to Watson (holding a scalpel), "Is it not surprising that the tools of healing and the tools of crime are so indistinguishable?" No, it is not surprising - a good tool is a policy- neutral force multiplier and it is intent, that is to say character, that determines the outcome of that force multiplication. Penetration testing is therefore good or bad depending on the intent of its practitioner and of the recipient of its results. We confine this article to penetration testing where the intent is good (the only kind one has to pay for).

2 Characterization and Specialization

Successful penetrations can be characterized as the illegitimate acquisition of legitimate authority. As such, a successful penetration will yield the ability to command network facilities to do other than what their owners expected them to do, to gain the full or at least substantial control of a host in a way normally reserved for trusted operations staff, or to acquire the management interface of an application or the functional equivalent thereof. In each variation, the authority obtained is not otherwise available to the person, place or thing which is performing the penetration whether that penetration be a test or a live fire attack. As the reader likely already knows, the most successful penetrations are those which materially decrease the labor required for a repeat visit and which are silent (alarm-free) in the process. Penetration testers at high levels of art will, therefore, attempt not only to gain access but to gain repeatable access at low/no probability of detection on those repeat visits. Though slang is never eloquent except by inadvertence, the terminology "owned" (Øwned) is rather apt when the penetration artist can not only get in the first time but can also get in at will on a repeat basis without detection.

Specialization is, for any field, a consequence of expanding knowledge and the accumulating complexity thereof. Where that knowledge is itself knowledge about complexity, as it is here given the fundamental axiom that complexity is the chief enemy of security, the growth rate in (knowledge) complexity compounds and the rate of needful specialization accelerates. Ergo, penetration testing is and must be rapidly specializing and its practitioners would be worthy of criticism were they not specializing. Network penetrations are already a clear speciality as can be seen in the array of low end products that perform the commoditized bulk of a penetration testing regime at the level of repeatability and low labor cost that are the hallmarks of a defined speciality. Some of these are hand held and some are remote actuators. Some make a bargain with a thorough look at the most likely causes of successful penetrations while others bargain that an inventory of everything worth looking into is a better value. Some require an expert practitioner to interpret, some condense the report to the level of the reader's skill and competence. Some evolve more or less continuously while others are stable. Network penetration tests are a proxy for two kinds of risk, losses of communications availability and losses of whatever part of the overall operational integrity depends, implicitly or explicitly, on effective perimeter control. If the network penetration test is being done for some other purpose, the results will tend to mislead.

A virtually similar situation exists with host penetrations and tools arrayed around hosts for penetration testing, but with the operating system rather than the protocol now uppermost. Of course, some operating systems are more worthy of testing than others both based on inherent risk due to complexity and history of risk due to inattention to security as a design characteristic. Such distinctions are not the subject of this essay, but they are real and likely to remain so for a very long time inasmuch as design choices made badly are difficult to fix when the installed base grows beyond some threshold long since passed for every operating system in common use. With host targeted penetrations, it is the facilities of the host that are sought and hence it is the power of the host that calibrates the level of effort that should be expended in testing or which will be expended by genuine attackers. Note that "power" here is a subtle concept; it is not merely the horsepower of some component but also the trust relationships that host has. As it will always be true that for any host there must exist some level of unchecked power such that the more serious aspects of systems administration can be done under diminished operability, host penetration tests are a proxy for the estimation of cascade failure of authorization integrity. As with network penetration testing, if the host penetration test is being done for some other purpose, the results will tend to mislead.

Applications are a slightly different kettle of fish as they are inherently difficult to define when one is serious about defining them. As applications expand by feature accretion and by the kinds of labor- dividing, redundant provisioning on which business continuity increasingly depends, a sidebar increase in complexity can easily exceed any one person's ability to understand the whole of the "application" that is delivering the "customer experience." While networks can be complex, the idea of a network operations center and crisp metrics on what constitutes effective network operation are at least well enough advanced that residual questions about "What is the real extent of my network?" are where the action is. Not so with applications, particularly so as applications are rarely built from the ground up in their entirety but rather represent adaptive re-use of numerous (or perhaps innumerable) libraries, caches, roles of authority, external identity control, and so forth. As such, an application pen test is much more akin to exploring the difference between what is thought to be in place and what is actually in place, viz., to run the application down paths that were not intentional in the application's design and implementation. An application pen test is less easy to automate except for a few classes of classic failures, e.g., session replay or crash-vulnerability to hostile input. Application penetration testing, in other words, is a young and abstract art attracting young and abstract artists at the moment of this essay. An application penetration test is a proxy for the illegitimate use of legitimate authority, a subtle but important difference with the illegitimate acquisition of legitimate authority. Where legitimate authority can be used for illegitimate purposes, there is no implication that the defined functions, that is to say product requirements, are failing. Rather the successful application penetration tester is showing that other code paths outside of the required code paths exist and that these code paths are reachable by the tester. Application penetration testing is, therefore, more like embezzlement and less like a stickup. As with the other two species of penetration testing, if the application penetration test is being done for some other purpose, the results will tend to mislead.

All three types of penetration testing have separate reasons for their continued existence even as they evolve differently. However it is fair to say that application testing is today where the ferment is because of trends in application deployment. To state the obvious, applications are federating - they are becoming conglomerates of pieces running in multiple locations under multiple ownerships and liabilities. In retail commerce, to choose an example, catalog, payment, fulfillment and customer service are often entirely different outsourced functions, each relying on network delivery strategies that involve independent hosting facilities, distributed network caching, and roll-with-the-sun handoffs of back office functions. An application may really and truly be the business for all intents and purposes yet the business as a legal entity may not own, control or operate any of the application. If for no other reason, the pressure of applications of this composite sort on the definition of a network perimeter is to dissolve that perimeter, a trend that is widely underway even before the impending tsunami of "web services" (which, with remote procedure calls carried on HTTP will defy even stateful content inspection as a security strategy) extinguishes the mirage of a corporate perimeter.

3 Time Line and Drivers

The fundamental irony of penetration testing is that the value received by the client is itself subtle but the clients who ask for penetration testing as their primary security activity are but rarely thinking subtly. The penetration artist may or may not endeavor to correct this, but so often the hope of the client is that the penetration tester will fail to penetrate. The better the penetration tester the less likely it is that s/he will fail to penetrate and, so the logic goes, the better the result and the greater value the client receives should the penetration tester fail. At that level of understanding, the value proposition for the client is that the penetration tester is selling their failure. The value of that failure is greater the less often the penetration tester fails, i.e., the less likely the penetration tester is to fail based on skill and history the higher value the client has obtained if and when the penetration tester does fail.

For the tester, one cannot sustain a high price for penetrations unless one fails rarely (proving you are good) but at the same time the satisfaction the customer receives from that testing is proportional to the degree the penetration tester fails. In penetration testing, then, one has the classic problems of selling something (failure of the penetrator) that is valuable fundamentally in proportion to its scarcity, hence revenue cannot be scaled up by expanding the supply of goods for sale as that would defeat the scarcity on which pricing is based. In that sense the question of "How much penetration testing is enough?" cannot be answered without first picking either the client or the tester point of view: The client wants enough testing for the result to be advertising-ready but not so much testing that the tester fails to fail. By contrast, the tester wants enough testing to fail to fail and thereby preserve their reputation as an entity whose failure is worth paying a premium for, but if the client is lame not so much testing as for the effort to get boring or failing to fail look too easy. No wonder optimization is a remote possibility.

While there is wisdom in that ancient English aphorism that "It is the poor carpenter what curses his tools," in penetration testing the best carpenters make their own tools. These tools are part labor productivity for the penetration tester - and advancing labor productivity is ever the core supply-side defense of profit margins - and part complexity rigging. These bespoke tools are, if anything, the intellectual content of the penetration testing field and the flux of these tools into the marketplace measures the stage of commodotized market development. Password crackers are a fine example - who would write one today now that first rate crackers are available for so little money that all you are really paying for is a user interface? Network service inventory takers are just as fine an example - who would write one of these when the Internet is so full of them that over 10% of total Internet traffic is the sort of low level scans these tools are built to do? In some sense, the point at which an artist's intuition moves beyond mere suspicion and s/he writes down (codes) what s/he knows in the form of a tool the state of the art is advanced – not everywhere and at once, but in the sense that the future is already here, just unevenly distributed. It is the tools of the artist class that define the state of their art, even if they will not show them to you.

The future is simple: The target of penetrations will be ever further inside the enterprise as the corporate perimeter dissolves and inside versus outside has ever less practical difference, i.e., for there to be a penetration there has to be something to penetrate and the corporate network perimeter is as interesting to penetrate as a month-old whale carcass. Penetration testing in the main will look more and more like quality assurance in that it will look more and more like falsifying hypotheses that such and such a flaw is present (by attempting to demonstrate that it is present and failing to do so) and less and less like a voyage of discovery about what hitherto unknown flaws might be present, excepting for the top end artists. There is always room at the top, but probably not much place else. The artists who can reliably estimate the level of effort to accomplish a penetration are the ones who will add value because they can chart the steepness of the curve of tradeoff costs as one moves ones worry from idle sociopaths to committed opponents to as-yet-trusted turncoats. The ones who are just taking inventory can be replaced with a button. The ones who can quantify in a way that makes risk management advance are the ones who will survive.

Part II: A Portrait of the Artist as a Penetration Tester

4 The Five W's of Application Penetration Testing

As Application penetration testing is the least commoditized of the major penetration test specialities, its future is the least distributed, and a closer examination of the current incarnation of its future is therefore warranted. Papers extrapolating specific exploits against specific applications abound on the web and elsewhere. There is little wisdom to be gained by rehashing such ephemeral morsels here. Instead, we will focus on the less immutable aspects of application penetration testing, and will expand on the justification for *pentesting*, as it is called, its methodology, its major players, its current and future placement in the development lifecycle and its area of prioritization and focus. In short, we will examine the why, what, who, when and where of application penetration testing. Without attempting to dissect and categorize the many species of application here, we will focus largely on web applications, though most of the principles discussed here apply, at least in part, to other types of applications as well.

4.1 Why

So we stop to ask ourselves, "Why should I pay someone to break into my own applications?" Especially if, as described in the first part of this article, penetration testing is at best a science of insecurity, pitting the skill and hopes of a security professional against the skill and hopes of the developer. Application penetration testing continues to yield a tremendous Return On Security Investment (ROSI) precisely because the future of application security is still so unevenly distributed. The focus of security consciousness has only recently shifted to applications, owing to the assumption that applications are the slaves of infrastructure and that it is the networks and hosts that define the boundaries of the corporation's digital assets. Indeed, we continue to refer to the corporate homeland as the "corporate network", not the "corporate application mass" (and not just because it is phonetically more pleasant). In any case, it is the application that reaches out across the Internet into every connected human's living room. So while we may think of firewalls, network ACLs and host defenses as our corporate walls and ceilings, the applications represent our doors and windows and are therefore becoming both the target of attackers and the focus of security professionals.

A quick glance at the lamentably few statistics on digital (in)security provide a sobering reminder of just how critical it is to buckle up before putting the corporation onto the information super highway. According to the CSI/FBI survey on computer crime and security (http://www.gocsi.com/press/20020407.html) which, while not without limitations is more likely to understate the extent of criminality than otherwise, ninety percent of respondents (primarily large corporations and government agencies) detected computer security breaches within the last twelve months. Eighty percent acknowledge financial losses due to computer breaches. Thirty-eight percent suffered unauthorized access or misuse on their Web sites within the last twelve months, with twenty-one percent admitting that they really didn't know whether there had been any unauthorized access or misuse, because they either weren't monitoring their sites for abuse or weren't sufficiently confident that they were monitoring those sites successfully.

While these survey based numbers are chilling, they are becoming mundane through repetition and are fairly frequently shrugged off as qualitative and/or personally irrelevant since they are based on a voluntary survey. The more quantifiable statistics from the honeynet project provide more prescient commentary on the world of digital abuse. Those who are unfamiliar with the honeynet project, should visit their site at http://www.honeynet.org – it is as interesting as it is enlightening. As a quick synopsis, the organizers of the honeynet project implemented a clever scheme for tracking and monitoring black hat activity passively. As their site indicates, a honeynet is a network "similar to a fishbowl, where you can see everything that happens inside it," a highly monitored network that is connected to the Internet but is not advertised actively in any way; in fact, it consists of nothing but a few IP addresses. All activity in the network therefore represents either the collision of curiosity and coincidence or an attempted attack. The captured activity illustrates the tools, tactics, and motives of the blackhat community. Some statistics taken directly from the honeynet project's web site:

- Between April and December 2000, seven default installations of Red Hat 6.2 servers were attacked within three days of connecting to the Internet. Based on this, we estimate the life expectancy of a default installation of Red Hat 6.2 server to be less then 72 hours. The last time we attempted to confirm this, the system was compromised in less than eight hours. The fastest time ever for a system to be compromised was 15 minutes. This means the system was scanned, probed, and exploited within 15 minutes of connecting to the Internet. Coincidentally, this was the first honeypot we ever setup, in March of 1999.

- A default Windows98 desktop was installed on October 31, 2000, with sharing enabled, the same configuration found in many homes and organizations. The honeypot was compromised in less than twenty four hours. In the following three days it was successfully compromised another four times. This makes a total of five successful attacks in less than four days.

The lack of production applications in the honeynet precludes revelations about the shift of attack activity to the application space, but the statistics clearly underscore the existence and tenacity of the black hat. Unrelated empirical evidence clearly indicates a shift in attack methodology from passively exploiting exposed network functionality, i.e. mounting an exposed share, to actively abusing networking applications, such as writing buffer overflows to subvert web servers or application servers. Also, it is important to note that the honeynet statistics underestimate the real danger to corporate applications since they capture only opportunistic activity, not targeted activity.

Many developers remain nonplussed by attack statistics, arguing that their corporate development process is highly optimized or that their applications do not expose critical functionality or that the corporate firewalls and Intrusion Detection System (IDS) will protect the applications from any real danger or that the Quality Assurance process will find any errors in implementation. Penetration testing is the key to resolving this debate and is critical for determining how and where to integrate defensive tactics in application development and deployment. First of all, while the application development process has been highly optimized over the last 20 years, this optimization has been largely focused on work flow management, i.e., on enhancing the ability of developers to collaborate effectively and efficiently on development, and on performance enhancement and feature expansion. The increasing pace of development has ensured that security and other hitherto unmarketable "features" have remained on the fringe of the development lifecycle. Secondly, application vulnerabilities can be leveraged to gain not only the intended authority of the application, but also the oft overlooked "power" of its service user. Witness the efficacy of Code Red and NIMDA, or even the URL encoding vulnerability in IIS 4.0/5.0. In all cases, an application level vulnerability (admittedly in an infrastructure related application) led to compromise of the host itself, thereby providing an avenue into the network, wreaking significant, well-document havoc. As for firewalls, they explicitly allow application traffic, essentially ignoring its contents or, at best, perusing it for a few generic, well-documented signatures. Additionally, the rapid expansion in application traffic, demand for high availability and increasing use of SSL all collaborate to render IDS systems ineffective or, at best insufficient, in deterring significant application centric attacks.

As for QA testing, it is and has always been a means of testing for "expected" functionality. Penetration testing is very much the opposite approach. A penetration tester is constantly probing for unexpected functionality. The skill sets required for each field are entirely different, and a million years of quality QA testing, though it may potentially highlight implementation errors with security implications, cannot hope to replicate the security knowledge to be derived from one week of penetration testing. Normal usage too will not replicate the process of an attack. Who, in the course of either QA work or normal application usage, would ever have thought to request a URL containing `..%255c..%255` or to submit 65,000 characters in a form field, much less an HTTP header? Clearly, until security design is advanced and common, penetration testing is a distinct activity legitimized by both the prevalence of digital attacks and the lack of adequate preventative measures elsewhere in either the application development or deployment lifecycle.

4.2 Who

Now that we understand the need to penetration test and that we have clarified the distinction between QA testing and penetration testing, we have begged the question of who should perform the testing. As stated earlier, there are both mules and artists out there. Clearly artists are unique, and clearly their skills are not yet commodities and are therefore not reducible to succinct description. Nevertheless, good penetration testers share several qualities based on the nature of the art they practice and we shall attempt to summarize the critical qualities they share.

First of all, it is essential that a penetration test professional be technically savvy. The requisite extent and breadth of this savvy can be argued, but facility with basic application technologies is a requirement, and specialization beyond general expertise is a significant advantage. Most, if not all of the application penetration testers who consistently "fail to fail" have extensive experience as developers, such that they anticipate design and implementation errors and can identify application structures based on the technologies in use and the failure modes discerned. Many successful application testers have at least some experience as system administrators, such that they are adept at leveraging vulnerabilities where applications and infrastructure intercept and such that they are able to leverage minor errors in either area (infrastrucure or applications) to create or enhance an advantage in the other area. Many test professionals have also been trained in the methodology and basic techniques of penetration, either through a corporate training program, government program or university program. A number of consultants at @stake have been formally trained by the National Security Agency (NSA), and like the Marines who are trained in hundreds of ways to physically kill opponents, these testers have an extensive arsenal of exploits, tactics and strategies for attacking both applications and the infrastructure that house them.

Still, not every tech savvy individual would make a good penetration tester. Creativity is an absolutely vital distinguishing characteristic. Because, as indicated earlier, "the list of potential insecurities is unknowable and hence unenumerable" and because penetrating defenses amounts to the "illegitimate acquisition of legitimate authority", penetration testing represents an art of discovering the unknown and revealing the assumptions inherent in someone else's creative pursuit. In a sense, it is the art of proving the existence of the unexpected. Also, as illustrated in the discussion on the irony of penetration testing, the field of qualified penetration artists narrows rapidly as the artists ply their trade since every vulnerability identified, classified and described expands the lists of known insecurities and arms the developer with another defensive tactic, or vindicates a strategic defense, thereby increasing the creativity required for future penetrations. Clearly it is an irony of all businesses that their ultimate goal is to obsolete themselves before the competition does, but it is particularly poignant in field where professionals are tasked both with undermining a body of knowledge and with contributing to it at the same time. In any case, the successful tester must be able to think in ways that others do not, since it is his/her very task to illustrate the path that everyone else overlooked. There is almost a child-like energy and curiosity required to behold a hammer as a potential shovel, or a chair as a potential table, or an authentication routine as a route to a command prompt or a SQL interpreter or both.

This ability to think differently and approach a problem playfully is essential, but it must also be tempered with a gift for discipline and organization since, as the writer and literary critic Norman Podhoretz puts it, "Creativity represents a miraculous coming together of the uninhibited energy of the child with its apparent opposite and enemy, the sense of order imposed on the disciplined adult intelligence." As it matures, penetration testing becomes increasingly rigorous and methodological. The drive to penetrate increasingly secure applications in relatively short amounts of time enforces a systematic approach to gathering information, verifying known vulnerabilities, hypothesizing new vulnerabilities and prioritizing analysis. The need for order and focus results in the creation of both tools and methodologies for creative analysis. Tools represent the empirical expression of technical lessons learned and, as suggested above, the best tools are always home made. This is especially true in the application space where the uniqueness of attack targets almost always demand customized tools. Application penetration testing tools generally consist of software proxies, vulnerability scanners, fuzzers, port scanners and sniffers, but the penetration test professional will also learn to use non-security specific tools, like browsers and debuggers, to their advantage. Often simple PERL scripts prove to be a tester's greatest asset.

Though successful penetrations can be very exciting, the majority of the testing process consists of failure. As Albert Einstein noted "I think and think for months and years, ninety-nine times, the conclusion is false. The hundredth time I am right." Indeed, if success were guaranteed, the process wouldn't be referred to as penetration "testing," it would just be called "penetration." The constant hurdles and elusiveness of success mean that test professionals must exhibit humility, determination and patience. Despite the difficulty of finding tech savvy and creative individuals, it is probably this requirement that so severely limits the number of adequate penetration test professionals in the field today. Too many testers anticipate easy success, only to discover that success can be difficult to achieve, especially when analyzing applications written by experienced, security conscious developers who have learned from previous tests and have integrated their knowledge into the development lifecycle. Point and click tools find only the lowest hanging fruit -- the fruit higher up the tree is much more rewarding, but it can be difficult to reach.

In any case, we have identified a good penetration test professional as technically savvy, creative, disciplined, and determined. Clearly, these qualities are difficult to come by in isolation from each other, and are obviously all the more difficult to find in combination. Still, they are beneficial qualities in and of themselves and are not impossible to find within existing development teams. Additionally, technical know-how and organization are skills that can be learned. So, assuming we have found several individuals who fit the proverbial bill, we will want to examine the pros and cons of leveraging internal capabilities versus outsourcing our penetration testing. Clearly there are advantages and disadvantages to each approach. For the purposes of our discussion, we will consider "outsourcing" to mean that the penetration test team will consist of individuals that are unrelated to the developers, i.e., they may belong to the same organization, but not the same development team, or they may belong to a different company altogether.

The critical advantages of outsourcing penetration testing include objectivity and specialization. It is a widely observed and understandable reality that few companies, if any, will argue that their products are inferior to those of their competitors. There can be widespread agreement, however, that some products are, in fact, clearly less desirable than others. It stands to reason, therefore, that some people are either consciously deceiving the public or are deceiving themselves about the relative quality of their products. Now let's extend this analogy for a moment to developers and their applications. It can be easily verified that developers will tend to defend their code in the face of criticism and that some of this defense is unjustified given the errors found in software today. Given the instinctual defensiveness of the developer relative his/her code, it is largely unreasonable to expect a developer to indicate precisely why and where his/her code is broken or faulty. It is fairly unreasonable to expect a developer to recognize where his/her own expectations can be undermined, since it is precisely those expectations that will guide his/her examination of the code to begin with. External parties have the distinct advantage of having less of themselves invested in proving that the application is really rather clever and well written after all. Additionally, using a dedicated team of penetration testers has all the advantages that accompany specialization. Speed, cost and exhaustiveness are perhaps the most obvious and desirable of these advantages. The possession of tools and methodologies clearly facilitates a more rapid analysis, which in turn reduces the cost of the analysis since, it can be argued, time is money (which is more true in the world of software development than it is in most fields). The tools, methodology and expertise of the team additionally ensure that analysis will be more thorough, meaning both that more errors will be found and that confidence is more justified where no errors are found.

That being said, there are also advantages to having developers test their own code. Where extreme sensitivity is an issue, for instance, it may make sense to keep the code base exposed to as few people as possible. The NSA is not known for outsourcing code review or penetration testing, for example. Obviously, such work could potentially be "outsourced" to another department within the same organization, or precautions could be taken to outsource such work only to trusted individuals and to limit their likelihood of accidentally or intentionally revealing sensitive information by enforcing a "clean room" approach to the test procedures. Still, it is conceivable that sensitivity might contribute to an unwillingness to outsource. Cost may be another such mitigating factor. While specialist will likely work faster and therefore produce results in a more cost-effective manner, they will also charge real dollars for their work. Leverage of talented individuals in-house may, by producing fewer entries on the balance sheet, represent a preferable modus operandus. Finally, and perhaps most importantly, using developers as a critical resource during the penetration testing process has the distinct advantage of introducing the home court advantage. Developers are liable to know the weakest spots in an application's code base and will be intuitively familiar with its design and technology base. This type of knowledge is like gold to attackers, be they benign or malignant. The information gathering phase is one of the costliest and least productive phases of the penetration testing process. Nevertheless, it is one of the most critical phases. Leveraging insider knowledge can mean the difference between finding zero vulnerabilities and finding dozens. All in all, the most effective approach, i.e. the approach most likely to find the most vulnerabilities in the shortest amount of time, appears to be a hybrid approach. Ideally, several "outsourced" test professionals work in conjunction with a technical lead from the project to identify the most likely trouble spots and then analyze the product to determine whether vulnerabilities exist and where.

4.3 What

Now that we are convinced of the need to penetration test and we know whom to call, let's examine the nature of the service to which we are committing. First of all, there are two basic categories of penetration testing, white box and black box. In actuality, all penetration testing is really gray, but as the particular shade of gray is so clearly dependent on the degree to which it approximates one end of the spectrum, it is convenient to speak of the spectrum's anchors as if they are absolute. In any case, black box testing is intended to most closely replicate the attacks of a remote, uninformed attacker. Since the prevalent attack scenario stereotypes generally involve remote non-employees, this is a popular approach. Essentially, the penetration tester is only given publicly available information about the target, perhaps only an IP address. The advantage of this approach is that the tester is forced to gather as much information about the target as possible. It is common practice, for instance, to scour message boards for "assistance needed" emails from developers at the company in question. Developers frequently post code snippets for problem areas, and these snippets are sometimes very revealing, occasionally containing passwords or other sensitive information. In the very least, they indicate the technologies being used by the company, which may spur further investigation by the tester. Additionally, a company's help wanted ads generally reveal both the technologies they are using and the areas where they may be weak, i.e., where they need help. Consider for a moment an ad that reads "Senior Java developer wanted. Must be familiar with IBM's WebSphere software platform and be experienced coding web services that connect to Oracle databases." While this doesn't equate to vulnerabilities, it may indicate which bag of tricks to try first. Error messages and debugging functionality are the other more obvious locations to look for information leakage. Depending on the scope and intensity of the engagement, social engineering may be called into play as well.

White box testing differs from black box testing in that the testers are given near total access to information about the application they are attacking. This information includes technology overviews, data flow diagrams, code snippets and access to developers and business leaders. Clearly, white box testing is more likely to reveal vulnerabilities that might not be as obvious to the casual onlooker. It is also likely to produce results sooner. Additionally, a white box test will more closely mimic an internal attack and, contrary to common perceptions, these are the most likely. When it comes down to it, there is clearly more bang for your buck in a white box test. That being said, a white box tester is very unlikely to spend hours searching the web for information leakage about your product, if you have already given that tester all the information there is about the application, meaning you may not discover that your data flow diagrams have been posted on a cracker mailing list for over a month—clearly a piece of information you may want to know. Frequently, clients will engage a penetration testing team to employ a two-tiered approach, i.e., black box for one week, then white hat for one week, or have one team doing black box testing while another performs white box tests. The diagram below demonstrates an overall methodologic approach of an expert application pentester.

Regardless of what general approach is adopted, there are three main stages to the penetration testing:

- Prepare
- Analyze
- Document and Improve

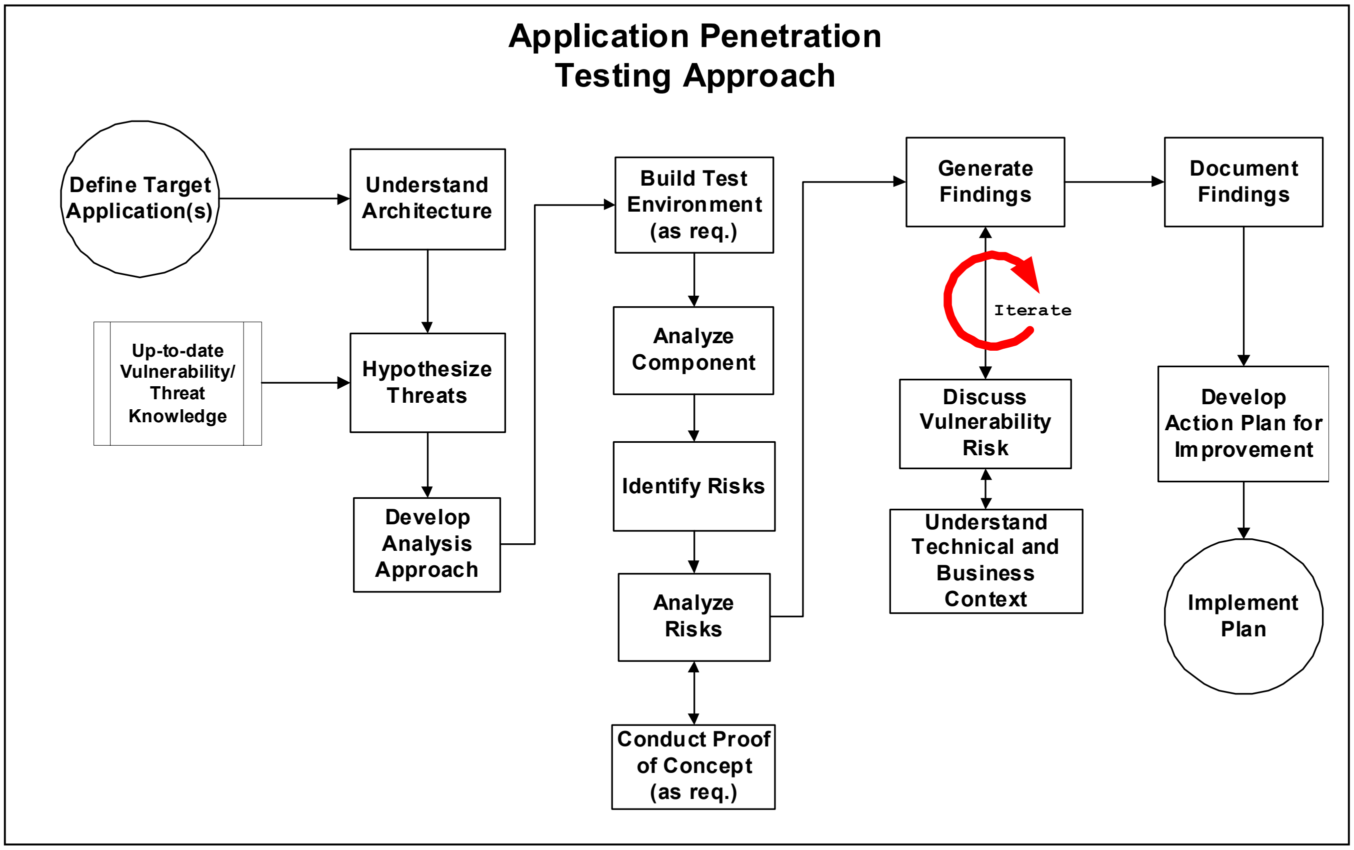
4.3.1 Prepare

The preparation step is frequently overlooked – it is extremely important to identify the scope and extent of the engagement. Penetration testing is by its very nature invasive even in its most innocuous forms. Accidentally targeting the wrong application or interface can have severe legal ramifications. Additionally, clients generally do not expect or want testers to bring production systems to their knees during peak hours, so it is vital that expectations be set about what will be attacked, when, from where and how. Administrative tasks like assembling a team, gathering documentation, acquiring test accounts, reserving equipment, etc. also fall under the preparation phase.

Information gathering also occurs at this stage. Obviously, this step varies in length depending on the size of the application to be tested and the "grayness" of the tests to be performed. If data flow diagrams can be acquired or produced faithfully based on information gathered, formal threat modeling may take place in this stage as well. Approaches to threat modeling vary greatly, but they generally include an enumeration of all application users, their access points and privilege levels, followed by an analysis of the process and privilege boundaries within the application itself. Analysis may then be prioritized on those interfaces and user paths that involve the sharpest difference in privilege levels, e.g., an interface whereby a remote, unauthenticated user influences a process that is running as the system user is probably of more immediate concern than an interface that allows an already authenticated administrator to influence a process running as system. However the prioritization and delegation of analysis occurs, it will greatly influence the later stages of the test process.

4.3.2 Analyze

The analysis stage is what most people envision when they think of penetration testing. It is here that testers


Application Penetration Testing Approach
Define Target Application(s)
Understand Architecture
Up-to-date Vulnerability/ Threat Knowledge
Hypothesize Threats
Develop Analysis Approach
Build Test Environment (as req.)
Analyze Component
Identify Risks
Analyze Risks
Conduct Proof of Concept (as req.)
Generate Findings
Iterate
Discuss Vulnerability Risk
Understand Technical and Business Context
Document Findings
Develop Action Plan for Improvement
Implement Plan

attempt to acquire and control legitimate authority illegitimately, i.e., this is where "hackers" attempt to Øwn the application and its host. Errors are generated where possible, unexpected input is supplied, interfaces are assaulted, protocols are examined and altered, cookie contents are abused, tools are employed and hopefully, for the tester at least, the application falters or stutters or falls and, hopefully, when it does, it sacrifices control to the tester. There is no need to further elaborate on the specific tactics used for analysis since that discussion is already widespread and exceeds the scope of this article. It is important to note here only that there is significantly more documentation during the analysis phase than most people would expect and that the analysis stage represents less of the total process than most people would imagine. Again, this is true by design and underscores the need for disciplined test professionals who are not solely and entirely motivated by a thirst to Øwn Øwn Øwn.

4.3.3 Document and Improve

Procedurally unglamorous, but nonetheless vital, the documentation of vulnerabilities and the identification of both strategic and tactical defenses requires both business and technical acumen. At first glance, the process appears rote and fairly unintelligent. Notes are gathered from all of the testers and formalized into a standard table. Templates are completed so that empirical evidence is presented intelligibly, indicating both the effects of vulnerabilities and their likely causes. Vulnerabilities may be classified into categories, charts may be produced for easy digestion, exact timelines of penetration test activities are likely to be built for comparison with log files, etc.

As unglamorous and routine as it may seem, however, this stage is vital to a successful engagement and is where excellent penetration testers distinguish themselves from very good ones. True professionals provide technically impressive findings *and* make recommendations that are closely aligned with business goals. Businesses make money by consuming risk wisely and it is irresponsible for a test professional to suggest that a business eliminate all risk regardless of the cost. As trite as it sounds, there is no such thing as 100% security, and not everything is worth protecting. Clients greatly appreciate technical analyses that are presented in relevant context and merged with management consulting wisdom. Excellent penetration testers will prioritize the discovered vulnerabilities based on the ease/likelihood of exploit, difficulty/cost to mitigate and impact to the business if exploited. Very good testers will only prioritize based on the former two factors, mediocre testers will focus solely on the first factor and beginner testers will provide no analysis at all, just a laundry list of vulnerabilities.

4.4 Where

The "where" of penetration testing, for our purposes, does not refer to an inconspicuous room with the shades drawn, lit only by the faint glow of the LCD screen. No, by *where* we mean of course where *in the application* should the penetration tests focus. As more common burglars might attempt a break-in at any of the various doors and windows on a house, digital attacks will focus on any and all interfaces provided by an application. This may include the front door, or user interface, and it may include the back door, which may only be intended for use by administrators or customer service personnel or developers. Rather than jimmying doors with crowbars, or picking locks in a literal sense, it is via input manipulation that the digital attacker plies his/her trade, and it is on input that testers will focus the majority of their time as well. That is to say, that forms will clearly be a focus, as will HTTP headers, cookies and any other input fields accepted, either explicitly or implicitly, by the application.

Without a valid account of some kind, the tester is confined to attacking the external doors and windows. Generally, this includes static pages, which are of no inherent value unless the web server itself can be attacked; generic functionality including help pages and sign-up forms, which have been known to yield under pressure; and the authentication functionality, which will likely bear the brunt of the tester's focus owing to its supreme security responsibilities.

All of the seven elements of security, i.e., authentication, authorization, confidentiality, integrity, non-repudiation, logging, and information disclosure are inter-related. Clearly, violating the integrity of data on a web server, e.g., by deleting or altering a configuration file, may affect its ability to properly manage authentication or authorization. Likewise, failures in an authorization scheme certainly jeopardize the goals of confidentiality, integrity, information disclosure and possibly logging and non-repudiation. But of all the seven elements, authentication is clearly the foundation. Without properly identifying the user, there can be no legitimate authorization scheme, there can be no confidentiality, integrity, logging, etc. In addition to controlling the most critical security-related functionality of an application, the authentication interface is, by its very nature, the most exposed interface of all since it is the interface by which illegitimate users are identified and rejected. Clearly authentication is a worthy target for attackers and it is where businesses ought to spend a significant amount of time testing their design and implementation.

As HTTP is stateless, web applications are constantly suffering from rather severe short-term memory loss, much like the main character in the movie *Memento*. This fact of life enforces significant complexity on the application and complexity is the fundamental enemy of security. As each and every request to the web server arrives void of any and all inherent connection to previous requests, the web server and/or application are forced to devise an artificial scheme for remember who is who and where they have been. This essentially amounts to constantly authenticating the user which is, of course, precisely what basic and digest authentication represent. As authentication is the foundation of all other security measures, the penetration tester has ample opportunity to "break" security due to this structural requirement for incessant session management. Guessing or acquiring by other means the unique identifiers of another user amounts to *becoming* that user in the eyes of the application, and so session management attacks and session theft will represent a significant focus of the penetration process.

The third and final area of automatic focus for the penetration tester is the management interface or the administrative account. Since the management interface, or "admin" account, explicitly provides heightened privileges relative to the other interfaces and accounts, its conquest reaps greater pay-offs for the attacker and warrants particular attention for the test professional. Additionally, while the application developers may wish to incorporate account lock-out policies into their general authentication scheme, it is difficult to justify locking out the administrator if, to wit, that administrator primarily plies his/her trade remotely, as is implied by the existence of a separate interface or remotely accessible administrative account. Therefore, it is frequently the case that the administrative account can be brute forced. Where there is lock-out functionality on the administrative account, it can easily be locked out, which amounts to a Denial of Service (DoS) attack that may gain the penetration tester or attacker some time to pry open another door somewhere else in the application.

4.5 When

The question of when to penetration test applications seems to be moot, but only because the process is still maturing. Generally, today's applications are tested for penetration after QA testing, either immediately before deployment, immediately following deployment, or both. This makes a great deal of sense, since it is the production application whose attack we are worried about, so it isn't surprising that this is the near universal norm. The @stake Hoover Project (http://www.sbq.com/sbq/rosi/-index.html), which pioneered the concept of Return On Security Investment (ROSI), illustrates precisely why an over-reliance on this approach is inefficient and contra-logical.

During the Hoover project, @stake studied the security vulnerabilities of applications encountered during our client application assessments. Using @stake's engagement data over an eighteen-month period, we created anonymized security profiles for forty-five (45) e-business applications and their potential risk to our clients' business. Among other conclusions, we determined that errors due to weak security design were responsible for 70% of the all vulnerabilities we identified. While implementation errors, including off-by-one errors or improperly called routines, are generally easy to repair, design errors frequently require major overhauls of the application, and can take weeks and months to implement sincerely and effectively. Most companies continue to treat security as a "penetrate and patch" activity typically performed after an application is deployed, rather than integrating secure software engineering practices into the entire development lifecycle. This approach is financially wasteful, however, as it means that significant errors can be integrated into the *structure* of the application, such that their mitigation requires significant redesign and redevelopment. The direct losses in time spent on redevelopment are compounded by the opportunity costs of the developers' inavailability. Conceptually, the "penetrate and patch" approach is akin to first constructing a skyscraper and then testing it for stability. Security, like stability, must be *built in*, it cannot be painted on or retrofitted.

Certainly we are not suggesting that developers forego penetration testing of the final application. This would be both irresponsible and foolish. To garner the highest ROSI, however, it is critical to design applications securely from the beginning of the development process. This means penetration testing during all phases of the application lifecycle and design elements that facilitate clean testing. The application's problem statement itself should be critically examined in a cost/benefit framework that incorporates digital security as a component. The requirements specification must be assessed for the inclusion of appropriate security requirements to ensure that technical and business needs are well synchronized. Most importantly though, the application design must be thoroughly "penetration tested" for vulnerabilities. Proposed designs must be deconstructed and examined for adherence to the essential maxims of security, including segmentation, structural security, the principle of least privilege and attention to input validation. Critical process boundaries and privilege boundaries must be examined carefully to ensure that the design incorporates sufficient tactics for ensuring appropriate authentication, authorization, confidentiality, integrity, non-repudiation, and logging, and that proposed solutions are conscious of information leakage and corollary risks.

Appropriate attention to security during the design phase will significantly hamper the penetration tester's ability to "fail to fail" by dramatically improving overall application security. Problems identified during testing will traditionally be easier and less expensive to fix, and are less likely to represent critical vulnerabilities. While rushed and overworked developers will be naturally reluctant at first to add steps early in the development lifecycle, such an approach will actually tend to abbreviate the development cycle as a whole, initially by reducing redevelopment requirements, but also because "fat" design phases produce more detailed documentation and more thoughtful designs that facilitate the implementation phase. Transitioning to this process requires developer training and should incorporate outsourced Application Architecture Assessments with a focus on knowledge transfer. As post-development penetration testing becomes more of a commodity, the artists will begin to ply their trade earlier in the development lifecycle, and cutting edge businesses will covet their work.

5 Coda

This duet, the art critic staring at the tide from the mountaintop and the artist looking at the mountain while standing in the surf, circumscribes penetration testing as it is and will become – one part position and one part momentum. As with any real challenge, knowing what we know and knowing what we don't know are of near equal value.

We suggest that the takeaways, the thoughts to infect others with, are these:

- Nobody likes surprises.
- Do not wait until your application goes live — integrate security into your application from the get-go.
- Assess security during design; before, during, and after development; and prior to testing and deployment.
- Security consulting firms can deliver focused expertise to quickly identify your security needs, and create a roadmap for implementing solutions.
- More importantly, security consulting firms are the only way you have to know how you compare to others in your field as only a consulting firm can combine trust-based data acquisition with identity-protecting pooling of that otherwise unobtainable comparability data.