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

S. Jajodia et al. (eds.), *Moving Target Defense II: Application of Game Theory and Adversarial Modeling*, Advances in Information Security 100,
DOI 10.1007/978-1-4614-5416-8_1,

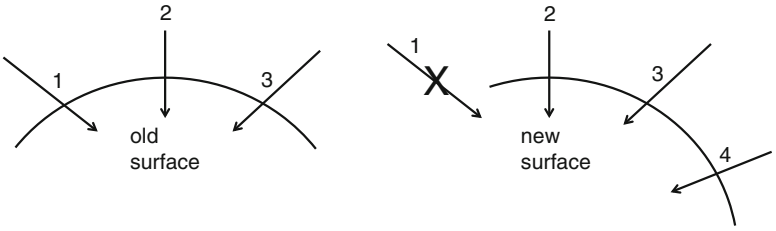

Fig. 1.1 If we shift a system's attack surface, then attacks that worked in the past, e.g., attack 1, may not work any more. The shifting, however, may enable new attacks, e.g. attack 4, on the system

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

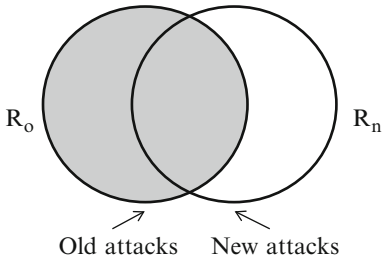

Fig. 1.2 If we shift a system's attack surface from R_o to R_n, then at least one attack that worked on R_o will not work any more on R_n

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.

Table 1.1 Different scenarios to modify and shift the attack surface. Not all modifications shift the attack surface

Scenario	Features	Attack surface shift	Attack surface measurement
A	Disabled	Yes	Decrease
B	Enabled and disabled	Yes	Decrease
C	Enabled and disabled	Yes	No change
D	Enabled and disabled	Yes	Increase
E	Enabled	No	Increase

1.3.3 A Security-Usability Trade-off

The defender may not always be able to shift and reduce the attack surface. The defender may have to enable new features or modify existing features to provide desirable services to the system's users at the expense of an increased attack surface measurement. For example, the users may demand remote access to the system and hence defender may have to open a new communication channel, e.g., a TCP port, to satisfy the demand. The increased attack surface may enable new attacks on the system. Similarly, attack surface reduction comes at a cost; the reduction process disables or modifies the system's features and hence the system may not be able to provide certain services as before. Hence the defender has to make a classic security-usability trade-off as part of the moving target defense. In Sect. 1.4.1, we discuss game theoretic approaches to determine optimal ways to modify and shift the attack surface to achieve moving target defense.

1.4 Game Theoretic Approaches

In our prior work, we viewed attack surface measurement and reduction as a "static" process, i.e., software developers measure their systems' attack surfaces without making any assumptions about an attacker (e.g., the attacker's resources, skill levels, and behavior) and then try to reduce the attack surface as much as possible. In this section, we consider attack surface reduction and shifting in a dynamic moving target defense environment - a defender continuously tries to protect a system from an attacker by reducing and shifting the attack surface. We model the interaction between the defender and the attacker as a two player game and use game theory to determine *optimal* defense strategies. The game theoretic model allows us to explicitly model the attacker. Hence the defender can choose optimal defense strategies for different attacker profiles such as script kiddies, experienced hackers, organized criminals, and nation states.

1.4.1 Optimal Moving Target Defense

We model the interaction between a system's defender and an attacker as a two-player stochastic extensive game [8].

1.4.1.1 A Game Model

Our stochastic game model is similar to the model introduced by Lye and Wing [3]. Our model is a 7 tuple, $\langle S, A^d, A^a, T, R^d, R^a, \beta \rangle$, where S is a set of system states, A^d is the defender's action set, A^a is the attacker's action set, $T : S \times A^d \times A^a \times S \rightarrow [0,1]$ is the state transition function, $R^d : S \times A^d \times A^a \rightarrow \mathbb{R}$ is the defender's reward function where $\mathbb{R}$ is the set of real numbers, $R^a : S \times A^d \times A^a \rightarrow \mathbb{R}$ is the attacker's reward function, and $\beta \leq 1$ is a discount factor for discounting future rewards.

The defender and the attacker play the game in the following manner. The system is in the state $s_t \in S$ at time t. The defender performs an action, $a^d \in A^d$, and the attacker performs an action, $a^a \in A^a$. The system then moves to a state, $s_{t+1} \in S$, with probability $T(s_t, a^d, a^a, s_{t+1})$. The defender's reward for performing the action is $r_t^a = R^d(s_t, a^d, a^a)$ and the attacker's reward is $r_t^d = R^a(s_t, a^d, a^a)$. The goals of the defender and the attacker are to maximize their discounted rewards.

1.4.1.2 States, Actions, and Transitions

We model the system as a set, F, of features. F represents the system's features that provide various functionality, e.g., a web server's login feature provides user authentication functionality. A feature can be disabled or if enabled, can be in one of several *configurations*; each configuration is a mapping of state variables to their values. A state, $st \in S$, of the system is a mapping of the features to their configurations, i.e., $st : F \rightarrow Configuration$. At a given system state, the defender may choose to shift and reduce the attack surface by *acting* on the features, e.g., the defender may enable a disabled feature, disable an enabled feature, modify an enabled feature's configuration, or leave a feature's configuration unchanged. Hence, a defender action, $a^d \in A^d$, is a mapping of the features to the actions performed by the defender on the features, i.e., $a^d : F \rightarrow \{enabled, disabled, modified, unchanged\}$. In a given system state, the defender can choose from a subset of the set of actions A^d. For example, if a feature, f, is in enabled configuration in a state, s, then the defender cannot perform any action that *enables* f; the defender can only disable f, modify f, or leave f unchanged. After the defender performs an action, the system's attack surface changes. The attacker then performs an action to attack the system by utilizing the change in the attack surface; the attacker's action may further enable and disable the system's features. Since the defender's action and the attacker's action enable and/or disable certain system features, the system moves to a new state according to the probabilistic transition function. The specific values of the transition probabilities depend on the system and its operating environment.

Potential state space explosion and potential action space explosion are two disadvantages of our model. The number of states and the number of actions are exponential in the number of features. For simplicity and tractability, we may focus on an important subset of the system's features and may bound the number of features that an action can enable, disable, or modify.

1.4.1.3 Reward Functions

When the defender performs an action, the action may benefit the defender in three ways. First, the defender may provide value to the system's users by enabling features. Second, the defender may mitigate the system's security risk by shifting the attack surface. Third, the defender may mitigate the system's security risk by reducing the attack surface measurement. The action, however, may cost the defender if it disables features or increases the system's attack surface measurement. Hence the defender's reward depends on the change in value derived from the features, the shift in the attack surface, and the change in the attack surface measurement.

Similarly, when the attacker performs an action, the attacker benefits from the increase in the attack surface measurement. But the shift in the attack surface costs the attacker. Hence the attacker's reward depends on the shift in the attack surface and the change in the attack surface measurement.

Consider a state, s, of the system. If the defender performs an action, a^d, in a state, s, and the attacker performs an action, a^a, then we denote the change in the system's features as ΔF, the shift in the attack surface as ΔAS, and the change in the attack surface measurement as ΔASM. Then we model the defender's reward, R^d, and the attacker's reward, R^a, as follows.

$$R^d(s, a^d, a^a) = B_1(\Delta F) + B_2(\Delta AS) - C_1(\Delta ASM)$$
$$R^a(s, a^d, a^a) = B_3(\Delta ASM) - C_2(\Delta AS)$$

B_is and C_is are functions that map the changes in features, attack surface shift, and attack surface measurement to real numbers; the numbers reflect the benefits and costs associated with the changes. The exact choice of B_is and C_is depends on the system and its operating environment. Please note that our choice of reward functions makes the game a general-sum game.

1.4.1.4 Optimal Defense Strategies

We model our game as a complete and perfect information game; each player knows the other player's strategies and pay-offs, and is aware of the game *history*, i.e., the actions already performed by both players in the game [9]. The goal of each player is to maximize their expected discounted pay-off.

A player's *strategy* is a plan of action that the player can take in the game; the strategy specifies the action(s), given the other player's strategy, the player can take at different stages of the game. An optimal strategy maximizes the player's expected pay-off.

A stationary strategy is a strategy that is independent of time and history, and depends only on the system's state. A pure strategy specifies a single action in a state whereas a mixed strategy specifies a probability distribution over possible actions in the state. We use the Nash Equilibrium solution concept to determine the defender's optimal stationary strategy. Filar and Vrieze introduced a non-linear program to find stationary equilibrium strategies in general sum stochastic games [2].

The defender may want an optimal strategy that is dependent on time and history. The strategy specifies optimal defender action(s) after *every* history of the game. Hence the defender can take an optimal action in response to the attacker's action at any time in the game. We use the subgame perfect equilibrium concept to determine the defender's optimal strategy [8]. Murray and Gordon introduced a dynamic programming algorithm to find a subgame perfect equilibrium in general sum stochastic games [7].

Hence the defender can use the concepts of Nash equilibrium and subgame perfect equilibrium to determine optimal strategies for shifting and reducing the attack surface. The optimal strategies enable the defender to make the security-usability trade-off in an informed manner; the system can then provide required services to its users without compromising its security.

1.5 Summary

In summary, we introduced a game theoretic attack surface shifting and reduction approach to achieve moving target defense. System defenders can use our approach to determine their best course of action to protect their systems while providing required services to their systems' users. In the future, we plan to instantiate our model on real world software systems and explore the efficacy of our approach in real world settings.

References

1. National cyber leap year summit 2009 co-chairs report. http://www.cyber.st.dhs.gov/docs/National_Cyber_Leap_Year_Summit_2009_Co-Chairs_Report.pdf (2009)
2. Filar, J., Vrieze, K.: Competitive Markov decision processes. Springer (1997)
3. Lye, K., Wing, J.M.: Game strategies in network security. International Journal of Information Security pp. 71–86 (2005)
4. Lynch, N., Tuttle, M.: An introduction to input/output automata. CWI-Quarterly **2**(3) (1989)
5. Manadhata, P.K.: An attack surface metric. Ph.D. thesis, Carnegie Mellon University (2008)

6. Manadhata, P.K., Wing, J.M.: An attack surface metric. IEEE Trans. Softw. Eng. **37**, 371–386 (2011). DOI http://dx.doi.org/10.1109/TSE.2010.60. URL http://dx.doi.org/10.1109/TSE.2010.60
7. Murray, C., Gordon, G.: Finding correlated equilibria in general sum stochastic games. Tech. Rep. CMU-ML-07-113, Carnegie Mellon University (2007)
8. Osborne, M., Rubinstein, A.: A course in game theory. MIT Press (1994)
9. Roy, S., Ellis, C., Shiva, S., Dasgupta, D., Shandilya, V., Wu, Q.: A survey of game theory as applied to network security. Hawaii International Conference on System Sciences **0**, 1–10 (2010). DOI http://doi.ieeecomputersociety.org/10.1109/HICSS.2010.35