Advances in Information Security 100

Sushil Jajodia · Anup K. Ghosh
V.S. Subrahmanian · Vipin Swarup
Cliff Wang · X. Sean Wang *Editors*

Moving Target Defense II

Application of Game Theory and Adversarial Modeling

Springer

Moving Target Defense II

Advances in Information Security

Sushil Jajodia
Consulting Editor
Center for Secure Information Systems
George Mason University
Fairfax, VA 22030-4444
email: jajodia@gmu.edu

The goals of the Springer International Series on ADVANCES IN INFORMATION SECURITY are, one, to establish the state of the art of, and set the course for future research in information security and, two, to serve as a central reference source for advanced and timely topics in information security research and development. The scope of this series includes all aspects of computer and network security and related areas such as fault tolerance and software assurance.

ADVANCES IN INFORMATION SECURITY aims to publish thorough and cohesive overviews of specific topics in information security, as well as works that are larger in scope or that contain more detailed background information than can be accommodated in shorter survey articles. The series also serves as a forum for topics that may not have reached a level of maturity to warrant a comprehensive textbook treatment.

Researchers, as well as developers, are encouraged to contact Professor Sushil Jajodia with ideas for books under this series.

For further volumes:
http://www.springer.com/series/5576

Sushil Jajodia • Anup K. Ghosh • V.S. Subrahmanian
Vipin Swarup • Cliff Wang • X. Sean Wang
Editors

Moving Target Defense II

Application of Game Theory and Adversarial Modeling

Springer

Editors

Sushil Jajodia
George Mason University
Fairfax, VA, USA

Anup K. Ghosh
George Mason University
Fairfax, VA, USA

V.S. Subrahmanian
University of Maryland
College Park, MD, USA

Vipin Swarup
The Mitre Corporation
McLean, VA, USA

Cliff Wang
U.S. Army Research Office
Triangle Park, NC, USA

X. Sean Wang
Fudan University
Shanghai, China

ISSN 1568-2633
ISBN 978-1-4614-5415-1 ISBN 978-1-4614-5416-8 (eBook)
DOI 10.1007/978-1-4614-5416-8
Springer New York Heidelberg Dordrecht London

Library of Congress Control Number: 2011935538

Preface

One of the most vexing problems in defending against computer intrusions is the seemingly endless supply of exploitable software bugs that exist despite significant progress in secure software development practices. At least once a month (e.g., on Patch Tuesday), major software vendors publish patches that fix the vulnerabilities in their deployed software code base that have been discovered. These patches are often published after the vulnerabilities are known and have been exploited, in some cases for months and years. In currently deployed systems, the attacker has a static target to study and find vulnerabilities, and then a window of exposure to exploit the vulnerability to gain privileged access on other people's machines and networks, until the exploit is noticed, vulnerability found, patch released, and then applied widely. The dynamics of this process significantly favors the attacker over the defender because the attacker needs to find only a single exploitable bug while the defender must ensure none exist. The attacker has plenty of time to analyze the software code, while the defender does not know when the attacker will strike. And finally, the defender typically can only block the exploit once the exploit or vulnerability is known, giving the attacker an automatic advantage in gaining access with zero-day vulnerabilities.

Against this backdrop, the topic of moving target defenses (MTDs) was developed to level the playing field for defenders versus attackers. The basic concept of MTD is to dynamically vary the attack surface of the system being defended, thus taking away the adversary's advantage of being able to study the target system offline and find vulnerabilities that can be exploited at attack time. MTD systems offer probabilistic protections despite exposed vulnerabilities, as long as the vulnerabilities are not predictable by the adversary at the time of attack. MTD has been identified as one of the four key areas of thrust in the White Houses strategic plan for cyber security research and development.

In the first volume of MTD, we presented papers on MTD foundations, MTD approaches based on software transformations and network and software stack configurations. In this follow-on second volume of MTD, a group of leading researchers describe game–theoretic, cyber maneuver, and software transformation approaches for constructing and analyzing MTD systems.

Acknowledgments

We are extremely grateful to the numerous contributors to this book. In particular, it is a pleasure to acknowledge the authors for their contributions. Special thanks go to Susan Lagerstrom-Fife, senior publishing editor for Springer, and Courtney Clark, editorial assistant at Springer for their support of this project. We also wish to thank the Army Research Office for their financial support under the grant number W911NF-10-1-0470.

Fairfax, VA

Sushil Jajodia
Anup K. Ghosh
V. S. Subrahmanian
Vipin Swarup
Cliff Wang
X. Sean Wang

About the Book

The chapters in this book present a range of MTD challenges and promising solution paths based on game–theoretic approaches, network-based cyber maneuver, and software transformations.

In Chap. 1, Manadhata explores the use of attack surface shifting in the moving target defense approach. The chapter formalizes the notion of shifting a software systems attack surface, introduces a method to quantify the shift, and presents a game–theoretic approach to determine an optimal moving target defense strategy. In Chap. 2, Jain et al. describe the challenging real-world problem of applying game–theory for security and present key ideas and algorithms for solving and understanding the characteristics of large-scale real-world security games, some key open research challenges in this area, and exemplars of initial successes of deployed systems. In Chap. 3, Bilar et al. present a detailed study of the coevolution of the Conficker worm and associated defenses against it and a quantitative model for explaining the coevolution. This study demonstrates, in a concrete manner, that attackers and defenders present moving targets to each other since advances by one side are countered by the other. In Chap. 4, Gonzalez summarizes the current state of computational models of human behavior at the individual level, and it describes the challenges and potentials for extending them to address predictions in 2-player (i.e., defender and attacker) noncooperative dynamic cyber security situations.

The next two chapters explore cyber maneuver in network contexts. In Chap. 5, Torrieri et al. identify the research issues and challenges from jamming and other attacks by external sources and insiders. They propose a general framework based on the notion of maneuver keys as spread-spectrum keys; these supplement higher-level network cryptographic keys and provide the means to resist and respond to external and insider attacks. In Chap. 6, Yackoski et al. describe an IPv6-based network architecture that incorporates cryptographically strong dynamics to limit an attacker's ability to plan, spread, and communicate within the network.

The remaining chapters present MTD approaches based on software transformations. In Chap. 7, Le Goues et al. describe the Helix Metamorphic Shield that continuously shifts a program's attack surface in both the spatial and temporal dimensions and reduces the program's attack surface by applying novel evolutionary algorithms to automatically repair vulnerabilities. The interplay between shifting the attack surface and reducing it results in the automated evolution of new program variants whose quality improves over time. In Chap. 8, Jackson et al. review their automated compiler-based code diversification technique, present an in-depth performance analysis of the technique, and demonstrate its real-world applicability by diversifying a full system stack. Finally, in Chap. 9, Pappas et al. describe in-place code randomization, a software diversification technique that can be applied directly on third-party software. They demonstrate how in-place code randomization can harden inherently vulnerable Windows 7 applications and provide probabilistic protection against return-oriented programming (ROP) attacks.

Contents

Chapter 1
Game Theoretic Approaches to Attack Surface Shifting

Pratyusa K. Manadhata

Abstract A software system's attack surface is the set of ways in which the system can be attacked. In our prior work, we introduced an attack surface measurement and reduction method to mitigate a software system's security risk (Manadhata, An attack surface metric, Ph.D. thesis, Carnegie Mellon University, 2008; Manadhata and Wing, IEEE Trans. Softw. Eng. 37:371–386, 2011). In this paper, we explore the use of *attack surface shifting* in the *moving target defense* approach. We formalize the notion of shifting the attack surface and introduce a method to quantify the shift. We cast the moving target defense approach as a security-usability trade-off and introduce a two-player stochastic game model to determine an optimal moving target defense strategy. A system's defender can use our game theoretic approach to optimally shift and reduce the system's attack surface.

1.1 Introduction

In our prior work, we formalized the notion of a software system's *attack surface* and proposed to use a system's attack surface measurement as an indicator of the system's security [5, 6]. Intuitively, a system's attack surface is the set of ways in which an adversary can enter the system and potentially cause damage. Hence the larger the attack surface, the more insecure the system; we can mitigate a system's security risk by reducing the system's attack surface. We also introduced an *attack surface metric* to measure a system's attack surface in a systematic manner.

Our prior work focused on the uses of attack surface measurements in the software development process. We introduced an *attack surface reduction* approach that complements the software industry's traditional code quality improvement approach to mitigate security risk. The code quality improvement effort aims toward

P.K. Manadhata (✉)
HP Labs, #301, 5 Vaughn Dr, Princeton, NJ 08854, USA
e-mail: manadhata@cmu.edu

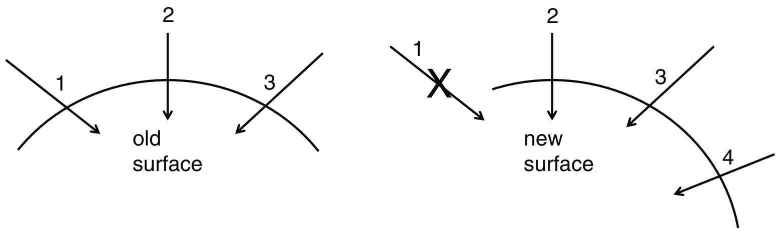

Fig. 1.1 If we shift a system's attack surface, then attacks that worked in the past, e.g., attack 1, may not work any more. The shifting, however, may enable new attacks, e.g. attack 4, on the system

reducing the number of security vulnerabilities in software. In practice, however, building large and complex software devoid of security vulnerabilities remains a very difficult task. Software vendors have to embrace the hard fact that their software will ship with both known and future vulnerabilities in them and many of those vulnerabilities will be discovered and exploited. They can, however, minimize the risk associated with the exploitation of these vulnerabilities by reducing their software's attack surfaces. A smaller attack surface makes the vulnerabilities' exploitation harder and lowers the damage of exploitation, and hence mitigates the security risk.

In this paper, we focus on the uses of attack surface measurements in the context of *moving target defense*. We consider a scenario where system defenders, e.g., system administrators, are continuously trying to protect their systems from attackers. Moving target defense is a novel protection mechanism where the defenders continuously *shift* their systems' attack surfaces to increase the attacker's effort in exploiting their systems' vulnerabilities [1]. As shown in Fig. 1.1, if a defender shifts a system's attack surface, then old attacks that worked in the past, e.g., attack 1, may not work any more. Hence the attacker has to spent more effort to make past attacks work or find new attacks, e.g., attack 4. We view the interaction between a defender and an attacker as a two-player game and hence explore the use of game theory in shifting the attack surface.

The rest of the paper is organized as follows. We briefly discuss our attack surface measurement approach in Sect. 1.2. In Sect. 1.3, we formalize the notion of shifting the attack surface and discuss the uses of attack surface shifting in moving target defense. In Sect. 1.4, we explore game theoretic approaches to attack surface shifting to achieve an optimal balance between security and usability. We conclude with a summary in Sect. 1.5.

1.2 Attack Surface Measurement

We know from the past that many attacks, e.g., exploiting a buffer overflow, on a system take place by sending data from the system's operating environment into the system. Similarly, many other attacks, e.g., symlink attacks, on a system take place because the system sends data into its environment. In both these types of attacks, an attacker connects to a system using the system's *channels* (e.g., sockets), invokes the system's *methods* (e.g., API), and sends *data items* (e.g., input strings) into the system or receives data items from the system. An attacker can also send (receive) data indirectly into (from) a system by using shared persistent data items (e.g., files). Hence an attacker uses a system's methods, channels, and data items present in the system's environment to attack the system. We collectively refer to a system's methods, channels, and data items as the system's *resources* and thus define a system's attack surface in terms of the system's resources.

1.2.1 Attack Surface Definition

Not all resources, however, are part of the attack surface. A resource is part of the attack surface if an attacker can use the resource in attacks on the system. We introduced the *entry point and exit point framework* to identify these relevant resources.

1.2.1.1 Entry Points

A system's codebase has a set of methods, e.g., the system's API. A method receives arguments as input and returns results as output. A system's methods that receive data items from the system's environment are the system's entry points. For example, a method that receives input from a user or a method that reads a configuration file is an entry point. A method m of a system s is a *direct entry point* if either (a) a user or a system in s's environment invokes m and passes data items as input to m, or (b) m reads from a persistent data item, or (c) m invokes the API of a system in s's environment and receives data items as the result returned. An *indirect entry point* is a method that receives data items from a direct entry point.

1.2.1.2 Exit Points

A system's methods that send data items to the system's environment are the system's exit points. For example, a method that writes to a log file is an exit point. A method m of a system s is a *direct exit point* if either (a) a user or a system in s's environment invokes m and receives data items as results returned from m, or (b) m writes to a persistent data item, or (c) m invokes the API of a system in s's environment and passes data items as input to the API. An *indirect exit point* is a method that sends data to a direct exit point.

1.2.1.3 Channels

Each system also has a set of channels; the channels are the means by which users or other systems in the environment communicate with the system, e.g., TCP/UDP sockets, RPC end points, and named pipes. An attacker uses a system's channels to connect to the system and invoke the system's methods. Hence the channels act as another basis for attacks on the system.

1.2.1.4 Untrusted Data Items

An attacker uses persistent data items either to send data indirectly into the system or to receive data indirectly from the system. Examples of persistent data items are files, cookies, database records, and registry entries. A system might read from a file after an attacker writes into the file. Similarly, the attacker might read from a file after the system writes into the file. Hence the persistent data items act as another basis for attacks on a system.

1.2.1.5 Attack Surface Definition

A system's attack surface is the subset of the system's resources that an attacker can use to attack the system. By definition, an attacker can use the set, M, of entry points and exit points, the set, C, of channels, and the set, I, of untrusted data items to send (receive) data into (from) the system to attack the system. Hence M, C, and I are the relevant subset of resources that are part of the attack surface and given a system, s, and its environment, we define s's attack surface as the triple, $\langle M, C, I \rangle$.

1.2.2 Attack Surface Measurement Method

A naive way of measuring a system's attack surface is to count the number of resources that contribute to the attack surface. This naive method that gives equal weight to all resources is misleading since all resources are not equally likely to be used by an attacker. We estimate a resource's contribution to a system's attack surface as a *damage potential-effort ratio* where *damage potential* is the level of harm the attacker can cause to the system in using the resource in an attack and *effort* is the amount of work done by the attacker to acquire the necessary access rights to be able to use the resource in an attack.

In practice, we estimate a resource's damage potential and effort in terms of the resource's attributes. For example, we estimate a method's damage potential in terms of the method's *privilege*. An attacker gains the same privilege as a method by using a method in an attack, e.g., the attacker gains `root` privilege by exploiting a buffer overflow in a method running as `root`. The attacker can cause damage to the system after gaining `root` privilege. The attacker uses a system's channels to connect to a system and send (receive) data to (from) a system. A channel's *protocol* imposes restrictions on the data exchange allowed using the channel, e.g., a `TCP socket` allows raw bytes to be exchanged whereas an `RPC endpoint` does not. Hence we estimate a channel's damage potential in terms of the channel's protocol. The attacker uses persistent data items to send (receive) data indirectly into (from) a system. A persistent data item's *type* imposes restrictions on the data exchange, e.g., a `file` can contain executable code whereas a `registry entry` can not. The attacker can send executable code into the system by using a `file` in an attack, but the attacker can not do the same using a `registry entry`. Hence we estimate a data item's damage potential in terms of the data item's type. The attacker can use a resource in an attack if the attacker has the required *access rights*. The attacker spends effort to acquire these access rights. Hence for the three kinds of resources, i.e., method, channel, and data, we estimate attacker effort to use a resource in an attack in terms of the resource's access rights.

We assume a function, *der*, that maps a resource to its damage potential-effort ratio. In practice, however, we assign numeric values to a resource's attributes to compute the ratio, e.g., we compute a method's damage potential-effort ratio from the numeric values assigned to the method's privilege and access rights. We impose a total order among the values of the attributes and assign numeric values according to the total order. For example, we assume that an attacker can cause more damage to a system by using a method running with `root` privilege than a method running with `non-root` privilege; hence we assign a higher number to the `root` privilege level than the `non-root` privilege level. The exact choice of numeric values is subjective and depends on a system and its environment.

We quantify a system's attack surface measurement along three dimensions: methods, channels, and data. We estimate the total contribution of the methods, the total contribution of the channels, and the total contribution of the data items to the attack surface. Given the attack surface, $\langle M,C,I\rangle$, of a system, s, s's attack surface measurement is the triple $\langle \sum_{m\in M} der(m),\ \sum_{c\in C} der(c),\ \sum_{d\in I} der(d)\rangle$.

1.3 Moving Target Defense

In this section, we discuss the uses of attack surface measurements in moving target defense. Moving target defense is a protection approach where a system's defender continuously *shifts* the system's attack surface. Intuitively, the defender may *modify* the attack surface by changing the resources that are part of the attack surface and/or by modifying the contributions of the resources. Not all modifications, however, shift the attack surface. The defender shifts the attack surface by removing at least one resource from the attack surface and/or by reducing at least one resource's damage potential-effort ratio. Everything else being equal, attacks that worked in the past may not work in the future if the attacks depended on the removed (modified) resource. The shifting process, however, might have enabled new attacks on the system by adding new resources to the attack surface. Hence the attacker has to spend more effort to make past attacks work or to identify new attacks.

1.3.1 Shifting the Attack Surface

We formalize the notion of *shifting the attack surface* in this section and introduce a method to quantify the shift. We introduced an I/O automata model of a system and its environment in our prior work; we use the model to define and quantify the shift in the attack surface.

Consider a set, S, of systems, an attacker, U, and a data store, D. For a system, $s \in S$, we define s's environment, $E_s = \langle U, D, T \rangle$, to be a three-tuple where $T = S \setminus \{s\}$ is the set of systems excluding s. U represents the adversary who attacks the systems in S. The data store D allows data sharing among the systems in S and U.

We model a system and the entities present in its environment as I/O automata [4]. An I/O automaton, $A = \langle sig(A), states(A), start(A), steps(A) \rangle$, is a four tuple consisting of an *action signature*, $sig(A)$, that partitions a set, $acts(A)$, of *actions* into three disjoint sets, $in(A)$, $out(A)$, and $int(A)$, of *input*, *output* and *internal* actions, respectively, a set, $states(A)$, of *states*, a non-empty set, $start(A) \subseteq states(A)$, of *start states*, and a *transition relation*, $steps(A) \subseteq states(A) \times acts(A) \times states(A)$. An *execution* of A is an alternating sequence of actions and states beginning with a start state and a *schedule* of an execution is a subsequence of the execution consisting only of the actions appearing in the execution.

Given a system, s, and its environment, E, s's attack surface is the triple, $\langle M, C, I \rangle$, where M is the set of entry points and exit points, C is the set of channels, and I is the set of untrusted data items of s. We denote the set of resources belonging to s's attack surface as $R_s = M \cup C \cup I$. Also, given two resources, r_1 and r_2, of s, we write $r_1 \succ r_2$ to express that r_1 makes a larger contribution to the attack surface than r_2. If we modify s's attack surface, R_o, to obtain a new attack surface, R_n, then we denote a resource, r's, contributions to R_o as r_o and to R_n as r_n. We define attack surface shifting qualitatively as follows.

Definition 1.1. Given a system, s, its environment, E, s's old attack surface, R_o, and s's new attack surface, R_n, s's attack surface has shifted if there exists at least one resource, r, such that (i) $r \in (R_o \setminus R_n)$ or (ii) $(r \in R_o \cap R_n) \wedge (r_o \succ r_n)$.

If we shift s's attack surface, then attacks that worked on s's old attack surface may not work on s's new attack surface. We model s's interaction with its environment as parallel composition, $s||E$, in our I/O automata model. Since an attacker attacks a system either by sending data into the system or by receiving data from the system, any schedule of the composition $s||E$ that contains s's input actions or output actions is a potential attack on s. We denote the set of potential attacks on s as $attacks(s,R)$ where R is s's attack surface. In our I/O automata model, if we shift s's attack surface from R_o to R_n, then with respect to the same attacker

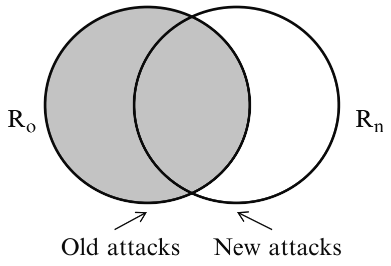

Fig. 1.2 If we shift a system's attack surface from R_o to R_n, then at least one attack that worked on R_o will not work any more on R_n

and the environment, a few potential attacks on R_o will cease be potential attacks on R_n. Intuitively, if we remove a resource, r, from the attack surface or reduce r's contribution to the attack surface during shifting, then executions of s that contain r will not be executions in the new attack surface. Hence the schedules derived from these executions will not be potential attacks on s in the new attack surface (Fig. 1.2).

Theorem 1.1. *Given a system, s, and its environment, E, if we shift s's attack surface, R_o, to a new attack surface, R_n, then* $attacks(s,R_o) \setminus attacks(s,R_n) \neq \emptyset$.

Proof (Sketch). If we shift s's attack surface from R_o to R_n, then from Definition 1.1, there is at least one resource, r, such that either (i) $r \in (R_o \setminus R_n)$ or (ii) $(r \in R_o \cap R_n) \wedge (r_o \succ r_n)$.

If $r \in (R_o \setminus R_n)$, then without loss of generality, we assume that $R_o = R_n \cup \{r\}$. Since $r \in R_o \wedge r \notin R_n$, following arguments similar to the proof of Theorem 1 in [5], there exists a method, m, such that $m \in R_o \wedge m \notin R_n$. Hence there exists a schedule, β, of the composition $s_{R_o}||E$ containing m such that β is not a schedule of the composition $s_{R_n}||E$. Hence $\beta \in attacks(s,R_o) \wedge \beta \notin attacks(s,R_n)$, and $attacks(s,R_o) \setminus attacks(s,R_n) \neq \emptyset$.

Similarly, if $(r \in R_o \cap R_n) \wedge (r_o \succ r_n)$, then r makes a larger contribution to R_o than R_n. Following arguments similar to the proof of Theorem 3 in [5], there exists a method, $m \in R_o \cap R_n$, such that m has a stronger pre condition and/or a weaker post condition in R_o than R_n. Hence there exists a schedule, β, of the composition $s_{R_o}||E$ containing m such that β is not a schedule of the composition $s_{R_n}||E$. Hence $\beta \in attacks(s,R_o) \wedge \beta \notin attacks(s,R_n)$, and $attacks(s,R_o) \setminus attacks(s,R_n) \neq \emptyset$. □

We introduced a qualitative notion of shifting the attack surface in previous paragraphs. We quantify the shift in the attack surface as follows.

Definition 1.2. Given a system, s, its environment, E, s's old attack surface, R_o, and s's new attack surface, R_n, the shift, ΔAS, in s's attack surface is

$$|R_o \setminus R_n| + |\{r : (r \in R_o \cap R_n) \wedge (r_o \succ r_n)\}|$$

In Definition 1.2, the term $|R_o \setminus R_n|$ represents the number of resources that were part of s's old attack surface, but were removed from s's new attack surface. Similarly, the term

$$|\{r : (r \in R_o \cap R_n) \wedge (r_o \succ r_n)\}|$$

represents the number of resources that make larger contributions to s's new attack surface than the old attack surface. If $\Delta AS > 0$, then we say that s's attack surface has shifted from R_o to R_n.

Our definition assumes that all resources contribute equally to the shift in the attack surface. We may be able to quantify the shift better by considering the resources' attributes, e.g., a resource's damage potential-effort ratio. We leave such quantification approaches as future work.

1.3.2 Ways to Shift the Attack Surface

The defender may modify the attack surface in three different ways. But only two of these three ways shift the attack surface. First, the defender may shift the attack surface and also reduce the attack surface measurement by disabling and/or modifying the system's *features* (Scenario A). Disabling the features reduces the number of entry points, exit points, channels, and data items, and hence reduces the number of resources that are part of the attack surface. Modifying the features reduces the damage potential-effort ratios of the resources that are part of the attack surface, e.g., lowering a method's privilege or increasing the method's access rights, and hence reduces the resources' contributions to the attack surface measurement.

Second, the defender may shift the attack surface by enabling new features and disabling existing features. Disabling the features removes resources from the attack surface and hence shifts the attack surface. The attack surface measurement, however, may decrease (Scenario B), remain the same (Scenario C), or increase (Scenario D). The enabled features increase the attack surface measurement by adding more resources to the attack surface and the disabled features decrease the measurement by removing resources from the attack surface; the overall change in the measurement may be negative, zero, or positive. Similarly, the defender may shift the attack surface by modifying existing features such that the damage potential-effort ratios of a set of resources decrease and the ratios of another set of resources increase. The attack surface measurement may decrease, remain the same, or increase.

Third, the defender may modify the attack surface by enabling new features. The new features add new resources to the attack surface and hence increase the attack surface measurement. The attack surface, however, doesn't shift since the old attack surface still exists and all attacks that worked in the past will still work (Scenario E). The defender may also increase the attack surface measurement without shifting the attack surface by increasing the damage potential-effort ratios of existing resources. We summarize the scenarios in Table 1.1.

From a protection standpoint, the defender's preference over the scenarios is the following: A $>$ B $>$ C $>$ D $>$ E. Scenario A is preferred over scenario B because scenario B adds new resources to the attack surface and the new resources may enable new attacks on the system. Scenario D increases the attack surface measurement; but it may be attractive in moving target defense, especially if the increase in the measurement is low and the shift in the attack surface is large.