# Comparing the Vulnerabilities of Software Architectures Using Attack Graphs

Chih-Cheng Lien[1] Shih-Chang Hsu[1] Kai Qian[2] Chih-Cheng Hung[2] Ju-An Wang[2]
Department of Computer and Information Science Soochow University[1]
e-mail： cclien@cis.scu.edu.tw[1]
School of Computing and Software Engineering Southern Polytechnic State University[2]
e-mail： kqian@spsu.edu[2]

## Abstract

Software architecture design is an important stage for early understanding the properties of a system. Security properties could be explored in an architecture for gaining benefits from the current approach of giving patches to recovery the happened vulnerabilities. In this paper, we proposed a model of attack graphs to study some security properties of a software architecture. We formalized the notations and relationships in the model, thus it is suitable for studying the security aspects of architectures for architects. Furthermore, we derived some indices to compare the security properties of different architectures. Then, a better design of security concern among some candidates could be found and explained with the formalization of indices.

**Keywords**: attack graphs, security, software architecture, vulnerability.

## 1. Introduction

Attack graphs have been widely adopted to study the security property of system design [6].To handle the security concern of software development in the earlier stages, it could acquire some benefits better than the current approach of providing patches for recovering the happened vulnerabilities[9]. Architecture design is an important stage for understanding the properties of a system early[1, 8]. In this paper, we adapted a model of attacks to study the security property of software architectures. Some indices are formalized to compare the security property of different architectures. This approach can be used to improve the understanding of security concern of architecture design.

Attack graphs have been used to document the possible states of vulnerability[3, 6].An attack graph captureing the paths attackers could be used to reach the target of attack. Some researcheres use them to study the vulnerabilities of system for migrating the risks found in the attack graphs. Usually, there are too many states from these studies to make them suitable for modeling attacks in a software architecture[7, 12].

Recently, an informal attack graph which is used to analyze the system's purpose and potential goals of attacking was proposed byGupta[10]. The purpose of the graph is to study the risks known at the architecture design. It can help architects and analysts to understand the risks of a system. So, software can acquire the benefits of earlier risk management of systems. However, the informal model cannot be used to compute some security properties for deciding the trade-offs among different designs.

In this paper, we formalized the attack graphs corresponding to software architectures. This formalization provides a common base for the communication among architects. And, some indices could be derived for presenting the security properties of an architecture. Thus, the goodness of security concern among architectures can be compared with the related indices. This is an improvement of the attack graph proposed by Gupta[10], which can some quantitative comparisons.

The remainder of this paper is organized as follows. section 2 describes two software architectures and the related attack graphs. Then, the model of the attack is formalized. section 3 derives some indices for comparing the security properties of the two architectures. Finally, section 4 gives the conclusions of this paper.

## 2. Software architectures and attack graphs

In this section, we will examine two architectures modified form [10]. Then, the formalization

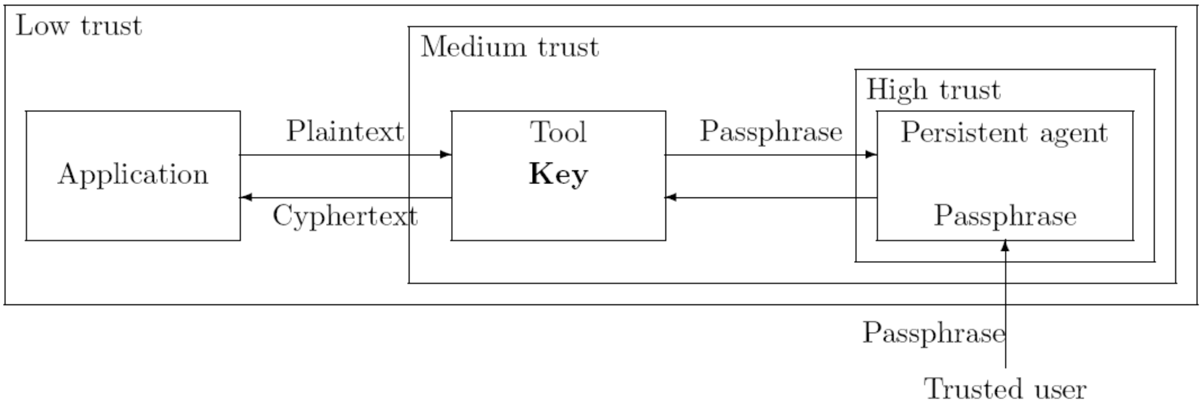


Figure 1: An architecture of the original design.

of the attack model will be explained.

A distributed information system is designed at first, shown in Fig. 1, where users retrieved the data managed by a persistent agent through the interface of application. Since the security goal of this system is to protect sensitive data managed by the distributed agents under various constrained circumstances. Thus, the ciphertext is distributed by a persistent agent for including the security of information, e.g., account of bank. And the user has to decrypt the ciphertext with the key which should be acquired by requesting the passphrase. The passphrase was installed by the trusted user when the encryption tool starts up. On the other hand, the plaintext input by the user will be encrypted by the encryption tool. Thus, the information security of this system can be set by the mechanism of cryptography with the aids of the encryption tool.

After some discussion based on the above description, some goals of possible attacks will be happened. The first goal is to gain the access of the sensitive data in the agent. Next possibility is to steal the key, then attacker can access the sensitive data in the systems with the key. The third possibility is to control a host to launch another attacks. From the economy of attacking, a stolen key can be used to expose all data protected by the key, so the second goal is encouraged to get more attacking than the others. For protecting the security of persistent data, some attack surfaces of the system are identified to support different security levels requirement. The persistent agent is put on the system of high trust to guarantee the top security of persistent data. The encryption tool is protected by the wall of middle -trust system. And, the interface of application is arranged into the part of low-trust system. Thus, the architecture of the original design is depicted in Fig. 1.

The possible attacks would be caused by people and systems, some of them are discussed as follows. The first case may be happened on the trust user of initializing passphrase. If this user is upset by some social effects, the passphrase of the system will be acquired without permission. And, the key will be explored by the adversary. The next scenario may be the encryption tool was compromised or impersonated. The third possible scenario is the application interface was compromised. The final possible scenario is caused by compromising the account of operating system to capture the account of tool.

We present the relationships of possible attacks in a graphical diagram for a better understanding of logical structure. In the diagram, rectangles denote subsystems or states of subsystems. Arrows used to present the causal dependencies between two states of subsystems. A solid bar in the input direction of a rectangle implies that all the inputs should be satisfied in order to complete the state of the subsystem denoted in the rectangle; i.e., the effect of an “and” node for the inputs. Otherwise, any input will cause the state of the subsystem denoted in the rectangle; i.e., the effect of an “or” node for the inputs.
The possible attack graph of the architecture of the original design as shown in Fig. 1, could be depicted as Fig. 2.

An attack model is a machine $M = (S, C, \delta, \nu, s_0)$, where $S$ is a set of subsystems or states of subsystems, $\delta \subseteq S \times S$ is a transition relation, and $s_0 \in S$ is an initial subsystem or states of subsystem. In the following, we will use the terms of “subsystem” and “states” interchanged

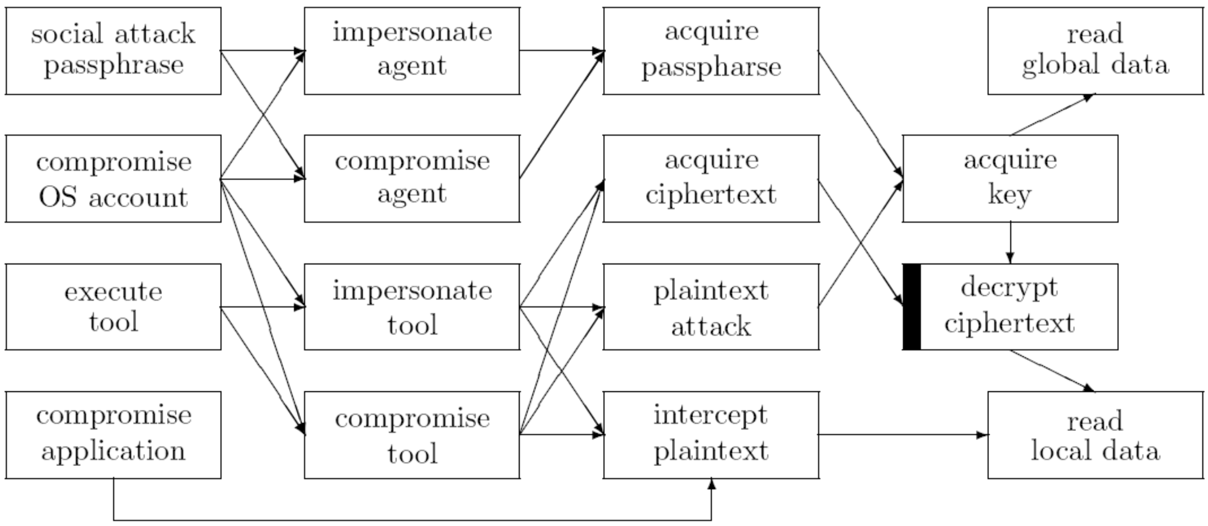


Figure 2: Some possible attacks of the architecture of the original design.

if no clarification has to be noted. For example, in Fig. 2, there are 16 states even in which some are subsystems viewed from architect. The initial state or subsystem should be a virtual subsystem to initialize the four node aligned in the left column of the diagram, we do not figure it out for saving the space of typesetting.

Set $C$ is a set of connectors for inputs of states, and $\nu \subseteq S \times C$. An "and" connector makes the conjunction of the inputs of a state. On the other hand, an "or" connector shows the disjunction of the inputs of state. In the attack graph, an "and" node are specified a solid bar in the input side of a state, and an "or" node has not the solid bar. For the possibility of a path of an attack is seldom set to be zero, currently, we set the outputs of a state will be in the case of conjunction.

A finite execution of an attack model $M = (S, C, \delta, \nu, s_0)$ is a finite sequence of states, $\beta = s_0 s_1 \ldots s_n$, where $(s_i, s_{i+1}) \in \delta$, for all $0 \leq i < n$. An attack of the model is a finite execution. For example, if a hacker attacked the passphrase by some social behaviors, then use it to impersonate the agent. And, he acquired the passphrase, and obtained the key with the passphrase. Finally, he can hack the global data with the stolen key, so the system security was compromised by the hacker. This sequence of states or subsystems yields and explains an attack was happened.

For a sequence of an attack, if the final state $s_n$ is a subsystem, then the subsystem is the target of the attack, if it is a state, it is a vulnerability of the system. Thus, we can analyze the vulnerabilities and the suffering targets in a system architecture with the model. This flexibility may be useful to architect. The architect can concern the attack target in a general view or he can find the vulnerability in a specific state of the subsystem. Considering Fig. 2, the two states of attacks are reading global data or local data without the permission of persistent agent. If we change the term of these rectangles to persistent agent, the target of attacks will be the agent. The partition of global and local data causes a more detailed analysis about the architecture of Fig. 1, however, it is not obviously identified from the architecture.

For comparing the security concerns of different architectures, a revised version could be derived as follows. The initial architecture solved this requirement of passphrase by storing it in memory such that the trust operator does not have to enter the passphrase at the console every time the tool starts. The passphrase will be protected by the encryption tool. Impersonating the encryption tool gives an attacker the possibility to explore the passphrase from the passphrase agent directly. And, the key will be captured for attacking the global data in the system.

This revised architecture, shown in Fig. 3, is conscious that the key may be stolen by attacking the transmission of passphrase. This is the same consideration as in the SSH protocols[11]. Thus, the revision moves encryption functionality from the command-line tool to the persistent agent. The application interface remains the same as the original design. A trusted user still enters the passphrase into the agent once, at the boot time.

Now, in this new architecture, the command-line

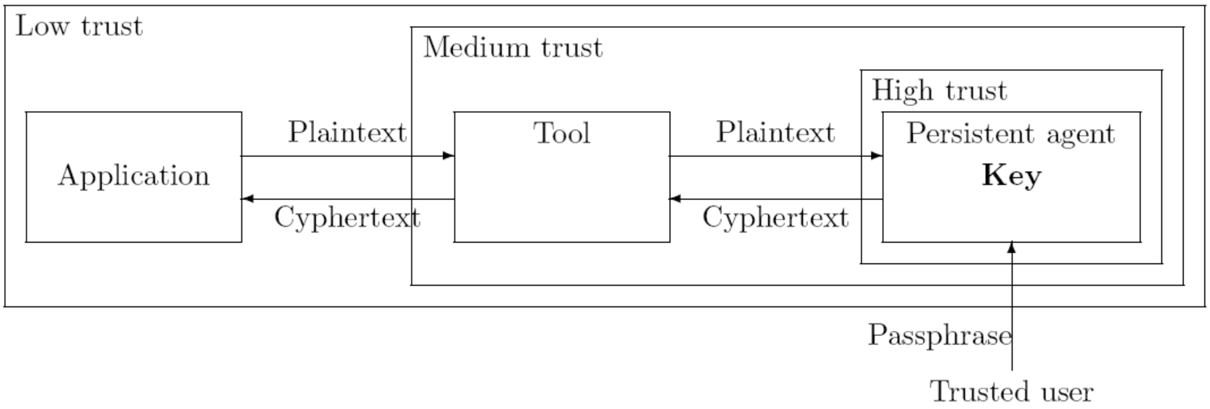


Figure 3: An architecture of the revised design.

tool needs not be trusted, but the persistent agent does not leak secrets. The command-line tool streams the data to be encrypted to the agent,then the agent does all the encryption operations without leaking out the key or the passphrase. The key or the passphrase remains inside a high-trust boundary to protect the possible attacks.

However, some risks still remain in the new architecture, the attacker could impersonate the command-line tool to contact the agent and decrypt files. Even the attacker can decrypt individual files by this manner, no direct access to the keys or the passphrase will be happened that would allow attacks on the group of data. A motivated attacker might still attempt to attack the keys by using a plaintext or ciphertext attack with the cryptographic skills, but such an attack requires more skillfully attacker than the attacker in the first design. Thus, the possible attack graph of the revised architecture could be described as Fig. 4.

Software architecture shows the whole picture of software design, however, at present it does seldom consider the security requirement at the architecture design stage. The attack graph of an architecture shows what attacks the designer considered explicitly. This would be a great help to system development for built-in software security, and it is good for maintainers, reviewers, and testers to understand the system's security posture and risk trade-offs in a system[9].

Even though the architecture of a simple system as illustrated, the corresponding attack graph may be very complex. Unless it is a trivial system, exhaustive list of all possible attacks is not feasible for the limited resources and time span. However, the slices of architecture for studying some interesting aspects is often can be conducted[8]. Thus, the attack graph of software architecture may be more simple than the graph at programming level or at some complex network systems.

## 3. Comparison of the attack graphs

In this section, we will derive some indices to compare the security properties of architectures. With these quantitative comparisons with indices, an architecture of better security concern can be searched for by architects.

Two indices can be derived to check the security quality of an attack graphs, as follows,

1. *The number of states in an attack graph.* For example, in Fig. 2 and Fig. 4, there are 16 and 14 states, respectively. So, form the viewpoint of vulnerabilities, the first system architecture may be worse than the second one.

2. *The number of attack paths to the target of attacks.* An attack is harmful if it will be arrival at the target of attack. Thus, this index may be better than the above index because the more vulnerabilities does not infer the more attack paths.

   To calculate the number of attack paths to the target of attacks, we first compute the number of attack paths for a target. Then we can summarize the number of attack paths of each target into the total number of attack paths in an attack graph.

   To compute the number of attack paths for a target, we can trace backward from the target to the attackers. This traversal will yield a tree like Fig. 5, where is the result

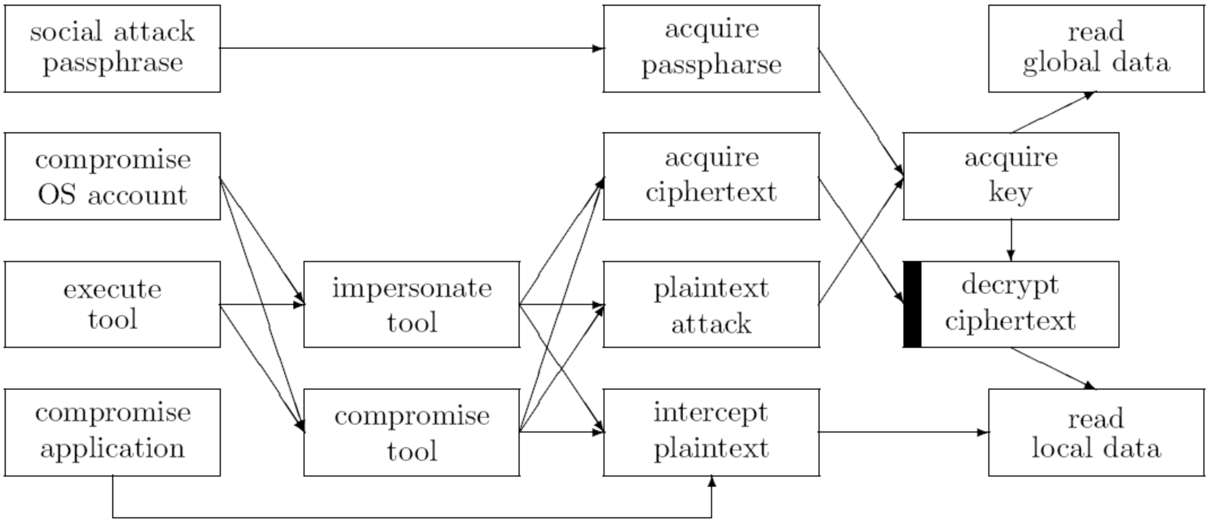


Figure 4: Some possible attacks of the architecture of the original design.

of tracing target "read local data" backward in Fig. 4. For saving the typesetting space, the names of the nodes in the attack graph have abbreviations as follows,

| | |
|---|---|
| ca | compromise application |
| et | execute tool |
| co | compromise OS account |
| sa | social attack passphrase |
| ct | compromise tool |
| it | impersonate tool |
| ca | compromise agent |
| ia | impersonate agent |
| ip | intercept plaintext |
| pa | plaintext attack |
| ac | acquire ciphertext |
| ap | acquire passpharse |
| rl | read local data |
| dc | decrypt ciphertext |
| ak | acquire key |
| rg | read global data |

Now, we can compute the number of attack paths to the target from the tree. If the children of a node yield an "or" relationships, then the number of the paths to the node will be the number of the children. For example, the node labeled "ip" has three paths from its children to it. Considering the combination of the descendants of this node, there are five paths form the descendants to the node.

On the other hand, if the children of a node yield an "and" relationships, then the number of the paths to the node will be one which combines all the children. For example, the node labeled "dc" has one path from its children to it. However, there are four and five paths to the nodes of the children, node "ac" and node "ak", respectively. Considering the combination of the descendants of node "dc", there are 20 paths, production of 4 and 5 paths, form the descendants to the node. Finally, the number of attack paths to the target "rl" will be 25, i.e., 5+20.

By the similar method, we can find that there are five paths to attack the target "read global data" in Fig. 4. On the other hand, there are eight paths to attack the target "read global data" in Fig. 2. Meanwhile, 37 paths can be traced to attack the target "read local data" in Fig. 2.

Comparing the numbers of attack paths in the attack graphs, we could find the property of the attack graph of the original architecture is worse than that of the revised one. And, the revised architecture is better than the original one about the software security.

The other indices can be devised to compare the security property of different architectures[5]. For example, to attach the cost or the possibility of attack paths will get some more precisely measurements than the above index comparison by the number of paths only. It is easy to extend this idea.

We have extended the attack graph proposed by Gupta[10]. We formalized the attack model and attack graph. Moreover, we derived indices to compare the security concerns for architects.

## 4. Conclusions

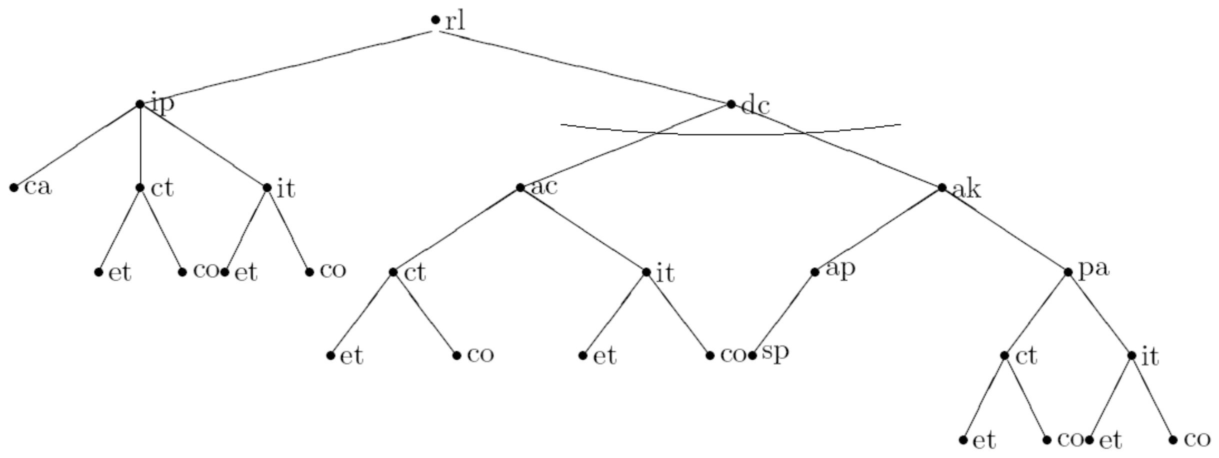


Figure 5: Some possible attacks of the architecture of the original design.

We had proposed a model of attacks to study some security properties of software architecture.We formalized the notations and relationships in the attack graph. So, it is suitable for architects to study the security aspects of architectures for. Furthermore, we derived some indices to quantify some security properties of software architecture. Thus, architects can use the indices to compare the security properties of different architectures. Then, a better design of security among candidates could be found and explained with the formalization of indices.

Attack patterns have been studied for understanding the behavior of various kinds of attacks [2, 4]. Discovering the possible attack patterns from an architecture may be useful to engage the security concern of software development. In the future, we are going to search for the useful attack patterns with the proposed model. If once this is completed, the built-in security of software may be improved.

## References


[1] G. Peterson and J. Seven, “**Defing misuse within the development process**”, IEEE Security and Privacy, vol. 4, no. 6, pp. 81-84, 2006.

[2] J. A. Whittaker and H. H. Thompson, **How to Break Software Security: Effective Techniques Security Testing**. Pearson Eduction, 2004.

[3] K. Ingols, R. Lippmann, and K. Piwowarski, “**Practical attack graph generation for net- work defense**”, in Proceedings of the 22nd Annual Computer Security Applications Conference (ACSAC’06). IEEE Computer Society, 2006, pp. 121-130.

[4] M. Andrerws and J. A. Whittanker, **How to Break Web Software: Funtional and Security Testing of Web Application and Web Services**. Addision Wesley, 2006.

[5] M. Jahnke, C. Thul, and P. Martini, “**Graph based metrics for intrusion response measures in computer networks**”, in Proceedings of 32nd IEEE Conference on Local Computer Network. IEEE Computer Society, 2007, pp. 1035-1042.

[6] O. Sheyner and J. Wing, “**Tools for generating and analyzing attack graphs**”, LNCS, vol. 3188, pp. 344-371, 2004.

[7] O. Sheyner, “**Scenario graphs and attack Graphs**”, Ph.D. dissertation, Carnegie Mellon University, 2004.

[8] R. S. Pressman, Software Engineering: **A Practitioner’s Approach**, 6th ed. R.S. Pressman and Associates, 2005.

[9] R. Sinn, Software Security Technologies: **A Programmatic Approach**. Thomson Course Technology, 2008.

[10] S. Gupta and J. Winstead, “**Using attack graphs to design systems**”, IEEE Security & Privacy Magazine, vol. 5, no. 4, pp. 80-83, July-Aug. 2007.

[11] T. YlÄonen, “**SSH: Secure login connection s over the internet**”, in Proceedings of 6th Usenex Secuirity Symposium, 1996, pp. 37-42.

[12] X. Ou, W. F. Boyer, and M. A. McQueen, “**A scalable approach to attack graph generation**”, in CCS ‘06: Proceedings of the 13th ACM conference on Computer and communications security. New York, NY, USA: ACM, 2006, pp. 336-345.