28TH ANNUAL
PACIFIC NW
SOFTWARE
QUALITY
CONFERENCE

OCTOBER 18TH – 19TH, 2010

ACHIEVING QUALITY IN A COMPLEX ENVIRONMENT

Conference Paper Excerpt from the

CONFERENCE PROCEEDINGS

Adaptive Application Security Testing Model (AASTM)

Ashish Khandelwal (Ashish_Khandelwal@McAfee.com**)**
Gunankar Tyagi (Gunankar_Tyagi@McAfee.com**)**

Abstract

Security testing is synonymous with terminologies such as Cross Site Scripting (XSS), SQL Injection, Key logging, backdoors, phishing attacks, and so on. However, the lack of skill & experience in Application Security Testing prevents practical implementation of security testing, and most project teams do not know where or how to start.

This paper follows the experimental ride of our team of functional testers and the new approach towards Application Security Testing. As a team, we faced many challenges from the beginning in understanding Threat Modeling, struggled to implement the same in a time-constrained manner, and had to deal with the failure to finally implement and yield value. We subsequently created a self-adjusting model, which we have coined as "***Adaptive Application Security Testing Model***".

The Adaptive model aims to maximize the efforts an Application Security tester puts in. With the hard deadlines and resource optimization that are commonplace in the industry, this paper gives you a strategy to achieve Application security without compromises.

Here are some of the strategic points, which an Application security tester can leverage from our paper:

a) How to position themselves on our Adaptive Ladder
b) Understand Application Security from a rudimentary yet adaptive perspective
c) How to create Attack Models
d) Review real case studies

Biography

Ashish Khandelwal *has more than 5.5 years of Software Testing experience. He holds a B.Tech degree from IIT Kanpur and works as a Senior QA Engineer with McAfee Enterprise product solution group. Being a Certified Ethical Hacker from EC-Council, he is interested in the latest Security Testing trends, and continuously seeks to improve software security. Ashish works towards becoming a technology solution consultant/ architect by providing technical insight to different process verticals in testing solutions.*

Gunankar Tyagi, *a Computer Science Engineer has around 2 years of experience solely in Black Box testing. Being CompTIA Security Plus certified, he has earned many accolades for his out-of-the-box testing skills and has proven himself a respected tester in a very short time. His areas of interest delve more into Security Testing.*

1. Introduction

Have you ever thought how easy it would be to write simple test cases that can reveal the security loopholes in your product? However as many of you may have experienced, this is easier said than done. We ourselves have witnessed quite a few of the security testing initiatives abandoned midway. We usually feel the need to justify it in terms of why we need it, do we need it at all and what will we achieve from it? After all, this is the herculean effort of bringing the security-testing framework alive. Even though the proven "Threat Modeling approach" remains the best tool available, it seems to be too expensive in terms of energy and effort investment

This paper tells our story as a test team, as we took the initiative to conduct security testing on our product. We attempt to describe the various challenges we came across and how we weaved our way around them.

We were not successful in finding security defects in the early part of our testing but soon we found that the very nature of 'Threat model approach' made it difficult and discouraging for beginners like us. The real breakthroughs started happening only when we realized that threat model itself is the hidden barrier and that we would have to reevaluate our strategy to move ahead. This led to a new approach and we have named it Adaptive Application Security Testing Model (AASTM) model.

We would like to begin by answering a few basic (frequently asked) questions to explain the need of security testing in general. Then we would like to share a few challenges faced by us in implementing application security testing. After this, we will elaborate the two-tier AASTM approach and its security testing types with the help of case studies. We conclude the paper with an Application security-testing checklist and a few lessons we learned along the way.

2. Application Security Testing – Primary Needs

In this section, we try to answer some of the basic questions, which come into the mind of security tester when he embarks with breaking the product.

Why to do security testing?

Companies discredit the need for Security testing because they feel there is no return on investment and thus most of the time the need of security testing is not taken seriously. Management does not always know about product flaws; company director assumes that every function works smoothly without any defects. However, experience shows that no product/ system can be deemed completely secure without controversy. There will always be bugs in a program; whether they are found or not is another question.

The losses associated with security flaws have been heavy as can be analyzed from the following security incidents:

TJX Company Breach

The TJX Company breach, which was first reported in January of 2007, has been widely recognized as the largest reported theft of personal details ever lost by a company.

Monster Job Site Hacked

Monster.com suffered a heavy security breach in Aug 2007 that reportedly resulted in the theft of the confidential information for some 1.3 million job seekers. Hackers stole information from the US online recruitment site's password-protected CV library by using credentials taken from Monster clients. They launched the attack using two servers at a web-hosting company in the Ukraine, combined with a botnet.

Operation Aurora

In mid-2009, the most ultra-sophisticated attack was uncovered for which McAfee later coined the term Operation Aurora. Reportedly, Chinese hackers have used a previously undiscovered security hole, existent in several major versions of Internet Explorer. This attack affected as many as 2,411 companies and compromised data ranges from intellectual property, classified documents to credit card transaction details. Such Advanced Persistent Threats are expected to grow in the future

Microsoft and the Love Bug

The love bug, also known as the "I LOVE YOU" virus was made possible because the Microsoft Outlook e-mail client was (badly) designed to execute programs that were mailed from possibly untrusted sources. Apparently, nobody on the software team at Microsoft thought through what a virus could do using the built-in scripting features. The damage resulting from the "I LOVE YOU" virus was reported to be in the billions of dollars.

Today, the hackers have organized themselves and hit the market at a time to cause maximum loss. To make matters worse, the tools available to the attackers are becoming more sophisticated and easier to use. Clearly, a product with vulnerable loopholes has a high probability to bring bad name to the company and thus a prime need to perform security testing.

How to do security testing?

This is where most of the initiatives are killed off, since no one has a clear picture of how to do it, and thus, the whole procedure gets undefined and unclear. The most general approach is to follow Threat Modeling. Nevertheless, is this the right way for you? Could some other strategy help you perform security testing effectively? We will address some of these questions as the paper follows.

Who will do security testing?

Certainly, there are no pre-requisites for a tester who want to perform security testing. The right amount of attitude with zeal to learn could be the starting point for a security tester.

We would like to introduce you to our product environment. We are a team of 15 QA Engineers and 20 Developers distributed across multiple location. The skill-sets and experience levels varies across the team, ranging from Software Engineers working in Quality to QA Analysts with deep domain knowledge. As an initiative, we (a team of 2 functional testers) took up the challenge to find security loopholes in our product. The next section describes the threat model with which we started and gives an idea as to why we failed with this model.

3. Getting Started – Threat Modeling

What Is Threat Modeling?

Threat Modeling is a structured approach to identify, quantify, and address security threats in the application. A threat can be viewed as a potential or actual adverse event that may be malicious (such as a denial-of-service attack) or incidental (such as the failure of a storage device), and that can compromise the assets of an enterprise.

The basis for threat modeling is the process of designing a security specification and then eventually testing that specification during application design.

Threat Model Process

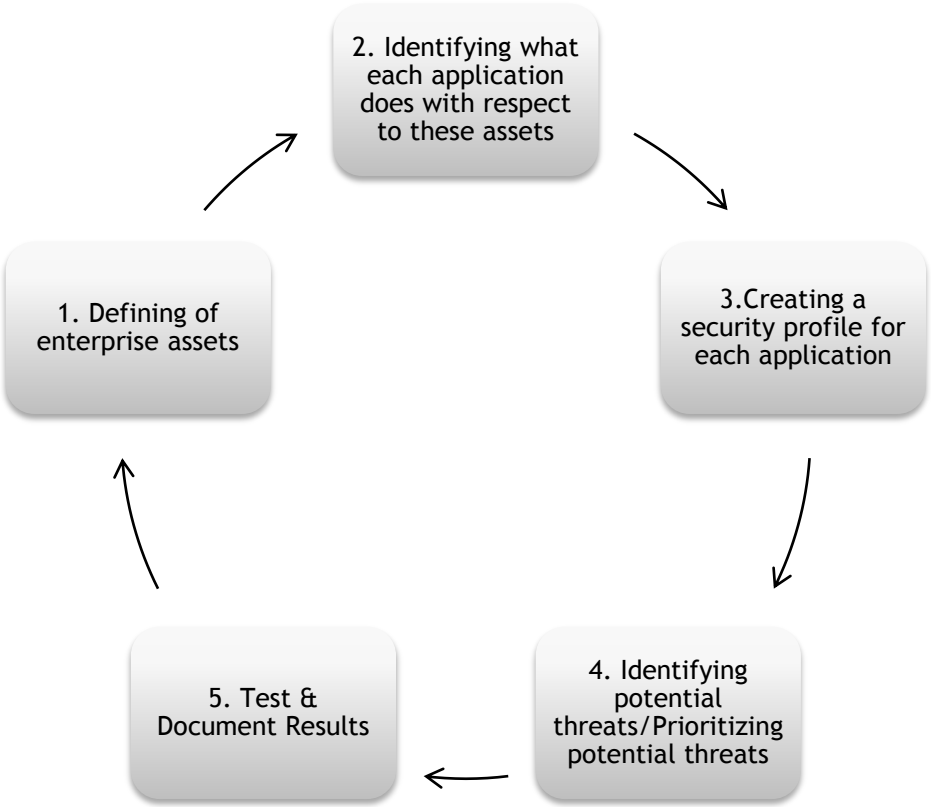

Figure 1 Workflow of a Threat Model

4. Threat Model – Trinity of Constraints

Though the model seems easy to comprehend, it is fairly difficult to implement. As we tried to follow this traditional way, we encountered three constraints as described below:

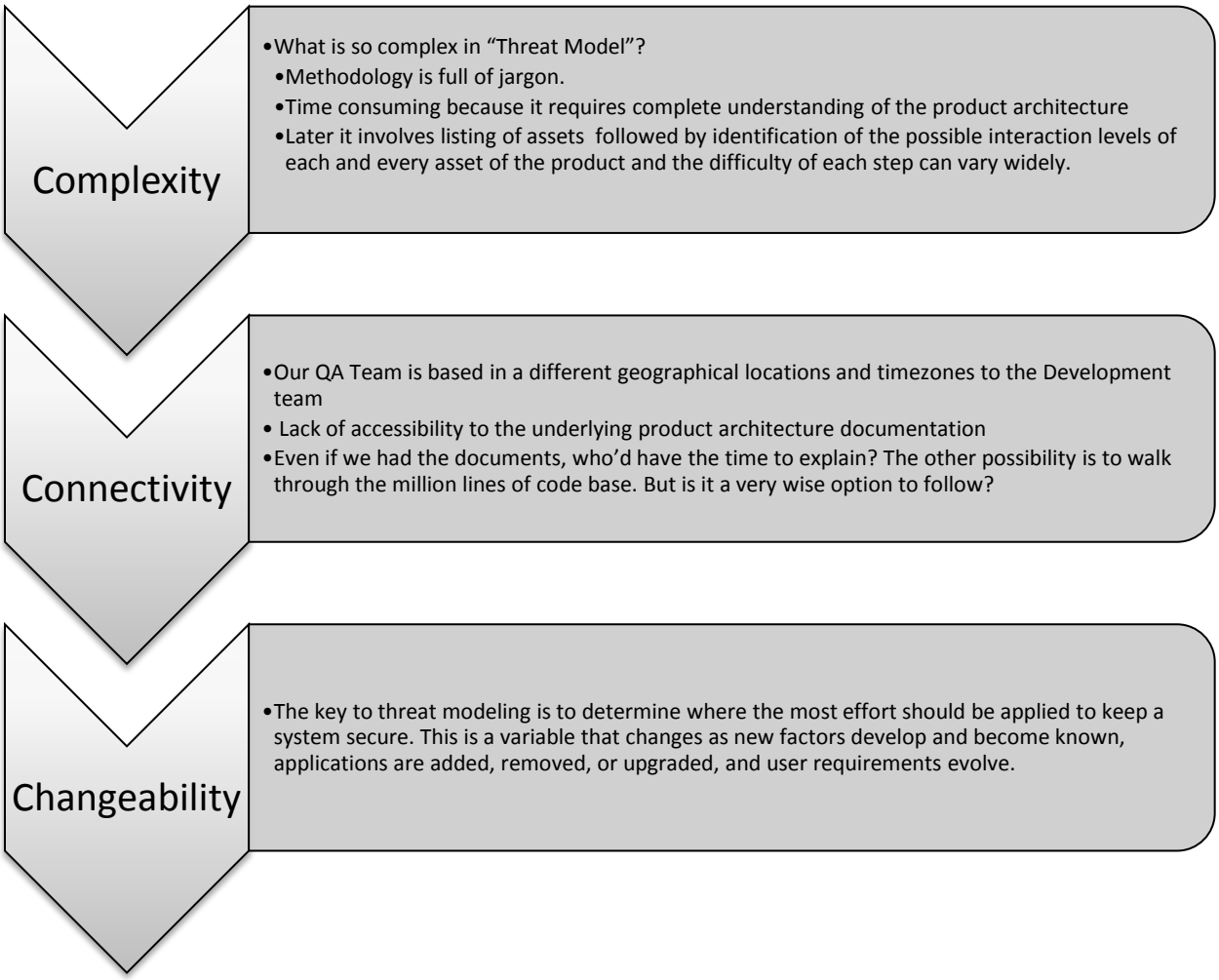

Figure 2 Trinity of Constraints

Finally as we found our motivation sagging we had two choices – either to quit or to find another way. We decided against quitting and decided to step back and reframe our strategy to see if we could come up with something useful. We came up with a strategy that we named AASTM model.

5. AASTM Model – Adaptive Ladder

"Adaptive" approach focuses more on finding security defects while the expertise level is still building. The ladder shown in Figure3 provides the foundation of this approach and compares it with traditional threat model behavior.

As we move up the ladder, the expertise of the security tester increases and gives us an edge by finding early defects that could be functional as well as security. Each step up the ladder optimized our results, but in turn required consistent upgrading of security testing skills and product knowledge.

This ladder also compares the two models in terms of the results achieved as represented by the colored circles (R1, R2, and R3). In Threat Model approach, we would have to go through the entire cycle to uncover the security defects. Although our approach provides the same result at the end of the cycle but with it you can start uncovering the security defects one by one as you step up the ladder. The significance of the security defects found increases as we move up the ladder and this strategy ensures that the tester involved does not lose motivation.

As can be seen from the ladder, it is a two-tier approach broken down into "*Peripheral* & *Adversarial*" described later in this paper. Adaptive ladder gives a foundation to this model that we named as "Adaptive Application Security Testing Model".

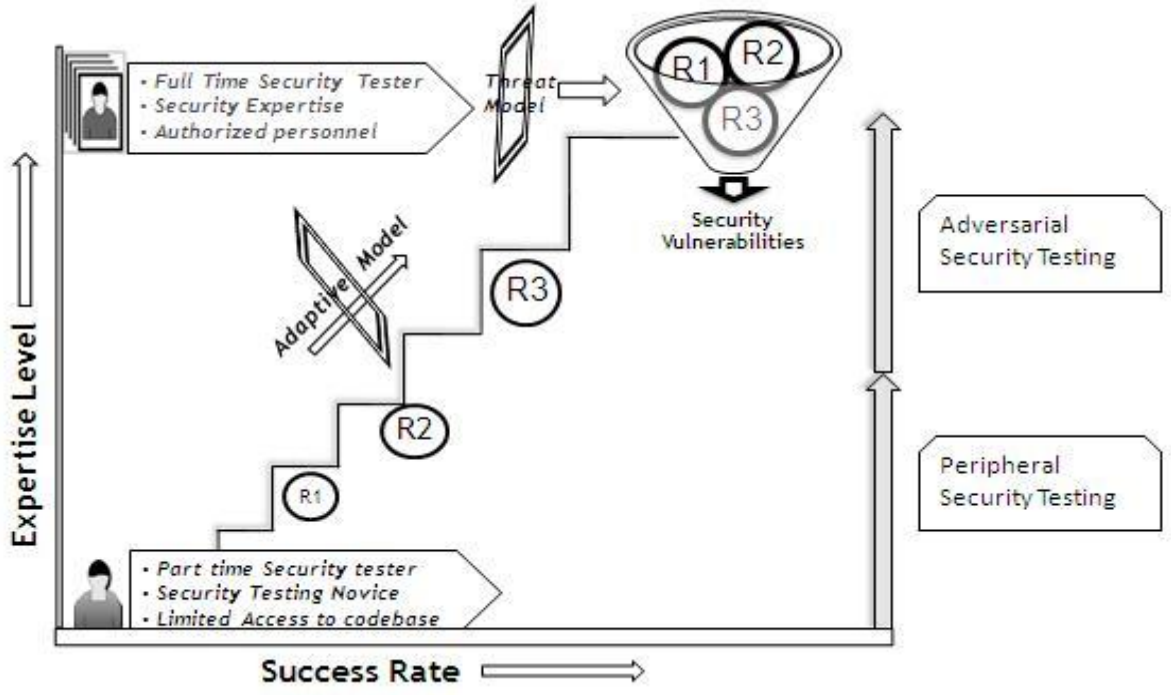

Figure 3 Adaptive Ladder

6. AASTM Model

The “Adaptive” Model in its initial stage is centered more on stimulating the security testing approach by focusing on finding early security defects rather than on studying deeply into technology and product in order to attack it. It increases enthusiasm and adaptability in a complex security arena. As we moved along with this approach, we started enhancing our knowledge and constraints to maximize our potential and results.

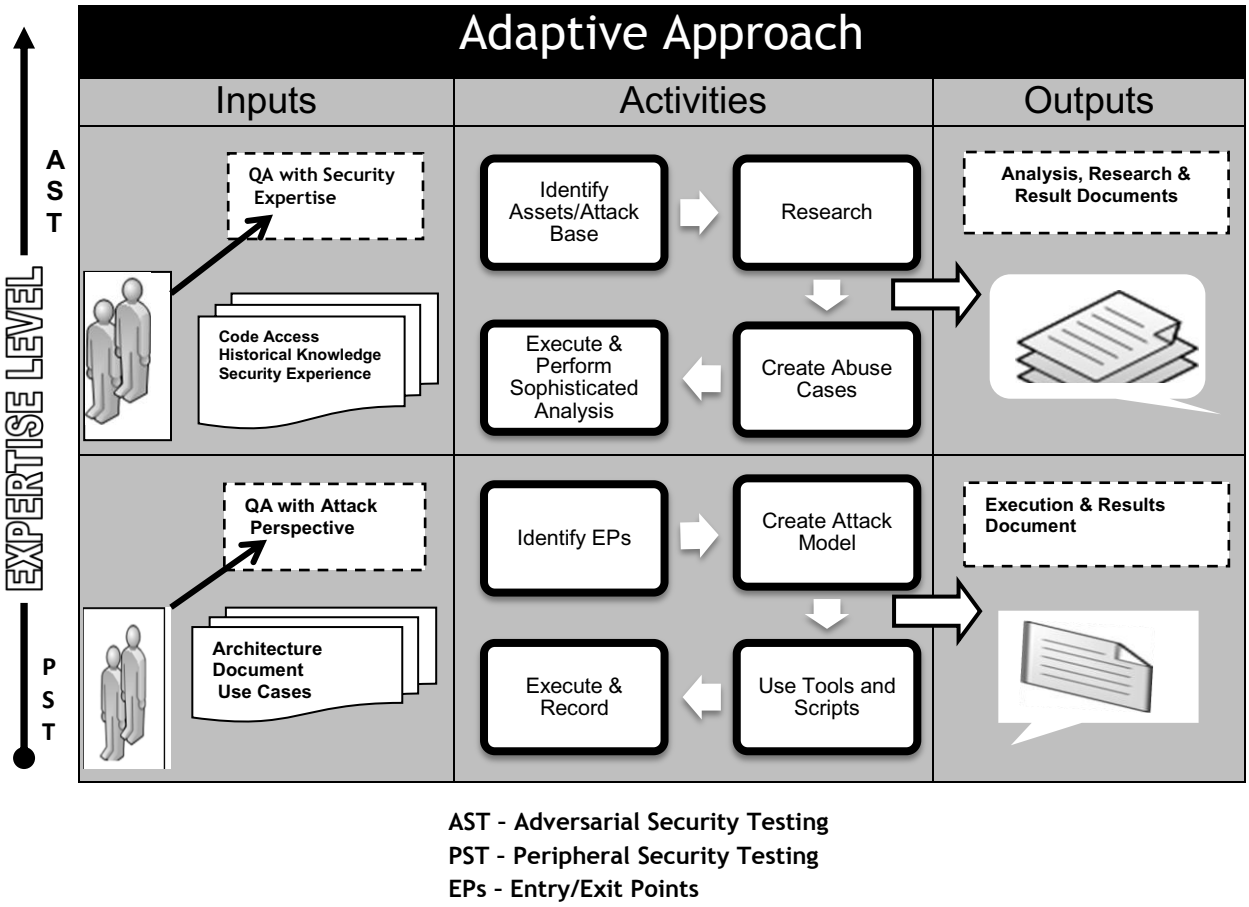

Figure 4 Adaptive Model

As a two-tier testing approach, this model talks about Peripheral and Adversarial Security testing. Adversarial testing is a successor of Peripheral. As we go up the ladder, our methodology changes and hence our priorities to test change. Each testing type is classified in terms of Inputs, Activities and Outputs as seen from *Figure4.*

- **Inputs** are the prerequisites required in order to conduct the particular type of testing.
- **Activities** describe the flow to perform the particular type of testing.
- **Output** takes care of results and analysis.

6.1 Peripheral Security Testing

As a team of functional testers, we were initially following the traditional way of doing security testing, by using Threat Models. However as time passed by with no results, we shifted our focus from traditional method to result based approach. We followed a peripheral approach that helped us in finding the low-hanging fruits together with building security expertise.

This is an entry point based, Black- Box testing approach with the aim of quickly finding surface-level but critical security issues.

Some of the key-definitions involved in this type of testing include:

Entry Point

An entry point is a place where inputs are supplied to your application. E.g. Files/ Folders/ Application UI

Exit Point

Any desirable/ undesirable output from the application. E.g. Log files/ tmp files

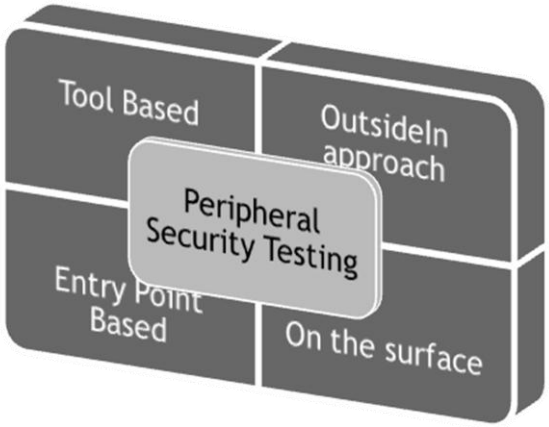

Figure 5 Peripheral Security Testing

Outside-In Approach

This technique breaks the software from outside without knowledge of internal implementation and thus enables testers to find security loopholes the black box way.

On The Surface

The targeted issues are easier to detect and require less effort.

6.1.1 Case Study I (Peripheral Security Testing)

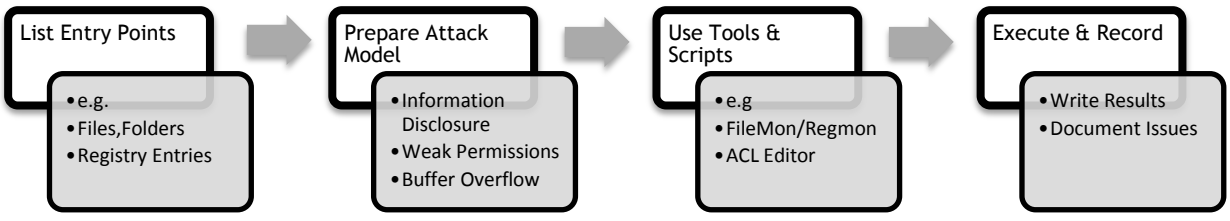

Figure 6 Case Study I (Peripheral Security Testing)

This is a 4-step process as briefed in the figure above. After identification of the entry points, the tester needs to prepare an attack model that is supposed to break the software. Lastly, we need to run the attack scenarios using appropriate tools and perform the result analysis.

List Entry Points

One of the easy ways to find defect is to work upon the most commonly used components of your software. Files, Folders & Registry Entries are notably the most essential and populated components of a product. We aim to explain here the exploitation of these components by following Peripheral approach.

Prepare Attack Model

Here are some preliminary questions, which leads to the preparation of this Attack Model.

Attack Model Development

- ✓ *Does your product functionality hamper if you deny the permissions to the temp folder?*
- ✓ *Do the files (logs/event xml/binaries) contain sensitive data?*
- ✓ *Is there is a way in which you can cause buffer overflow in the file extension /file names?*

This is not the end of the list; it may vary depending upon the tester's thought process and the application knowledge.

Each of the points in Attack Model development leads to one or more attack scenarios. The next section utilizes this attack model and chooses appropriate tools to crack the application under test.

Use Tools and Scripts

SysInternal Tools like RegMon, FileMon and ProcMon are used to monitor real-time File System, Registry, Process or Thread activity. By defining appropriate filters, we can monitor the resources of the product under test. The output of these tools could be analyzed to find information disclosures or Buffer Overflow conditions.

Other built-in Windows Tool called ACL Editor can be used to verify or manipulate the object permissions to detect weak permission flaws.

Few of the times, analysis of the result of one tool gives a way to attack the product using another tool. For instance, once RegMon discovers that our product uses an exploitable Registry Entry, we could find a way to manipulate the permission of this object using ACL Editor.

Execute and Record

The last step is to utilize the tools potential to find product loopholes with the aid of attack models. Side by side, we record our results for the issues found. In addition, the details of the results could be helpful for developing attack models and increasing knowledge base for our product.

Let us assume that our product uses temp folders extensively for doing file system operations. Here, the security issue could be grave if we leave some confidential information inside the temporary folders or if denying permissions to the temp folder makes our product non-responsive. At the same time, we can utilize this knowledge to perform buffer overflow if we repeatedly create tmp files using automation script.

Please refer **Appendix A** for a comprehensive list of various entry points with their attack models and tools list.

6.2 Adversarial Security Testing

As we uncover the surface defects using peripheral approach, it gives us the ability to think maliciously, improvise knowledge base and increase expertise level. After scrutinizing the entry points applicable to our product, we can aim to use our acquired knowledge in order to find the hidden security loopholes.

We achieve this by streamlining our strategy with the Adversarial approach. It is an expertise-based, hostile Security testing approach, which is carried inside out to reveal loopholes in the product. As a successor to the peripheral approach, this enables the security tester to attack the product in holistic way. It takes historical knowledge and security experience as the inputs with the target on the hidden security loopholes.

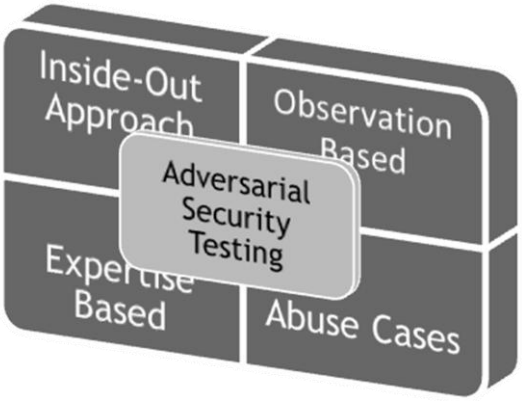

Figure 7 Adversarial Security Testing

Here are few of the key-definitions required in order to understand this approach.

Attack Base

An entity of the product or the Operating System, which can be manipulated to perform an attack on software.

Inside out

This technique requires studying the product internals and eventually uses the data or control flow of a program to break the software from inside.

Historical Knowledge

Gather past vulnerability information about the attack base. Have a quick check in the vulnerability repositories like *cve.mitre.org, securityfocus, osvdb* etc.

Abuse Cases

Abuse cases (sometimes called misuse cases as well) are a tool that can help you begin to think about your software the same way that attackers do.

6.2.1 Case Study II (Adversarial Security Testing)

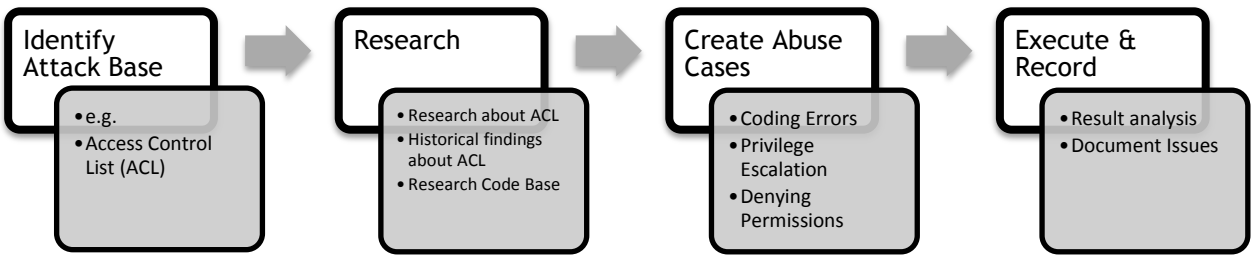

Figure 8 Case Study II (Adversarial Security Testing)

Identify Attack Base

Windows ACL is quite literally your application's last backstop against an attack, with the possible exception of good encryption and key management. This is fundamental part of Microsoft Windows NT and later, however, to some extent one of the least understood feature of Windows. It is not always possible for the developers to map permissions with the resources due to various reasons such as ignorance, design issues or high cost.

Being a fundamental component of Windows, it provided us a good case study by exploring the Access Control Mechanism used by our product. We will follow a 4-step adversarial approach to explain the exploitation of this attack base.

Research

After identifying the attack base, our task is find product ACL weaknesses by applying historical knowledge and analyzing code base.

To start with, we can perform code coverage of our product and look for any of the following improper ACL definitions types

a) NULL ACL Definition

b) Dangerous ACE types

c) Verify if proper Access Control Lists are defined for the product resources

Vulnerability disclosure websites such as CVE, Secunia or SecurityFocus are the major source of gathering historical knowledge about any Attack Base. Until Windows XP, Shatter Attack was one of the most prominent attacks to exploit Windows ACL. It takes advantage of a design flaw in Window's message-passing system where arbitrary code could be injected into any other application or service running in the same session. If your application has a graphical interface, which runs with higher Local System privileges, possibly it will fail with Shatter Attack.

Create Abuse Cases

Now, based on our research the following defined the foundation of Abuse Cases development.

Abuse Cases Development

- ✓ *Complexity of ACL's configuration*
- ✓ *Permissions cannot be assigned to all objects*
- ✓ *Exploiting Integrity Level (Windows Vista/Windows 7 specific)*
- ✓ *Shatter Attack*

Depending upon individual approach and product behavior, above list can be manipulated.

Execute & Record

A part of the execution is completed in the research section where we looked for improper ACL definitions in the code base. In addition, we can perform shatter attack on the product components or try to exploit Integrity Level.

Finally, we report the issues and document the results found by exploiting ACL.

For more information about this testing type, please refer to **Appendix B**, which displays the list of various attack bases together with their Abuse Case scenarios.

7. Conclusion

No doubt, in absence of a guided test scenario and lack of product management support, performing security testing becomes a tough task. It goes via myriad motivational issues, as success is not expected to come overnight (not even over months). To make security testing a part of the functional Black-Box testing, one would need to create the necessary skill-set first, which can be appended only by technical certifications, user group studies, basic understanding of the security concept, network/OS elements.

In today's market, increase in the number of security incidents has made security testing an inevitable part of SDLC. Hence, treating security testing as any other testing type, rather than continuing to give it a specialized treatment, would be a good start. Adaptive approach promises to empower functional testers to perform security testing and thus brings a security solution easy to implement and adopt by the management.

We hope that our model simplifies the understanding of application security testing and optimize the efforts put down by testers in finding security loopholes.

8. References

a) *Tom Gallagher, Bryan Jeffries and Lawrence Landauer , Microsoft Press Hunting Security Bugs, 2006*

b) *G. Hoglund and G. McGraw, Exploiting Software, Addison-Wesley, 2004.*

c) *http://news.cnet.com/2100-1001-240112.html*

d) *http://en.wikipedia.org/wiki/Operation_Aurora*

e) http://www.infosecwriters.com/text_resources/pdf/need_for_security_testing.pdf

f) http://www.zdnet.co.uk/news/it-strategy/2007/11/14/the-worst-it-security-incidents-of-2007-39290745/

g) http://en.wikipedia.org/wiki/Shatter_attack

h) http://en.wikipedia.org/wiki/Mandatory_Integrity_Control

i) http://archive.hack.lu/2007/cracking_windows_access_control.ppt

Appendix A: Peripheral Security Testing checklist

Table 1 Peripheral Security Testing Checklist

S.No.	Entry/Exit Points	Attack Model	Tools/Scripts
1	File & Folders	Information Disclosure Weak Permissions Buffer Overflow	FileMon ACL Editor strings
2	Sockets	Man-in-the-middle Attack Sniffing network traffic Send malicious data	Wireshark netstat.exe netcat
3	Registry Entries	Registry Accessed by the product Permission of the registry keys	Regmon ACL Editor
4	Named Pipes	Exploit weak permission Hijack the creation Impersonate the client	PipSec PipeList CreateAgentPipe ObjSD
5	User Interfaces	Shatter Attack Format String Attacks	Shatter Tool WebText Convertor
6	Command Line Arguments	Exploit Undocumented command Line switches	Command Line switches /?, -?, /h, or -h. Process Explorer Image tab
7	Environment Variables	Uncover Environment Variables used by Product Manipulating data inside Product defined Environment Variables	Process Explorer Environment Tab System Environment Variable Tab
8	ActiveX Control	ActiveX Repurposing Attacks ActiveX Fuzzing	COMRaider OLEView
9	Drivers	I/O Verification DeadLock Detection Dangerous APIs Exceptions/Handlers/Memory Pool Tracking Loading and Unloading Filter Driver Attach and Detach Filter Driver	Windows Utility-> Verifier.exe Windows Utility -> fltmc Microsoft Application Verifier Velocity Tool by Microsoft

Appendix B: Adversarial Security Testing checklist

Table 2 Adversarial Security Testing Checklist

S.No.	Attack Base	Abuse Case Scenarios	References and Historical Knowledge
1	Access Control List	Verification of apt ACL's for your product resources Target NULL DACL Look for dangerous ACE types -> Everyone (WRITE_DAC) -> Everyone (WRITE_OWNER) -> Everyone (FILE_ADD_FILE) Target Windows DAC weakness Target Windows MIC weakness	Shatter Attack http://www2.packetstormsecurity.org/cgi-bin/search/search.cgi?searchtype=archives&counts=26&searchvalue=win2000+attack+.c Exploiting Integrity Levels http://archive.hack.lu/2007/cracking_windows_access_control.ppt
2	Shell Extensions	List out shell extensions used by your product List the resources utilized by our Shell Extension Behavior of shell extension. Effect of impersonating your product shell extensions.	http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-5902
3	Plugins	List resources used by BHO Learn how to write a BHO Understand functionality of IE Plugin. -> This can give more attack vectors Effect of impersonating our product BHO Find a way to disable IE Plugin	http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-2382
4	Denial Of Service	Do basic analysis of DOS Attacks Identify the services rendered by your product Identify the ports used by the services Identify tools to send specially crafter packets to perform a DOS Attack on our product. (use historical info) Analyze the results	http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-1855