Reflections on Operating in Hostile Environments

Anil Bazaz, James D. Arthur, Randolph Marchany
Virginia Tech
Blacksburg, VA 24061
{abazaz, arthur, marchany}@vt.edu

Abstract

We introduce a generally applicable framework to assess and substantiate the security of a software component of a computer system. The framework constitutes a meta-model. Security models can be derived from it for components of a computer system. The concept of trust is interwoven into the meta-model and is an integral part of derived security models.

1. Introduction

A popular legend says that a computer system is secure if is twenty feet below ground in a locked basement without any connection to the outside world, and even then we cannot be sure. The above statement summarizes the difficulty in assessing and ensuring the security of a computer system. Computers today are being used for a wide variety of applications ranging from personal use to managing critical infrastructure. Moreover, the advent of the Internet has increased the accessibility of computer systems dramatically. This widespread use has made computers an *enticing* target for attacks and their accessibility makes them an *easy* target. Numerous attacks targeting computer systems are evidence of the fact that they are both easy and enticing targets. These attacks have caused damages worth millions of dollars (e.g., the denial of service attack on e-commerce websites in February, 2000) and much suffering. In 2001 an Australian man hacked into Maroochy Shire, Queensland waste management system and spilled millions of liters of sewage into parks and rivers. The computerized sewage system was not connected to the Internet and was accessible only through a modem. The man responsible was caught. It turned out that he worked for the company which had installed the computerized system [15]. This attack underlines the importance of security for even private networks. Any multi-user system, or a system connected to the Internet, or any network for that matter is susceptible to attacks. We can safely assume that such a system is operating in a *hostile environment*. As in any hostile environment protection of the system from attacks is a prime concern. Attacks can range from data disruption, denial of service, to an attempt to *own* the system or even destruction of the system. Attackers will employ a variety of tools and techniques to attack the system. An attack to the system can come from outside of the network or from inside the network. Steps must be taken to ensure the security of the system when operating in such an environment.

Assessing and ensuring the security of a computer system is a very difficult task. In his paper "Reflections on Trusting Trust" Ken Thompson illustrates the difficulty of assessing security by demonstrating how easy it is put a Trojan horse into the source code, and how difficult it is to detect it [19].

Several issues make assessing the security of a computer system a very complicated task. The first issue stems from the fact that modern computers are very complex systems composed of highly complicated and inter-dependent components. To assess the security of a computer system we have to assess the security of its components. Assessing the security of a component involves not only assessing its security but also the security of all the components whose services it utilizes. This makes security analysis of a computer system a very complicated task. The second issue is recognizing the vulnerabilities of a computer system. "*Know your enemy and know you self; in a hundred battles you will never be in peril* ", said Sun Tzu in the Art of War [18] and this holds true in the case of computer security too. However, there are many known vulnerabilities of a computer system e.g., buffer validation errors, heap validation errors etc. It is not

This research and presentation was supported in part by Virginia Tech Department of Computer Science, Systems Research Center, Digital Library Research Laboratory, and NSF Grant No. DUE-0121679.

practical to analyze an application for each known vulnerability. We need to analyze the vulnerabilities to identify the characteristics of the root causes of these vulnerabilities. These characteristics point to sources of the vulnerabilities. We refer to the sources of vulnerabilities of the system as "A*ttack Surface*"[1] [9]. We also define the term "*Attack Depth*" as the extent of damage caused if these vulnerabilities are successfully exploited. The terms *Attack Surface* and *Attack Depth* embody the potential threat to a system. The third issue stems from the fact that a vulnerability can be exploited in a number of ways. For example, buffer validation errors can be exploited by buffer overflow attack, off by one attack etc. An intelligent categorization of known exploits is needed to identify ways in which vulnerabilities can be exploited. The fourth issue is how to derive effective strategies for assessing security of a component of the computer system? The information about the sources of vulnerabilities and ways of exploitation has to be used effectively to evolve verification and validation strategies to assess security. The fifth issue is how do we address the reality that new vulnerabilities and new ways of exploiting vulnerabilities are being discovered constantly? Thus any technique for analyzing security of a computer system should be easily updateable. These issues have made the detection of security errors a very difficult, if not an impossible task.

In this paper we introduce a framework for assessing and substantiating security of a software component of the computer system. The framework consists of a meta-model from which models for verifying and validating security can be derived for any component of the computer system (operating system kernel, system software, and application software). The meta-model takes into consideration the concept of trust while verifying and validating security of the system.

The meta-model is still evolving; the major purpose of this paper is to present this idea to the security community and to gain valuable feedback from them. We also limit our discussion to software components since the framework concentrates only on the software components of the computer system.

[1] The term "Attack Surface" has been used in Howard *et al's*. book "Writing Secure Code" with similar meaning.

1.1 Related Work

There has been a significant amount of research on individual security threats. Tools have been developed to safeguard against specific vulnerabilities. For example, StackGuard is a tool for avoiding buffer overflow problem [10]. Significant research effort has also been put into building more secure operating systems, e.g., REMUS (Reference Monitor for UNIX Systems) [4]. Various attack taxonomies have also been developed: Protection Analysis Taxonomy [5], RIOS (Research In Secured Operating Systems) [1], etc. A large amount of literature (in fact too much) is available on individual security flaws and how to avoid them. Nonetheless, research that takes an integral view of the problem has been limited.

Considerable research has been done on process based approaches for analyzing security. Amoroso *et al.* describe the Trusted Software Methodology (TSM), a system for defining and measuring software trustworthiness [2]. Systems Security Engineering–Capability Maturity Model (SSE-CMM) is another process reference model for evaluating and improving security engineering practices in a organization [17]. Although process based approaches are essential in evaluating security, they are not a replacement for product based evaluation approaches. For example, a product developed by an organization having high SSE-CMM level is not necessarily security defect free. Thus, to evaluate the security of a computer system, product based evaluation approaches have to be identified.

Other approaches have also been developed for evaluating security. For example, Common Criteria for Information Technology Security Evaluation (CC) [8] is the combined effort of North American and some European governments to establish criteria for security evaluation in the information technology domain. CC is impaired by some basic disadvantages. Since CC addresses a wide range of products, it is specified in general terms and is subject to a variety of interpretations while being applied to a specific product. These interpretations determine the level of security evaluation and can be imprecise. CC also carries with it a significant bureaucratic baggage, and it is doubtful that it will be able to evolve with increasing threats [9]. Other similar approaches, e.g., TCSEC, ITSEC, suffer from similar disadvantages. Additionally, imposition of standards and criteria for engineering of the software has not proven to be decidedly effective to date. Pfleeger *et al.* [14], found 250 standards for the engineering of software, and concluded that they were mostly ineffective. Another

approach for certifying the security of a system is formal assurance methods. These methods have proven however, to be of limited use [16].

Our research differs from the earlier research in a number of ways:

1. The framework presents a comprehensive solution for the assessing security. It can be applied to any software component of a computer system, provided a proper model for that component is determined.
2. The framework introduces a product based approach for assessing the security of a computer system. It can be used for evaluating security even if no development documentation is available.
3. The concept of trust is interwoven into the framework.
4. The design of the framework is modular. The individual components of the framework can be easily updated or replaced as need arises.
5. We also intend to keep framework as simple as possible so that professionals will actually *use* it.

2. The Framework

In this paper we introduce a framework for deriving verification and validation strategies that supports a multi-level approach towards assessing and substantiating the security of a component of a computer system. The framework allows analysis of the component at any point during or after completion of its development lifecycle. The framework is comprised of two components: the trust levels and the meta-model. The *trust levels* categorize the components of the computer system into groups and organize them in a layered structure. These layers constrain the view of the computer system, thereby limiting the scope of security analysis and making it a manageable task. The *meta-model* is a system for assessing the security of a computer system component. Operational models can be derived from the meta-model for a trust level and applied to an application[1] operating at that trust level. For example, we can derive an operational model from the meta-model for application software. It is obvious that the operational models will be different for different trust levels. We are currently investigating how these models differ and/or overlap with trust levels.

[1] Note: In this paper application is used as a general term for any component of computer system. For example, operating system kernel is an application at trust level 1, hardware is an application at trust level 0.

Applications executing at any of the trust levels are complex and can be subdivided into many smaller components. Security assessment involves breaking an application down into its components and analyzing the components by using the meta-model. Analysis must also be carried out for the application as a whole. We conjecture that this approach will catch most, if not all security errors arising from the integration of components.

2.1 The Trust Levels

The trust levels categorize the constituent components of the computer system into four groups: (1) hardware, (2) operating system kernel, (3) system applications, and (4) user applications. Any computer system is composed of one or more of the above components. Hardware comprises of hardware components of system, e.g. the CPU, hard disk, memory, etc. The Operating system kernel is the core operating system program running on the hardware. System applications are applications providing operating system services to users and user applications, e.g., password program, authentication services etc. User applications are applications which users build and employ, e.g., Microsoft Word, Multimedia programs etc. Security of the system cannot be guaranteed until each of these components is secure.

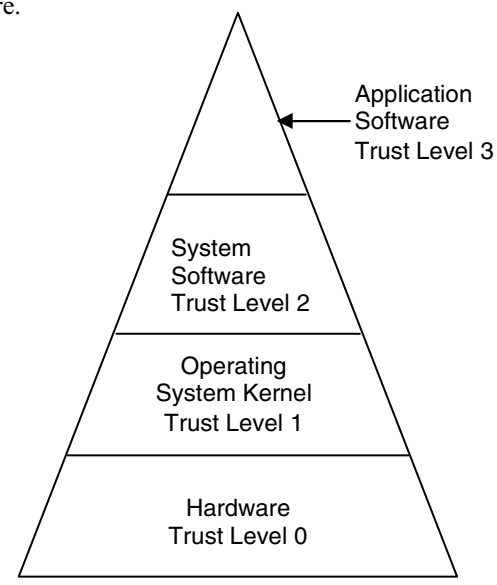

Figure 2.1 Trust Levels

These categories of components are organized in a layered structure. We define the layers as "*trust levels*" with hardware at trust level 0, the operating system kernel at trust level 1, the system software at trust level

2, and the application software at trust level 3. For a system component to be secure all trust levels below it must be secure. So before verifying and validating the security at any trust level we must assume the security at all the trust levels below it. Thus, trust level 1 depends on trust level 0, trust level 2 depends on trust levels 0 and 1, and trust level 3 depends on trust levels 0, 1 and 2.

Figure 2.1 illustrates trust levels of a computer system. The surfaces of the triangle (except the base) can be viewed as the attack surface. This surface is what is seen by an attacker, who can attack at any trust level of the system. Attack surface is thus equivalent to sum of attack surfaces at trust level 3, 2, 1 and 0.

Our approach to assessing security assumes initially that we can limit the attack surface. For example, assessing the security of a component at trust level 3 assumes impregnability of trust level's 2, 1 and 0. This makes security assessment of a system more manageable. We can develop our model and methods for that trust level accordingly.

2.2 The Meta-Model

The meta-model is used to assess and substantiate the security of a component of the computer system. It is composed of three components: (1) the threat model, (2) the taxonomy of attacks, and (3) Verification and Validation (V&V) strategies. The *threat model* is a compilation of the common characteristics of the root causes of vulnerabilities. It enables us to identify the sources of vulnerabilities that may be present in a software component and its damage potential. The *taxonomy of attacks* is an intelligent grouping of exploits based on the root causes of vulnerabilities. It identifies ways in which the sources of vulnerabilities, identified by the threat model, can be exploited. *V&V strategies* provide us with methods for analyzing the security of the software component. The sources of security defects (threat model) serve as targets for V&V strategies and the taxonomy provides us with ways of exploiting these sources. Together, the threat model and the taxonomy enable us to evolve V&V strategies for assessing and substantiating the security of a software component of the computer system.

Figure 2.3 illustrates the framework for assessing and substantiating the security of a component of the computer system. The succeeding subsections describe the components of the meta-model in detail.

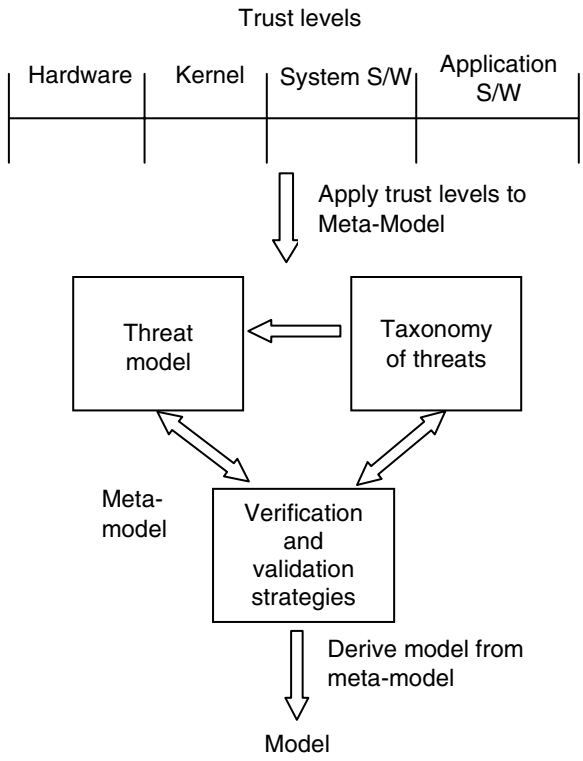

Figure 2.3 Relationship between meta-model and trust levels

2.2.1. Threat Model

The first component of the meta-model is the threat model. The threat model is a representation of the common characteristics of the root causes of security defects. These characteristics enable us to identify the sources of vulnerabilities in an application, thus establishing its attack surface. The threat model also provides an assessment of the damage (attack depth) if an attack is successfully accomplished. The threat model consists of three components:

1. **The Interface**
2. **Temporary components**
3. **Privilege levels**

An application receives input, produces output, and interacts with the environment through its interface and temporary components. Most of the applications at any trust level are attacked through manipulation of these three threat model components; these are the points at which an application is most vulnerable. For example, some applications operating with dynamic privileges are attacked during raising or lowering of privilege levels. Thus, Interface, temporary components, and

privilege levels provide us with the sources of vulnerabilities in an application, and are collectively referred to as *Attack Surface*. Additionally, privilege levels convey an idea of resources that can be accessed by the application, thus giving us an insight into the damage potential of the application. The privilege levels, therefore, gives us the *Attack Depth* of an application. Collectively Interface, Privilege level and Temporary components characterize the potential threat to the application.

The first component, the interface, is defined as the part of application that accepts input. Through the interface an application accepts input from its environment. It can be *local* if input is accepted only from local user, it can be *system wide* if input is accepted from multiple users in a multi-user environment, or it can be *global* if input is accepted from the network or Internet.

The second component of the threat model is the temporary components. This includes *temporary files, sockets, named pipes, RPC, registry variables, environment variables, etc.,* that are used by the application. The difference between the temporary components and the interface is that the temporary components are not used directly for input but are used during the course of computing. All of the above temporary components can be exploited in an attack on an application, and thus are sources of vulnerabilities.

The third component, privilege levels, determine the access an application process has to system resources. A process can run at user privilege, special privilege, administrator privilege, and can even have dynamic privileges. All of the above privilege levels are defined below:

1. **User Privilege**: An application process has user privilege level, if it has limited access to system resources and the impact of successful attack on the application will be limited to the user only. Different users in a system can have different resource accesses, but this has been ignored in our analysis. The reason is that even with different privileges the impact of a successful attack on a process being run at user privilege will be limited to the user.
2. **Special Privilege**: If the application process has access to resources which, if compromised, can have system wide impact, then the privilege level is classified as special privilege.
3. **Administrative Privilege**: An application process can also have administrator-level privileges. At this privilege level access to all the system resources is possible. A successful attack on an application with this privilege level can be devastating and can lead to the complete destruction of the system.
4. **Dynamic Privilege**: An application process can have dynamic privilege levels, that is, the privilege levels can change while the process is running. For example, setuid programs in Unix and its variants, exhibit dynamic privileges.

When considered together these three components (interface, temporary components and privilege levels) enable us to characterize the sources of vulnerabilities of an application and to estimate the extent of damage, if a successful attack is achieved on the application.

2.2.2. Taxonomy of Attacks

The second component of the meta-model is the taxonomy of attacks. The taxonomy categorizes the ways in which an application can be attacked. The taxonomy should be detailed enough to cover all the ways in which an application can be attacked. However it should not be a simple listing of attacks, but an intelligent grouping that captures the root causes of computer security errors. Furthermore, the categorization should map to the sources of vulnerabilities identified in threat model. An understanding of the relationship between the two enables us to evolve verification and validation strategies for assessing and substantiating the security of an application.

VERDICT, a taxonomy build by Daniel Lough [12] is currently being used as our taxonomy of attacks for this meta-model. VERDICT is an acronym for Validation Exposure Randomness Deallocation Improper Conditions Taxonomy. VERDICT goes beyond simple listing of attacks, and classifies attacks based on root causes of security problems. Taxonomies like *the taxonomy of security faults* developed at the COAST laboratory [3] can also be used as taxonomy of attacks.

VERDICT classifies all security errors into four categories. They are:

1. **Validation:** A security error is classified as a validation error if the error is caused by a violation of limits imposed by the system. Validation errors are a significant cause of security errors.
2. **Exposure:** A security error is classified as an exposure error if it reveals information that can be used directly or indirectly for exploitation of a vulnerability. An exposure error is not usually

a direct security error, but can be used in a multi-step attack on a system.

3. **Randomness:** A security error is classified as a randomness error if the error results from improper use of random sources for maintaining secret information.
4. **Deallocation:** A security error is classified as a deallocation error if a compromise stems from residual information left by the process.

For example, a Buffer Overflow exploit is caused by improper (or lack of) bounds checking that results in overwriting of the program stack. A carefully crafted input can be used to transfer the control of program to anywhere the attacker wants. VERDICT classifies Buffer Overflow exploit as a validation error. This is because the root cause of buffer overflow exploit is improper or lack of bound checking. Thus, VERDICT concentrates on the root cause of the security error rather than numerous attacks that exploit the root cause.

2.2.3. Verification and Validation Strategies

This section describes the third and final component of the meta-model, Verification and Validation (V&V) strategies. V&V strategies are used to assess and substantiate the security of the application based on the analysis from the threat model and the taxonomy of attacks. The sources of vulnerabilities of the application (threat model) serve as objects for V&V. The taxonomy of attacks provides us with ways of exploiting these sources of vulnerabilities. Together, they suggest the evolution of V&V strategies to assess and substantiate the security of the application.

We envision V&V strategies that provide us with three levels of assessment depending on the amount of information available about the development of a suspect component and the stage of development cycle that component is in:

1. **Validate Only:** This approach is used when we have a component where its development is complete and no development documentation for the component is available. Validation is achieved through only the execution of a suspect application.
2. **End game Verification and Validation:** This approach is used when we have the component (development is complete) and we also have the documentation for that component. We can have various levels of documentation. For example, we may have only the source code and nothing else, or at the other end we may have complete documentation about development of component along with its source. Clearly, validation is applicable because we have access to the executable code. Additionally, we can employ verification methods to development artifacts to determine the extent to which proper, security related development procedures were followed during the development process.
3. **Full Verification and Validation:** Full verification and validation is possible when the component is in the initial stages of the development cycle and we have complete access to development activities and artifacts. In this case we can instrument the development process to achieve full V&V.

Based on the stage of development that the application is in, and the amount of documentation available, one of the above three approaches is used to assess and substantiate the security of the application. Although it is desirable that we analyze a component during its development cycle, it is not practical to consider that this will always be the case. There will always be projects whose development cycle has been completed without due consideration given to security. Also there will be components developed for which no information is available. So it is very important to devise various levels of V&V for components at various stages of development.

2.2.4. Relationship between Meta-Model Components

All three components of the meta-model are inter-related. Those relationships form the key part of the meta-model. The threat model identifies the sources of vulnerabilities in an application and its damage potential. It thus establishes the attack surface and the attack depth of the application. The taxonomy of attacks identifies how vulnerabilities (identified by threat model) can be exploited to attack the application. V&V strategies provide us with methods for analyzing the security of the software component. V&V strategies are evolved using the sources of security defects (threat model) and ways of exploiting these sources (taxonomy of attacks).

3. An Example

To illustrate the power and applicability of the framework we analyze a “broken” version of the password program on SunOS and HP/UX (this is a historic case of the TOCTOU (Time Of Check Time

Of Use) flaw which was first introduced in [5]). In these operating systems, the user information and his/her password is stored in the .rhosts file in the users home directory. This particular version of the password program took the name of password file as input from the user. The program (1) opens and reads the password file and retrieves information about the user executing the program and then closes the password file, (2) then creates and opens a temporary file "ptmp" in the same directory as the password file, (3) opens the password file and copies the unchanged contents from old password file and the modified contents to the temporary file, (4) and then closes the password file and renames the temporary file to the new password file.

The above program is a system application and is thus operating at trust level 2. It is assumed here that all the trust levels below it are secure. We also assume that we have no documentation available about the development of the password application. So we will use the "validate only" approach.

We will begin the security assessment of the above application by deriving the model from the meta-model. This model has the same components as the meta-model but is specific to the above application. The model and its components are described in detail below.

3.1 Threat Model

The above application manipulates passwords in an operating system. Since passwords often are the only line of defense in a computer system, the application is highly sensitive. A security error in this application can lead to a complete compromise of the system. We shall begin the analysis by constructing its threat model. To construct the threat model for the above application, we need to identify the three components of the threat model: *interface*, *temporary components, and privilege levels*.

The above application operates in a multi-user environment and is available to all users of the system. So the application supports a *system wide* interface. It uses a *temporary file* as a temporary component. The above application operates with *dynamic privileges* as it changes its privilege level to access resources requiring administrative access. It operates with the privilege of a user but dynamically changes its privilege level to administrator level using setuid.

A number of sources of vulnerabilities are recognized as we analyze the application using our threat model:

1. **Interface:** The application takes input from all users of a system. This makes it vulnerable to attack from anyone who has an account on the system. The part of the application that accepts input is a source of vulnerability for the application.
2. **Temporary Components:** The application opens a temporary file with write access which gives attackers one more avenue for attack. The attacker can exploit the file to attack the system. Thus, the temporary file is a source of vulnerability for the application.
3. **Privilege Levles:** The application runs with dynamic privileges. It runs with user and administrator level privileges. Since the application can run with administrator privileges, it has the potential for completely subverting system security. Thus, changing of privilege levels is a source of vulnerability for the application.

This simple analysis completes the threat model. That is, the application has several sources of vulnerability. These sources can be exploited if errors are made in designing and/or coding the password program. Additionally, the application is used for manipulating passwords which makes it an enticing target. Although it does not have a network interface it does have the potential of providing root access, and if compromised, can be used in a multi-step attack on the system.

3.2 Taxonomy of Attacks

With the threat model complete we must use the taxonomy of attacks to identify ways in which the sources of vulnerabilities identified by the threat model can be exploited. Analysis from the threat model shows that the application has sources of vulnerabilities (accepts system wide input, uses temporary file, and runs with dynamic privileges) and that it runs with administrative privilege levels for a period of time. We shall now apply the taxonomy of attacks, VERDICT, to the application to see how these sources can be exploited.

3.2.1 VERDICT Validation

VERDICT Validation errors are caused by the violation of limits imposed by the system. In this section we will analyze ways in which an attacker can

exploit vulnerabilities of the password application having validation errors.

Validation errors for a system wide interface:

1. The buffers used for storing of all input can be exploited by attacks. Attacks exploiting buffer overflow, heap overflow, format string, etc. can be mounted on the application if proper bounds checking is not performed.
2. The rights of the user invoking the application can be exploited, if not properly validated. Any user can change any other users password if the rights are not properly validated.
3. Since the password file is provided as input to the application, it too can be exploited if it is not validated. The password file of another user can be provided to the password application to manipulate the passwords of other known users, if proper validation is not performed.

Validation errors for temporary files being used:

1. Temporary files can be exploited in attacks on an application. A user can change the contents of a temporary file or even remove that file while the application is still using it. A temporary file can be created *a priori* with full access permission given to an attacker, and then used to manipulate passwords of other users while their applications are still executing.

Validation errors for dynamic privileges:

1. Dynamic privileges can be exploited to gain administrative access to the system. The attacker can force an application to stop execution before terminating root privilege and use its process to gain root access to the system.

3.2.2 VERDICT Exposure

Exposure errors deal with revealing information that can be used for direct or indirect attacks on an application. In this section we analyze methods of exploiting such vulnerabilities.

1. Improper access permission on the password files can reveal information that can be used indirectly for attacks on the application.
2. Improper access permissions on the temporary file can reveal information that can be used for attacking the application.
3. Information about a temporary file can be used to attack an application. The users of the application need not know the location and the name of the temporary file being used by the application, but in this case they did.

3.2.3 VERDICT Randomness

Security errors resulting from improper use of random resources are classified as randomness errors. We now identify ways of exploiting these types of vulnerabilities in the password application scenario.

1. Improper or lack of use of a one-way function for storing passwords in the password file. This can be used to break or simply read the passwords of other users.
2. Improper or lack of use of randomness for keeping the name of temporary file secret. If the name of temporary file is revealed, it can be exploited to attack an application.

3.2.4 VERDICT Deallocation

Security errors caused by residual information are classified as deallocation errors. In this section we examine ways of exploiting vulnerabilities in the password application through the use of deallocation errors.

1. An improper procedure for deleting the original password file can be exploited to attack an application. If the original password file is not properly deleted, it can be used to reveal information about other users of the system.

3.3 Verification and Validation Strategies

Sections 3.1 and 3.2 outlined sources of vulnerabilities and ways of exploiting these sources in the password application. This section introduces V&V strategies to ensure that the application is secure. This is the final component of the meta-model.

We assume that for the password application we have access only to the executable files and a general description of how the application works. We do not have access to any development documentation. Since no development documentation is provided for the application we can only apply Validation[1] methods on the application. That is, we can only execute the application to determine validity characteristics. Validation strategies are provided next that help in assessing the security of the application. They are organized according to ways of exploiting vulnerabilities characterized by the taxonomy of attacks outlined in the previous section.

Validation strategies for VERDICT Validation errors:

[1] Note: Validation in this section differs from Validation errors in Taxonomy of Attacks. Validation here refers to validation strategies for detecting security errors.

1. Validate if all input buffers have proper bounds checking procedures.
2. Validate if the user invoking the application has rights to manipulate password file and the passwords of user that he/she tries to manipulate.
3. Validate if the application determines that the input file being used is the valid password file.
4. Validate if the application overwrites the real password file even if the input file provided is invalid.
5. Validate that the access permissions of the temporary file being created are appropriate. The user should not have any access rights to it.
6. Validate if the new password file has proper access permissions.
7. Validate if the administrator privilege (assuming the application uses dynamic privileges) is terminated properly for all cases.

Validation Strategies for VERDICT Exposure errors:

1. Validate that the password files of the users have proper access permissions and does not reveal information about other users.
2. Validate if the temporary file has proper access permissions and does not reveal information.
3. Validate if the temporary file and its location are kept secret.

Validation Strategies for Randomness errors:

1. Validate that a strong one-way function is being used to encrypt the passwords of the users in the password file.
2. Validate that proper random variables have been used by the application to keep the name of the temporary file a secret.

Validation Strategies for Deallocation errors:

1. Validate that the proper deletion procedures have been used for deleting the old password file.

3.4 Results

The above is an example of application of the meta-model for a system application. The application used was a simple one; its operational scenarios contained additional simplifying assumptions. Nonetheless, it does exhibit those characteristics that enabled us to convey the relationships among the meta-model components.

On performing our analysis we see that the threat model supports a characterization of the sources of vulnerabilities of the application and enables an estimate of the damage potential (assuming that the vulnerabilities are successfully exploited). The taxonomy of attacks indicates how those sources can be exploited. V&V strategies are evolved based on characteristics identified by the threat model and taxonomy of attacks, and then used to detect the security vulnerabilities in the above application.

If the V&V strategies outlined in section 3.2 had been applied to the system software before it was released, multiple security vulnerabilities would have been discovered in that application. In particular, the verification and validation strategies would have reveled that the temporary file name and its location are not hidden by the application, a serious security flaw. Security errors like these form the building blocks for the TOCTOU (Time of Check Time of Use) flaw. An application of our framework would have revealed these errors, and the TOCTOU flaw could have been avoided.

4. Conclusion

We have introduced a framework for assessing and substantiating the security of a software component of computer system. The framework provides us with a systematic and standardized approach to assess the security of a computer system component. In particular, the trust levels provide us with a multi-level approach towards assessing security by dividing a computer system into multiple levels. The meta-model is used to assess security at any of the trust levels. The meta-model uses a threat model to identify sources of vulnerabilities of the system and gauge the amount of damage in case a successful attack is mounted on the application. The taxonomy of attacks provides a categorical classification of the ways in which the sources of vulnerabilities can be exploited. V&V strategies enable us to assess the extent to which a component is secure. Furthermore, verification and validation can be performed at any stage of development depending on how and when development information is available. Although security assessment of a component during its development is best, it is not vital for the success of the analysis using the framework. The real success of the analysis is a function of how well the synergy between the threat model, the taxonomy of attacks and the verification and validation strategies permit the identification of security flaws. Although in its infancy, we do contend that this synergy exists and indicates a step in the right direction.

The framework introduced in this paper is not limited to computer systems and can be applied to any

application that operates in a hostile environment and wants to improve its security.

5. Future Work

The meta-model is in an evolving research effort and much more still remains to be done. A quantitative system for measuring potential threats to an application needs to be integrated into the threat model. We also need a wider taxonomy to enhance the relationship between the meta-model components. More effective approaches for evolving V&V strategies have to be devised. The synergetic relationships between the components of the meta-model need to be refined.

6. References

[1] Abbot, R.P. et al., "Security Analysis and Enhancements of Computer Operating Systems", Technical Report NBSIR 76-1041, Lawrence Livermore Laboratory, Institute for Computer Science and Technology, 1976.

[2] Amoroso, Ed, "Towards an Approach to Measuring Software Trust", IEEE Computer Society Symposium on Research in Security and Privacy, 1981, pp. 198-218.

[3] Aslam, T., Krsul, I., and Spafford, E. H., "Use of Taxonomy of Security Faults", 19th National Information Systems Security Conference, 1996.

[4] Bernaschi, M., Gabrielli, E., and Mancini, L. V., "REMUS: A Security Enhanced Operating System", ACM Transactions on Information and System Security, Vol. 5, No. 1, 2002, pp. 36-61.

[5] Bisbey, Richard II and Hollingworth, Dennis, "Final Report Research ISI / SR-78-13", University of Southern California, Information Sciences Institute, 1978.

[6] Bishop, Matt, "How to Write a Setuid Program", Cray User Group Proceedings, 1986.

[7] Bishop, Matt and Dilger, Michael, "Checking for Race Conditions in File Accesses", Computer Systems, 9(2), 1996, pp. 131-152.

[8] Common Criteria Board, "Common Criteria for Information Technology Security Evaluation", Version 2.1, 1999.

[9] Computer Science and Telecommunications Board, "Trust in Cyberspace", National Academy Press, 1999.

[10] Cowan, Crispin et al., "StackGuard: Automatic Adaptive Detection and Prevention of Buffer-Overflow Attacks", Proc. 7th USENIX Security Conference, 1998, pp. 63-78.

[11] Howard, Michael and LeBlanc, David C. "Writing Secure Code", Second ed., Microsoft Press, 2002.

[12] Lough, Daniel L., "A Taxonomy of Computer Attacks with Applications to Wireless Networks", PhD Dissertation, Virginia Tech, 2001.

[13] Parnas, D. L., et al., "Evaluation of Safety-Critical Software", Communications of the ACM, Vol. 33, No. 6, 1990.

[14] Pfleeger, S. L., Fenton, N., and Page, S., "Evaluating Software Engineering Standards", IEEE Computer, Volume 27, No. 9, 1994, pp. 71-79.

[15] Simth, T., "Hacker jailed for revenge sewage attacks", the Register, 2001, available at http://www.theregister.co.uk/content/4/22579.html.

[16] Smith, R. E. "Cost Profile of a Highly Assured, Secure Operating System", ACM Transactions on Information and System Security, Vol. 4, No. 1, 2001, pp. 72-101.

[17] Systems Security Engineering-Capability Maturity Model Project, "Model Description", Version 2.0.1, 1999.

[18] Tzu, Sun, "The Art of War", 500 B.C.

[19] Thompson, Ken "Reflections on Trusting Trust", Communications of the ACM, Vol. 27, No. 8, 1984, pp. 761-773.

[20] Viega, John and McGraw, Gary "Building Secure Software: How to avoid security problems the right way", Addison Wesley Professional, 2001.