Testing Production Systems Safely: Common Precautions in Penetration Testing

Sven Türpe
Fraunhofer Institute for
Secure Information Technology (SIT)
Darmstadt, Germany
e-mail: sven.tuerpe@sit.fraunhofer.de

Jörn Eichler
Fraunhofer Institute for
Secure Information Technology (SIT)
Darmstadt, Germany
e-mail: joern.eichler@sit.fraunhofer.de

***Abstract*—Unlike testing in a laboratory or test bed situation, the testing of production systems requires precautions to avoid side effects that might damage or disturb the system, its environment, or its users. This paper outlines safety precautions to be taken when testing production systems. Specifically we discuss precautions for penetration testing aiming at identifying security vulnerabilities. We generalize and document experience we gained as penetration testers, describing how the risks of testing can be mitigated through selection of test cases and techniques, partial isolation of subsystems and organizational measures. Though some of the precautions are specific to security testing, our experience might be helpful to anyone testing production systems.**

***Keywords*-security test; penetration test; risk mitigation**

I. Introduction

Software testing is a well established and analyzed discipline of software engineering. Testing is *"the process of executing a program with the intent of finding errors"* [1]. Testing activities are part of every phase of the software project life cycle and apply a multiplicity of techniques. It is common practice to separate development, test, and production artifacts as well as environments. Flawed separation of these environments has been reported on several occasions as a cause of software defects [2].

But separation of test and production environments is not always possible. Some test objects are hard to isolate from production environments or the objectives of testing prohibit effective isolation of the test object. A known but not often analyzed example is penetration testing. Another example is testing within orchestrated service environments, where only production instances may be available for some of the services. Enterprise systems as a third example are often highly interconnected with their environments, making them hard to replicate. The number of hard-to-isolate test objects is increasing since *"applications are federating"* [3].

This paper discusses the risks associated with testing in production environments and the mitigation of these risks. Penetration testing is used as an example.

The remainder of the paper is organized as follows: The next section briefly introduces penetration testing to provide some background. Section III discusses the risks involved in testing production systems and of penetration testing in particular. Precautions to mitigate these risks are described in sections IV through VI. This discussion is split into three topics: low-risk test cases and techniques (section IV), semi-isolation (section V), and organizational precautions (section VI). The paper closes with a brief conclusion.

Related literature is referenced where appropriate but not specifically discussed. The authors are not aware of literature touching the immediate subject of this paper—precautions for testing in production environments—and this apparent lack of documented best practice is our primary motivation for writing this paper.

II. Penetration Testing

A. Definition

The concept of *penetration testing* was introduced by Farmer and Venema describing their tool *SATAN* [4]. They proposed to analyze computer networks from the perspective of an attacker. Highly disputed in the beginning, penetration testing is now an established procedure in information security [5], [6].

A common definition of penetration testing states that *"a penetration test is the controlled attempt at penetrating a computer system or network from 'outside' in order to detect vulnerabilities. It employs the same or similar techniques to those used in a genuine attack."* [7] A vulnerability in the sense of this definition is a design flaw, bug or misconfiguration that could result in a breach of security policies, e.g. an unintended information disclosure.

This definition can be broadened by including applications into the test scope and by omitting the *outside* attribute [8], [9]. The justification is that significant portions of reported security incidents originate inside an organization, and many vulnerabilities reside in applications rather than the underlying infrastructure. This leads to the statement that an penetration test at its very center aims at an *"illegitimate acquisition of legitimate authorization"* [3].

B. Objectives

Objectives of a penetration test vary and have to be agreed upon by the stakeholders of the test. Objectives can be categorized using the following general aims:

- Identifying vulnerabilities

- Improving the security of technical systems as well as of the organizational and personnel infrastructure
- Confirming the IT security by an independent third party

Penetration tests can be executed at nearly all stages of the software project life cycle but they are one of the rare examples of testing methods that are commonly applied to production environments or the operational stage [6]. Tests on systems in the operations stage check, *"whether a system is operated according to its current security requirements. This includes both the actions of people who operate or use the system and the functioning of technical controls"* [10]. In many cases penetration testing is required by legal and organizational regulation and thus mandatory.

Specific to penetration testing—or perhaps to most parts of the security testing discipline—is that only a rough test plan can be outlined upfront. Security testing is largely exploratory testing, picking up trails of potential vulnerabilities and following them until exploitability and impact can be assessed. Observations made in the course of the testing thus lead to new test cases incrementally. This property is rooted in the difference of testing expected behavior (does an application work as specified?) vs. unexpected behavior (does an application do other things as well?) [3].

More feasible than detailed planning of tests is often the planning of restrictions, situations or time frames during which certain tests, or tests with certain risks, are bared. Restrictions are typically driven by operational requirements such as availability of a system during business hours, and by operational capabilities such as availability of personnel to handle incidents. Feasible is also continuous short-term planning, e.g. day-to-day planning and coordination of predictable activities. In cases of doubt tests can be postponed until the next scheduled planning update without causing too much delay.

C. Methods

The methods and tools used for penetration testing have to be aligned with the objectives. Several attributes have been proposed to classify penetration tests [7].

Information base: the amount of information about the test object available to the testers (minimal: black-box vs. maximal: white-box)

Aggressiveness: the impact on the test object that the tester accepts (passive, cautious, calculated, aggressive)

Scope: the range of the penetration test with regard to the test object (full, limited, focused)

Approach: the information level of stakeholders (e.g. system administrators) within the organization concerning the execution of the penetration test (covert, overt)

Technique: the access vectors to the test object (network-based, other communication, physical access, social engineering)

Starting point: the initial position of the tester with respect to the organizational boundaries (outside, inside)

Most test runs combine several of the variants outlined above. For instance one commonly combines automated passive scanning to get an overview with more focused and more aggressive attacks of limited scope [11].

III. Risks

A. Risk Factors

Principal risk factors are the security testing itself, the system under test (test target), the embedding of the test target in its environment, and the operational data residing on the target. In most testing situations one would mitigate the latter three by setting up a dedicated test system in a lab environment using test data, rendering the risk caused by the testing itself acceptable. This is obviously not possible when a production system is to be tested.

Security testing causes risks to the target by its very nature. Like an attacker the penetration tester deliberately leaves the relatively safe grounds of intended use and expected activity. Security testing is inherently invasive where it employs techniques similar to those used in an attack. Consider code injection for example, where one attempts to manipulate a system into executing arbitrary code provided by an unauthorized party.

The *target system* poses a risk as its operation has implications in the real world. People and processes rely on it and the data the system handles or produces. Processing within the system has a meaning outside and may trigger or affect actions in the real world. From a more technical point of view the test target is *embedded* in an environment comprising other systems or services, personnel, and users. Manipulations and disruptions may therefore have an impact elsewhere even if no processing with real-world implications occurs. Finally, the *data* stored on or processed by the target system are often sensitive in their own right, requiring that their integrity, confidentiality and availability has to be maintained. See subsection III-C below for an example.

B. Specific Risks

Specific risks of penetration testing can be technical, organizational, and legal.

1) Technical Risks: These are caused directly by the testing or by the system being tested. Technical risks include:

- Failure of the target or connected systems
- Disruption of service or reduced performance
- Loss, modification or contamination of data
- Disclosure of data to the testers or to 3rd parties
- Spill-over of data into connected systems
- Initiation of irreversible real-world actions
- Triggering of automated responses and countermeasures
- Loss of defenses against real attacks

2) Organizational Risks: Possible side effects of testing are:

- Unnecessary triggering of incident handling processes; testing these may, however, also be an objective of the test
- Reduced attention to real incidents during and after the test
- Disruption of business processes and functions
- Loss of reputation if 3rd parties are affected

3) Legal Risks: Legal obligations and possible side effects to 3rd parties cause further risks:

- Violation of legal obligations
- Inadvertently committing punishable acts

C. Example

To get a better idea what these risks might mean in practice, consider the following example of a system that might be subject to penetration testing. Imagine a typical online shop, a Web site offering merchandise for sale. Users of this Web site select items from the catalogue, order them by proceeding to checkout, and receive the ordered items by mail after having made their payment. On the part of the shop, most of the process is automated. The only human intervention normally occurs only in a warehouse where the ordered items are retrieved from the shelves and put into a box for shipping.

This shop is interwoven with its environment in various ways. On the technical level the Web site may be hosted on a shared server along with other Web sites of the same or of other companies. Testing may thus affect unrelated Web sites if for instance a test slows down or crashes the entire server, or if the testers gain access to data that belong to one of the other Web sites.

When an order is being processed, real merchandise will end up in a real box, which will be shipped to the specified address. Unless the testers find a way around payment, a real amount of money will be charged to a real credit card or other means of payment. A number of e-mail messages may be sent out in the course of order processing, and e-mail addresses may be reused later for advertising purposes.

Careless testing in this situation could set off fraud detection alarms of credit card companies, cause merchandise to be shipped to arbitrary address, or annoy innocent bystanders with unsolicited e-mail messages. In addition, if the testers are successful in their attempted attacks, they might gain access to sensitive data of customers.

IV. Low-risk Test Cases and Techniques

Security testing has some peculiar properties. There is often a wide range of different test cases for the same vulnerability, and testing is biased towards false-positive results. The first property follows from the fact that most security vulnerabilities can be exploited in various ways to different ends, some being more invasive than others. The second is related: the expected result of a security test is neither pass nor fail but rather a set of vulnerabilities that an attacker might be able to exploit. This result may include potential vulnerabilities, defects whose exploitability is likely but has not been demonstrated. A further peculiarity is the reversal of perspectives. Just like an attacker, the penetration tester is focused on undesired functionality of a system. From this viewpoint the regular and intended functions imply side effects to the attack or, in our case, the testing.

These properties often allow the tester to prefer less risky test cases over more critical ones. Common strategies and techniques are:

1) Indirect Testing: Instead of testing for the actual defect the tester or a testing tool collects just enough evidence to conclude that a vulnerability is likely to be present. This technique is particularly useful when testing for known vulnerabilities in software packages. The simplest approach is to enumerate the software packages present in a system or accessible through an interface, determine their versions and patch levels, and look up known defects in a database. Results are not verified so they might contain false warnings. Vulnerability scanners such as Nessus [12] support this technique.

2) Limited Exploitation: The tester prefers test cases that demonstrate the vulnerability and how it can be exploited, but limits actual exploitation to effects that are presumably harmless. Read access to data is preferred over writing and writing new data is preferred over manipulating existing data. If a vulnerability allows code to be executed on the target system, the tester attempts to use a payload that shows a visible or measurable effect without undesirable side effects. In many cases it is possible to conclude from such testing what else an attacker could achieve.

3) Delayed Effects: Sometimes it is possible to design tests for delayed effects. The tester then determines the test result inside the system and cancels or inhibits any further processing before it would occur. This strategy, if applicable, is helpful where tests have real-world effects, e.g. transactions being initiated while testing an Internet shop or banking system. If a vulnerability is found this way, the tester stops and concludes that undelayed operations might be vulnerable, too.

4) Interruptible Testing: Trivial as it may sound, testers have to ensure they can interrupt their testing at any time so they can react quickly if unintended consequences are observed or reported. The use of self-replicating programs—worms and viruses—is commonly considered off-limits. Interruptible testing may be difficult if the system under test triggers processes elsewhere where the tester has no control over their execution.

5) Throttled Tools: When using automated tools to execute a large number of individual tests, the tester has to ensure the target won't be overloaded, resulting in disruption

of service. Examples of such tools are vulnerability scanners, port scanners, or brute force password guessing tools. These tools can easily exhaust resources of the target system by sending a large number of requests to the target system in short time. Dedicated security testing tools often support throttling [12]. Deliberate attempts to cause denial of service are commonly considered off-limits.

6) Avoid Lock-out: Besides overloading the target, repeated tests might also trigger functions designed to lock out attackers. Password-protected systems for instance often limit the number of failed login attempts. After the limit has been reached the account will be locked and an administrator has to unlock it. Testers must ensure they do not cause denial of service by triggering such functions or limit their testing to a domain where lock-out can be handled.

7) Direct Predictable Effects towards the Tester: If the testing has predictable effects outside the immediate target system, test cases should be designed to direct these effects towards a system or domain controlled by the tester. If, for instance, the test target is a banking system and the tester attempts an unauthorized transfer of money, both the source and the destination account used should be test accounts rather than arbitrary real ones.

8) Use Reserved Addresses, Avoid Broadcast: Wherever possible the tester should use reserved addresses as test input to avoid affecting other systems or users. For the Internet for instance, RFC 2606 [13] specifies top-level domains reserved for testing and documentation. They can be used to construct addresses that are guaranteed to be invalid. Furthermore the tester should avoid using unnecessarily extensive addressing, such as broadcast addresses for a local network. Using reserved addresses can fail if the system considers them invalid and refuses to accept them as inputs.

V. Semi-Isolation

Although penetration testing targets systems that are not easily replicated in a test bed or isolated from their environment, some degree of isolation may be feasible to achieve.

1) Testing Tests: The exploratory part of penetration testing—developing new tests based on vulnerability hypotheses—is inherently more risky than the mere execution of well-planned tests. To the extent feasible, testers should attempt to use a lab environment to develop and try tests before running them against the real target.

2) Establish Testing Domains within the Target: Many systems comprise parameterized compartments that are separated from each other to some degree but identical in their software inventory and most of their configuration and environment. Examples are separate user accounts within the same system and individual worker hosts in a load-balancing environment. Limiting testing to a single one or to a small number of representative compartments can limit some of the effects and side effects of testing to this compartment, too. Since the compartments are mostly identical, test results are likely valid for all of them.

3) Partial Isolation and Replication: Subsystems can sometimes be reconfigured for testing, either dynamically or by setting up a replica of the subsystem in a different configuration. An application that uses a database for example could be replicated on top of the same database, or it may be possible to use the same application instance with either a production or a test database. This way, some of the side effects of testing can be limited to a less critical domain.

4) Confining Tools: Penetration tests usually involve interacting with the target over a network. To avoid accidental execution of risky tests against other systems than the designated target, it is recommendable to enforce the scope of the test on the network level. Dedicated tools such as Nessus [12] may support this internally; otherwise a firewall will do the job.

VI. Organizational Precautions

1) Identifiable Test Input: The precautions discussed so far aim at preventing adverse effects of testing. But even if testers adhere to these precautions the testing will leave traces within the target. Penetration testers therefore need to choose test inputs that are easy to identify in a later clean-up, and they must sufficiently document their testing. Choosing distinct test patterns may be difficult depending on the data types involved. Identifiable test inputs also help avoid overreaction if the testing is detected by operations personnel and interpreted as an attack.

2) Rules of Engagement: Most penetration tests are carried out by contractors. The client and the contractor must establish clear, unequivocal rules of engagement. The target and the limits of the test must be specified, and the client must give explicit permission for testing within the limits. 3rd parties that might be affected by a test must be notified.

3) Establish Communication Channels: Before a penetration test starts, communication channels must be established between the parties involved. Whoever notices a problem in the course of the test needs to know whom to contact. To ensure availability of the contact persons, time slots should be arranged for any testing activity that interacts with the test target. Existing incident handling and emergency plans must be taken into account; decision makers specified in these plans must be aware of the ongoing test.

4) Awareness and Education: Testers need to be aware of the risks and their specific rules of engagement. They must be educated about the mitigation strategies available and their application in practice. This is an additional required qualification on top of what the testers need to know about testing as such. Besides understanding how to make a system fail, testers of production systems also need to consider the impact and side effects of their testing on the target and its environment.

A simple yet helpful tool is defining risk classes and requiring testers to assess the possible impact and side effects of any test beforehand in terms of these classes. A single, three-step discrete scale is often sufficient. Risk classes can be used when communicating with stakeholders and defining the rules of engagement. For instance one might require that high-risk tests be carried out only at certain times or after notifying certain stakeholders.

5) Documentation: Testing production systems requires additional documentation. Besides reproducing tests, the documentation must also facilitate analyzing any unintended adverse effects that the testing may have had, for later cleanup and recovery. Preferably all interaction with the target should be logged. This may, however, produce a large amount of data, particularly if automated tools are used. At the very least testers should document readily available information such as the start end end times of test runs and the addresses of their own systems that they use for testing.

VII. Conclusion

We have argued that testing within production environments is sometimes necessary and desirable. Conventional wisdom is to separate environments for the different life cycle stages and to allow testing only within development or test environments. In some cases this has to be complemented with adequate considerations concerning tests within production environments. Penetration testing is one example, for which we have described the risks involved and the precautions that testers should take to mitigate these risks based on our experiences. Following the documented precautions allows testers to better control the risks associated with testing in production environments.

One immediate conclusion can be drawn from this outline of best practice with regard to penetration testing itself: penetration testing should not be confused with simulated hacking. While similar techniques are applied, penetration testing implies the secondary objective of not damaging the target. The practice of treating the test target as a black box, which some penetration testers propose for the sake of 'realism', therefore seems questionable. Penetration testers need to think like an attacker but they should not be in the same situation regarding prior knowledge of the target.

The type and scope of our paper prevent us from delving into a discussion of the possible effects of precautions on test results. Taking precautions restricts the set of test cases available, which may affect the results. It seems worthwhile to further explore the conflicts between the primary objectives of security testing and safety as a secondary objective in production systems. Likely affected and constrained by safety precautions are:

- Tests aiming for denial of service (DoS) conditions as their primary effect
- Tests with a high risk of DoS as a side effect, e.g. testing for buffer overflow or format string vulnerabilities
- Testing techniques with inherently unpredictable results, such as fuzzing (random input patterns)
- More generally, any automated technique that produces a large number of test inputs
- Any testing for flaws in the core business logic of a system.

Considering such possible limitations we have to conclude that testing in the production phase of a system's lifecycle can complement but not replace testing in earlier phases.

Acknowledgement

This work was supported by CASED (www.cased.de).

References

[1] J. G. Myers, *The art of software testing*. Wiley, 2004.

[2] E. Dustin, *Effective software testing: 50 specific ways to improve your testing*. Addison-Wesley Professional, 2002.

[3] D. Geer and J. Harthorne, "Penetration testing: a duet," in *Proc. of 18th Computer Security Applications Conference, 2002*, 2002, pp. 185–195.

[4] D. Farmer and W. Venema, "Improving the security of your site by breaking into it," *Available from ftp.win.tue.nl*, 1993.

[5] C. Weissman, "Penetration testing," in *Information security: an integrated collection of essays*, M. Abrams, S. Jajodia, and H. Podell, Eds. IEEE Computer Society Press Los Alamitos, CA, USA, 1995, pp. 269–296.

[6] J. Wack, M. Tracy, and M. Souppaya, "Guideline on network security testing," NIST SP800-42, 2003.

[7] BSI, "A penetration testing model," http://www.bsi.bund.de/english/publications/studies/penetration.pdf, 2003.

[8] F. C. Freiling and J. Liebchen, "Iterative Kompromittierungsgraphverfeinerung als methodische Grundlage für Netzwerkpenetrationstests," in *Proceedings SICHERHEIT 2008, Saarbrücken, 2008-04-02*, 2008.

[9] M. Keeney, E. Kowalski, D. Cappelli, A. Moore, T. Shimeall, and S. Rogers, "Insider threat study: Computer system sabotage in critical infrastructure sectors," 2005. [Online]. Available: http://www.secretservice.gov/ntac/its_report_050516_es.pdf

[10] M. Wanson, "Security self-assessment guide for information technology system," NIST SP800-26, 2001.

[11] P. Herzog, "OSSTMM 2.2: Open source security testing methodology manual," Open source document, http://www.isecom.org/osstmm/, 2006.

[12] N. Dhanjani and J. Clarke, *Network Security Tools*. O'Reilly, 2005.

[13] D. Eastlake and A. Panitz, "RFC 2606: Reserved top level DNS names," http://www.ietf.org/rfc/rfc2606.txt, 1999.