Autonomous Vehicle Security: Composing Attack, Defense, and Policy Surfaces

Michael Clifford
Toyota InfoTech Labs
Mountain View, California, USA
michael.clifford@toyota.com

Matt Bishop
Department of Computer Science
University of California at Davis
Davis, California, USA
bishop@cs.ucdavis.edu

Miriam Heller
MHITech Systems
Arlington, VA, USA
heller@mhitech.com

Karl Levitt
Department of Computer Science
University of California at Davis
Davis, California, USA
levitt@cs.ucdavis.edu

ABSTRACT

An *attack surface* enumerates resources accessible to an attacker for cyber attacks on a system. These resources are: methods that can be called as part of an attack; channels that an attacker outside the system can use to get to a system's interface; and untrusted data that an attacker can use in conjunction with the system's programs and channels. Historically, a system's attacks surface has provided a metric on the vulnerability of a system, in part to compare two systems' exposure to attack.

In this paper we extend the attack surface to (1) include rules on the system's methods and channels that if enforced would prevent many attacks, and (2) be a composition of more primitive surfaces each characterizing vulnerabilities associated with types of resources, application-specific or system-specific, e.g., files, directories, and channels. We also introduce two additional surfaces. The *defense surface* identifies system mechanisms that can thwart cyber-attacks through prevention, or through detection followed by mitigation of an attack in progress and then system restoration. The *policy surface* defines the security policy of a system as reflected by constraints on its interface expected to be satisfied in the system's operation.

The security policy for a corporation would include steps the organization takes to prevent attacks and actions required to address a security incident. More relevant to this paper, the security policy for a community of autonomous vehicles would specify the minimum separation among vehicles that must be maintained even in the presence of a cyber-attack, i.e. a (safety) property. Through an analysis of the intersection of the three surfaces, it is, in principle, possible to determine if a defense exists for every attack that causes a policy violation. And, through computationally-efficient model checking, the defense action can be identified. If more than one defense action exists, model checking will identify all of them, thus enabling the selection of the optimal action based on criteria associated with a CAV.

NSPW '22, October 24–27, 2022, North Conway, NH, USA

ACM ISBN 978-1-4503-9866-4/22/10.
https://doi.org/10.1145/3584318.3584325

CCS CONCEPTS

• **Security and privacy** → **Distributed systems security**; **Vulnerability management**; **Malware and its mitigation**; **Intrusion detection systems**; *Formal security models*; **Authorization**; • **Computer systems organization** → **Embedded systems**; *Redundancy*; Robotics; • **Networks** → Network reliability.

KEYWORDS

Attack surface, defense surface, policy surface, informed defense against cyber attack, security of connected autonomous vehicles

ACM Reference Format:
Michael Clifford, Matt Bishop, Miriam Heller, and Karl Levitt. 2022. Autonomous Vehicle Security: Composing Attack, Defense, and Policy Surfaces. In *New Security Paradigms Workshop (NSPW '22), October 24–27, 2022, North Conway, NH, USA.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3584318.3584325

1 INTRODUCTION

1.1 Background

The Internet has expanded beyond traditional computers and networks to incorporate the rapidly expanding Internet of Things (IoT), as well as cyber-physical systems. IoT devices often have very limited security capabilities and poor security maintenance, making them particularly vulnerable to attack. Cybercrime is estimated to have exceeded $6 trillion by 2021 [41]. Cyber-physical systems operate critical infrastructure, and form the basis for control of autonomous systems such as Connected, Autonomous Vehicles (CAV). Consequently, increasing device connectivity provides not only new entry points into systems and networks, but also the potential for increased risks and costs from attack damage.

Connected, mobile devices reflect this risk. While many such devices are targeted by attackers because of their access to sensitive or personal data, or because of their importance in a wide variety of activities, mobility-focused systems such as CAV are a particularly tempting target. If attackers can manipulate the behavior of CAV,

they can disrupt traffic, cause accidents, and endanger lives [29]. Preventing attacks on CAV is, therefore, critical to the future of autonomous transportation. In the future, we expect that both CAV, and other types of devices, will work together to achieve complex goals. Ensuring that attackers cannot control, manipulate, deceive, or disrupt connected devices is required in order to protect their users, and the larger scale systems that incorporate them.

The increase in attack risk and cost implies the need for improved defensive tools, such as those that provide prevention, detection, and analysis, as well as enhanced attack monitoring and response. Such tools exist in many forms. The MITRE ATT&CK framework [49] provides a standardized categorization of attacks. In 2021, MITRE introduced a complementary knowledge base, D3FEND [25] is a knowledge graph of cybersecurity countermeasures associated with offensive and adversarial techniques. It currently serves to standardize the security vocabulary. One software tool, Bishop-Fox's Cosmos [4], analyzes systems and networks to identify and manage attack surfaces. Another, Penetra, is an automated penetration tool that gives information that can be used to minimize attack surfaces [42]. An attack is defined as "a sequence of actions that create a violation of a security policy" [2, p. 959]. Thus, whether some sequence of actions is an attack depends on the security policy. Unfortunately, these tools do not address policy, and therefore can not identify certain classes of attacks.

A simple example is a buffer overflow attack resulting in root access. Suppose a security policy states that "all input shall be checked for proper length." In that case, *any* input that avoids this check is an attack, as it violates the security policy. But if the policy instead stated "only authorized users shall acquire *root* privileges," then input that causes a buffer overflow may or may not be an attack. Specifically, if the program with the buffer overflow were not setuid to *root*, then the user will not gain any extra privileges, and hence this is not an attack. But if the program were setuid to *root*, then the buffer overflow would enable the user to gain privileges they are not authorized to have — hence, an attack.

This study develops a surface, analogous to an attack surface, but for policy. By examining the intersection of the two surfaces, the attacks of interest can be identified. Then it goes a step further, defining a defense surface that implements defenses as required by the policy surface. In this way, reasoning and identifying attacks takes policy into account. Moreover, the explicit, structured relationship of policies defined in the policy surface, in combination with the attack and defense surfaces, could enable explicit verification and updating.[1]

The next section of the paper reviews the extensive prior work on attack surfaces, attack trees and attack much more on prior work 1) examining the concept of "attack surfaces" in detail and tying them to the Requires/Provides model [50] of attacks; 2) extending the definition of attack surfaces to incorporate attack trees, graphs and paths, formally defining attack, defense and policy surfaces, and analyzing how all three surfaces interact; 3) arguing that these surfaces can be divided into an object-oriented set of sub-surfaces representation, and showing that this simplifies analysis and may reduce attack and defense costs; 4) showing that this representation allows for the use of iterative and recursive surfaces, which simplifies the modeling of attack, defense, and policy interactions at many different levels and scales; 5) applying these new paradigms to the attack and defense of complex systems, such as platoons of CAV.

Following a section that discusses surfaces, the next three sections present attack, defense, and policy surfaces. The next two sections give examples, and the section after presents a representation of an attack on CAVs to show how the surfaces work in practice. The remaining sections cover root cause analysis, cost, strategies, and other real world applications. We conclude with a discussion of the paper's key ideas directions for future work.

[1]While we primarily discuss surfaces in the context of autonomous systems, they can be applied to non-autonomous systems, such as cloud systems or machine learning models as well. We elaborate on attacks on machine learning models in 5.3.11.

2 RELATED WORK

2.1 Attack Trees and Attack/Defense Trees

Attack trees [44] represent the possible ways for an attacker to reach some goal state (at the tree's root node). Each goal has its own attack tree, with the goal state represented by the root node. Leaf nodes represent attacks. Tree branches are combined using Boolean expressions or comparisons. While attack trees model potential ways for an attack to occur, they do not model whether an attacker currently has the ability to carry them out. This idea was extended by Kordy, et al. [28], which introduced the concept of an Attack/Defense Tree (ADT). An ADT allows attack and defense nodes (or subtrees) to be added as countermeasures to each other.

2.2 Fault Tree Analysis

A related approach, Fault Tree Analysis, is summarized in [13, 30]. While attack trees look at how things fail, fault trees model whether or not failures can occur at all.

2.3 Process Analysis

Process analysis [3] has been used to model insider threat attacks. In this approach, process models represent specifications of agents, activities, artifacts, and required model precision. The approach utilizes fault tree analysis, but also adds in Finite-State Verification, which is used to verify whether it is possible for an agent to corrupt a process.

2.4 Capability Paths

The Requires/Provides Model [50] addressed a limitation of attack trees — the inability to model whether an attacker *can* actually carry out an attack. It describes attacks using sequences of capabilities (preconditions that attackers must satisfy in order to carry out attacks) and concepts (Boolean expressions of subtasks that the attacker must complete in order to reach a goal.) These sequences forms paths that the attacker follows from their current state to some goal state.

2.5 Attack Graphs

Attack graphs [46] extend the idea behind the Requires/Provides model by combining multiple attack paths into a graph. This graph accounts for attacks that can be carried out in multiple ways, with each attack path representing a different sequence of atomic attacks that ends in some ultimate objective. This work was extended in

[23], which applied model checking to attack graphs, introduced automated graph generation, and represented attack graphs using preconditions. Additionally, tools for the automatic mapping of attack graphs [22] have been developed. More recently, attack and defense graphs have been used for mapping, simulation, and automated response simulation of network threats.

2.6 Graph Scalability and Performance

Predictive graphs [32, 33] attempt to address the scalability and performance of attack graphs by only adding one instance of each new vulnerability to the attack graph. Improvements to improve scalability and performance using multiple-prerequisite graphs were presented in [21].

2.7 Attack Surfaces

Attack surfaces were described conceptually in [20], and formally in both [35] and in [37] using state-machines and I/O automata [34] respectively, in order to model system state transitions. The formal treatments of both consider a system's attack surface in terms of data, interfaces (such as networks), and methods (such as programming interfaces). In both cases, if a system resource can be used by an attacker to attack a system, then it is part of the system's attack surface [36]. In both the 2004 and 2011 references, a resource's damage potential and effort are considered in terms of the resource's attributes, e.g., "method privilege, access rights, channel protocol, and data item type. Our estimation method is a specific instantiation of our [sic] general measurement framework and includes only technical impact (e.g., privilege elevation)."

Theisen et al. [51] discuss the methods and results of a systematic literature review (SLR) of the term, *attack surface*, in the context of security. A key goal of the study was to motivate standardizing the definition of *attack surface*. The SLR first screened the titles and abstracts of 1433 papers, reducing the set of papers to 644. These papers were categorized: 7% provided their own definition; 13% referred to another paper's definition; 71% used the term without definition or reference; 9% were not relevant. While 55% of these papers were theoretical, the remaining 45% percent were categorized as: 23% pertaining to systems, e.g., operating systems, 8% pertaining to networks, 4% corresponding to functions, methods, or operations within the code, and 2% were associated with binary files. The remainder were not categorized. Theisen et al. recommend that researchers and practitioners use one of six themes identified in the analysis: "*Reachable Vulnerabilities*: The attack surface is the vulnerabilities that are exposed to the end users via paths or flows, rather than the paths or flows themselves" [51]. Separately, forty-eight different definitions for *attack surface* were identified by Manadhata and Wing [36]. This paper's working definition was cited the most, at 43 times.

Kim et al. [26] provides a review of 151 papers addressing autonomous vehicle cybersecurity. Several identify or refer to *attack surfaces*. Miller and Valasek (2014) provided a survey of remote attack surfaces on various vehicles based on their internal network architectures. They identified physically controllable features and noted defense strategies were needed, such as the detection of message injection.

The local and remote attack surfaces for vehicles, and the software that they interact with, has been another area of prior work. Li et al [31] looked at an infotainment system attack and included a description of the attack surface of modern vehicles. Eriksson et al. [14] investigated the attack surface and vulnerability of in-vehicle Android apps and suggested countermeasures for fine-grained permission, API control, system support, and information flow. Checkoway et al. [7] claim to be the first to apply the notion of attack surfaces from outside rather than inside cars, and include an experimental demonstration of these attack surfaces. Foster et al. [15] described the local and remote attack surfaces of the car's telematics control unit. The local attack surface was also explained for web, telnet console access, NAND dump, and SSH service methods.

2.8 CAV Attack Surfaces

Finding the attack surface of a Connected, Autonomous Vehicle (CAV) is critical, as it allows attackers to find feasible points of attack for the vehicle, and defenders to determine both how to allocate defensive resources, and how to block potential attacks.[2] Maple, et al. [39] presents a reference architecture for a CAV, draws on it to model an attack surface, and explores attack graphs that incorporate specific points of entry within the attack surface. However, this work does not address either attacker goals, or defender defenses and policies.

2.9 Relationships Between Entities and Properties

The properties of the entities participating in an attack (the attackers, from the defender's perspective, and vice-versa) are often uncertain. Neither party normally has complete information about the other. This is especially true when the attackers and targets in an attack may both be unknown, as in the case of fleets of connected, autonomous vehicles. Probabilistic Relational Models (PRMs) attempt to address this uncertainty by reasoning about the probabilistic relationships between entities and their properties. [17, 27]

2.10 Path Discovery

Finally, both attackers and defenders need to be able to determine the set of possible state changes that are reachable from either their current states, or from that of their adversary. Attackers need to know what attacks are possible, and what defenders could do to counter their moves. Likewise, defenders need to predict potential attacker moves, as well as what responses they could take to counter them. Both need to be able to do so efficiently. The Solar Trust Model [10–12] computes trust relations between entities along dynamically generated paths using a Path Discovery Algorithm.

3 THE SURFACES

A "surface" is an abstraction of a system that represents some properties of that system. For a system composed of a single program, one surface of interest is a specification of the program's functional

[2]Note that the "defender" of a CAV is likely to be an autonomy stack, or other software, that makes decisions for the vehicle, rather than a human driver. In some cases, that software may be remote from the vehicle.

behavior that hides details of the implementation. A different surface would represent the system's performance under different conditions. For a system claimed to be tolerant to independently failing components, a surface could represent the system's ability to perform as components fail and the failures are addressed [53]. A common approach to representing a surface is as a state machine, possibly defined by a set of constraints induced by the properties of interest.

Three surfaces represent the security of a system: (1) the attacks for which the system is vulnerable (the *attack surface*), (2) the defenses that can be deployed to counter an attack (the *defense surface*), and (3) the policy that defines the security of the system (the *policy surface*).

A brief description of the relationships of the surfaces will orient the reader to what follows.

- The attack surface is composed of points that an attacker can use to attempt to compromise the system. Put another way, in order for a successful attack to occur, the attacker must find points that are in, or can be transitioned (perhaps through an exploit) to, a state in which the policy constraints are not satisfied. This ties attack surfaces to attack trees. Attack trees encapsulate the steps needed to find and exploit the vulnerabilities that lead to a failure to meet policy constraints.
- The policy surface is the set of points of the system, and constraints upon those points that define "security".[3] The points have attributes such as permissions and policy constraints that specify the allowed settings of these points. If any point is set in such a way as to not satisfy those constraints, the system is not secure.
- The defense surface consists of the points and their settings that satisfy the policy constraints, and the points that detect unauthorized changes to the point settings or the points themselves, and that reset settings back to values allowed by the policy surface constraints. These points are a (possibly improper) subset of the points composing the policy surface. Ideally, the attack surface points will be a subset of these also, so the attacks cannot be realized. In practice, the defense surface points are a proper subset of those of the policy surface, and hence the system is vulnerable to attack.

We now formally define the surfaces, as well as how they interact.

3.1 Attack Surfaces

As an attack surface is tied to points needed to reach a goal, the points' use must violate the security policy. We augment the definition of an attack surface [36] with a set R of rules that describe the actions in which an attacker could use those resources to violate the policy, and how the attacker would do so. The security policy is created based on a threat model, because the policy must address the threats. So the threat model defines the rules R, and the resources necessary to instantiate those rules. Hence it also describes the attack surface

[3] A subset of these constraints may define the system boundary. Policies may also specify contexts, defining conditions under which specific constraints do, or do not, apply.

Definition. Given a system with environment E, the system's attack surface is the tuple (M^E, C^E, I^E, R^E) where M^E is the set of entry and exit points in the system, C^E is the set of channels of the system, I^E is the set of untrusted data items in the system, and R^E is a set of rules that an attacker can apply to the resources (M^E, C^E, and I^E) to launch or further an attack.

Intuitively, the attack surface is a set of points (resources) that can be used to (attempt to) compromise the system and the rules an attacker must apply to do so. An attack tree (vector) describes the steps needed to carry out such a compromise. The rules are what the attacker applies to take steps. The rules and resources are time dependent, for example when the attacker is launching a race condition attack. For simplicity, we consider time a part of the environment E.

Example 1. Consider a network port, which is both an entry and exit point. The rule is "sweep ports". An attacker does a port sweep. Thus, the attacker is applying a rule (sweep ports) to a point (the network port).

Example 2. A platoon of CAVs has a large attack surface, as each car can be attacked in such a way as to disrupt the platoon, and the platoon itself can be attacked. Here, the attacker determines the frequency of the communications among the cars (and, possibly, any other entities that guide the platoon) and masquerades as one of the cars. They then send bogus information to the platoon, indicating that there are obstructions ahead, causing the platoon to stop. Here, the entry point is the CAV's receiver controlling the position, speed, and direction of the car; the untrusted data items are the messages, which are neither digitally signed nor authenticated; and the rule being applied is to force perception of an illusion.

3.1.1 Describing an Attack Surface Using Preconditions. An attack surface can be described using sets of preconditions, state transitions, and post-conditions. The system has some current state σ. A precondition represents the conditions that must be satisfied in order for σ to transition to some state σ'. If the precondition is authorized to be satisfied, then no violation of policy occurs. However, if the precondition is satisfied without authorization by policy, then we can say that an attack has occurred.

A precondition can be represented by a pair (σ, f), where σ is the required state of the system and f is a description of the conditions that are to be satisfied. Considering this description to be written as a set of logic formulae provides one approach to analysis. Suppose a precondition p_1 requires that 3 specific conditions in an environment must be true; call these c_{11}, c_{12}, and c_{13}. Then $p_1(f) = c_{11} \wedge c_{12} \wedge c_{13}$. If preconditions p_1 or p_2 must hold for an attacker to successfully penetrate the attack surface AS, then $AS = p_1 \vee p_2 = (\sigma, (c_{11} \wedge c_{12} \wedge c_{13}) \vee (c_{21} \wedge c_{22} \wedge c_{23}))$. This reduces to a satisfiability problem.

The set of resources in the attack surface are included as part of the conditions. A typical condition will define the resource(s) and state components that are required to exercise a capability. Thus, intuitively, a precondition is the conjunction of conditions enabling the attacker to advance towards a goal. This also implies that an attack tree can be embedded in the augmented attack surface.

Additionally, in practice, postconditions may not hold even though the corresponding preconditions hold. The reason is that

the attack may disrupt or compromise the mechanism that causes the postconditions to hold. To detect this, an intrusion detection system can analyze the variables in the postcondition that define the state. In what follows, we consider this as a part of the rules, and hence do not distinguish it from other rules.

Attack surfaces can be dynamic. For example, a path might work from 8 AM until 5 PM each day (as dictated by a system policy), or until blocked (by making some required capabilities unreachable). An attack surface could also be spatially dynamic, with certain attacks only becoming possible when the target system, or certain capabilities, are located within a certain set of locations. Spatial and temporal attack surface dynamics are of particular importance for autonomous vehicles, since they are subject to attacks that are only possible when vehicles are within certain parts of the space-time continuum (such as near an electronic sign, at night) [40].

3.1.2 From Attack Surfaces to Attack Graphs. The Requires/Provides Model [50] describes attacks in terms of capabilities (the preconditions required for some subset of an attack to take place) and concepts (a set of sub-tasks that must be satisfied, represented as a Boolean expression).[4] These can be represented as a graph, in which edges represent concepts, and nodes represent capabilities. While reaching a new node may add capabilities, traversing an edge may consume them. To succeed in an attack, an attacker with some initial set of capabilities (represented by a leaf node) must traverse some path through the graph until they reach their goal state, at the root of the graph. There may be multiple paths available between the attacker's leaf node and the graph root.

We extend this idea to an attack surface. As an initial approach, we describe the attack surface in terms of its entry points — the leaf nodes representing the capabilities to launch an attack. But any intermediate node may represent a set of capabilities that the attacker already has, and is therefore also an entry point. Further, the attacker may already possess the goal state. Therefore, our initial attack surface consists of all nodes (entry points) such that there is a sequence of rule applications of length $n \geq 0$ from that node to the goal state. (See Figure 1.)

Note that both temporal and spatial limitations on a system can be represented using capabilities, and that these capabilities must be acquired by the attacker for certain attacks. For example, if an attack only works in a specific location, or in a specified time range, the attacker may utilize other capabilities to force a vehicle into the required region of space-time. The set of valid attack paths would, therefore, include nodes that have appropriate space and time requirements.

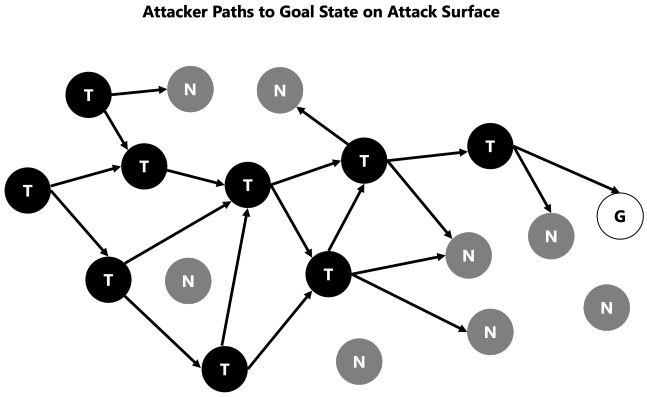

Figure 1: The attacker wants to reach their goal state, G. While the attacker can enter the surface at any state in the surface, only "threat states" (T) are connected by a capability path to G. All other states are "non-threat states", N. An entry at T offers the potential for a successful attack. An entry at N offers no possibility of success. A defender can block an attack by removing capabilities from a path, so that the attacker can no longer traverse the path to the goal state. This removal can be implemented using policy.

3.2 Defense Surfaces

A defense surface is similar in structure to an attack surface, but models how attacks can be addressed. That is, it also consists of entry and exit points, channels, and untrusted data, and is augmented by a set of rules that the system enforces.

Definition. For a system with environment E, the system's defense surface is the tuple (M^E, C^E, I^E, A^E) where M^E is the set of entry and exit points in the system, C^E is the set of channels of the system, I^E is the set of untrusted data items in the system, and A^E is a set of actions that a defender can apply to the resources (M^E, C^E, and I^E) and rules R^E to prevent, detect, or monitor an attack.

An action here is some step or steps the defender can take to handle the attack. For example, the action may be to delete or modify a rule, which would involve preventing the attacker from using the rule in some way. It may involve validating untrusted data to remove it from I^E. The defender also can simply set up monitoring to see when the rule is used.

Key to instantiating this surface is to identify what defense mechanisms support attacks being prevented, or detected and actions dispatched to mitigate the attack (the "Resilience" cycle). Detection is often achieved at runtime by intrusion detection, but code inspection or validating cryptographic signatures of code and data may identify that attacks have occurred. Once an attack is detected, actions can be taken to (1) block or at least slow the progression of the attack and (2) recover from the attack. The defender is able, in principle, to address attacks at numerous locations in an interconnected system. These include not only the ultimate target of the attack, but also nodes upstream from the target, so as to block the attack from reaching its ultimate destination. Thus the defender can, again in principle, dispatch "methods" to locations in the network where effective attack counter actions can be taken, provided these locations are accessible to the defender.

Example 1 (*continued*). The defense surface includes the action of responding. The security policy would determine whether the system should respond, and if so, how. In the former case, where the system should not respond, the point of the action in the defense surface would be that the system remain silent.

[4] The required state is collapsed into the capabilities and so is omitted in what follows.

Example 2 (*continued*). The defense surface for the platooning CAVs is the mechanism used to detect masqueraders and spoofing. One defense is to require each vehicle to authenticate itself when joining the platoon, and require all messages to be digitally signed using a public key algorithm. This prevents forged messages from being accepted. Here, the action is to accept the initial authentication and validate each message.

Again, similar to an attack surface, a defense surface consists of rules (modeled with pre-conditions and post-conditions), communication channels and data objects. We can think of the preconditions defining capabilities a "defender" needs to invoke a method and the post-conditions as defining the capabilities the defender gains or loses through execution of the method.

Thus a defense surface models the points and settings needed to prevent an attacker from gaining the capabilities enabling them to reach a given goal state (root node), or anything that alters a path from the attacker's current state to a goal state in such a way that no capability set can be used to traverse it. The basic defense surface, therefore, includes a defense mechanism that removes the capability of an attacker to traverse an edge along a path to a root node.

Note that capabilities need to be blocked along a path, not just at a single node. An attacker who is blocked from obtaining a required capability at one node might add that capability at another node along the attack path. Failure to remove the capability from the entire path may delay, but not block, the attacker. Additionally, because multiple paths might be traversable from the attacker's current state to the goal state, an ideal defense surface must block all such paths in order to guarantee that the attacker cannot reach the goal state. In practice, the defender may not know, or be able to learn of, or block, all such paths, or even all edges that make up all such paths. Therefore, in practice, a defense surface blocks a subset of all edges that make up the set of all traversable attack paths.

Note also that blocking the discoverability of a capability is not equivalent to controlling access to that capability. For example, if the defender cannot block a path, but can block the attacker from being able to infer the existence of that path (essentially, making critical attack path segments invisible to the attacker), then the defender wins (at least in the short term). Short term path invisibility is equivalent to security through obscurity, and shares its susceptibility to discovery.

3.3 Policy Surfaces

A policy surface represents the security policy of the system.

Definition. Given a system with environment E, the system's policy surface is the tuple (M^E, C^E, I^E, P^E) where M^E is the set of entry and exit points in the system, C^E is the set of channels of the system, I^E is the set of untrusted data items in the system, and P^E is a set of constraints applied to the use of a set of resources (M^E, C^E, and I^E).

The constraints apply to the use of the resources. There are three types of constraints: what is required to be done, what is allowed but not required to be done, and what is disallowed to be done. As an example, starting a car is required to use it. Adjusting seats, mirrors, and so forth are allowed but the car will run without the driver doing so. Not fastening your seat belt is disallowed, as a driver can receive a ticket for not doing so.[5] In theory, these constraints cover every possible action and state. In practice, they do not, as a system with non-trivial uses rarely (if ever) has its security policy complete. What usually happens is that users take actions resulting in a state that should have been explicitly disallowed but was not. So a new constraint is added to the policy. Contrariwise, some action may be blocked even though a job requires the system to enter that state. So an existing constraint is altered or removed. We address this case in the future work section; for now, we assume the constraints cover every case, and so the security policy is complete.

This contrasts the policy surface with the defense surface. The latter deals with the points and settings that (usually imperfectly) enforce the policy. Hence the surfaces overlap at points where the defense techniques and mechanisms correctly enforce the policy, and gaps between the surfaces exist where they do not. This is the traditional view of policy vs. mechanism: policy says what is and is not allowed, and mechanisms (attempt to) enforce the policy.

Example 1 (*continued*). The policy surface constraints includes constraints that enable the detection and countering of port sweeps. These guide the actions in the defense surface. For example, the policy may say not to respond to any probes; the defense mechanism would then ensure only exchanges that are correct for the protocol tied to that port.

Example 2 (*continued*). The policy surface for the platooning CAVs contains constraints that require the CAVs all be legitimate members of the platoon, in the sense that they are providing accurate information to the other CAVs, and that any messages come from cars in the platoon. The defense surface chooses the mechanism used to ensure this. In our example, it is to require authentication and validate digital signatures on the messages.

Policies can be expressed in many ways, for example, using invariants, or using constraints. Four forms of constraints compose the rules of the policy surface.

- Assumptions — for example, any new user will be properly identified and vetted before being given a login and password;
- Technical constraints — for example, a privileged program will function correctly and not grant privileges unless the security policy requires it;
- Physical constraints — for example, a computer that measures the speed of a car, and will prevent the car from exceeding the safe speed for the road; and
- Social constraints — these include laws and regulations, the rules of the environment in which the policy is defined, and customs of that environment.

The first three can be validated by intrusion detection systems. For example, the IDS could monitor the fan temperature through a different circuit than the computer controlling the fan uses, and if the fan overheats and is not shut down, the IDS could signal a failure. Similarly, the IDS can verify that technical constraints are valid. Of course, the assumptions cannot be mechanistically validated by their nature, such as a belief that another organization's systems meet certain requirements. In this case, one must trust the

[5] At least in many US states.

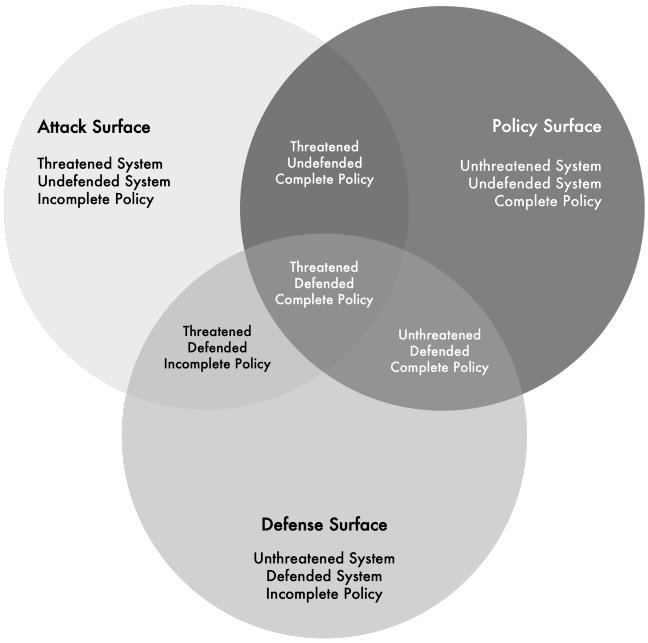

Figure 2: Relationships between surfaces at the same recursion level. The attack surface contains the set of states that are under threat of potential attack. The defense surface contains the set of states that the defender can use to prevent or mitigate attacks. The policy surface contains the set of states where the system policy is complete - that is, where the policy specifies what is required to be done, what is allowed but not required to be done, and what is disallowed to be done. Where the attack and defense surfaces intersect, the attacker can disrupt defenses, and the defender can disrupt attacks. Where the attack and policy surfaces intersect, the attacker can alter policies to aid attacks. Where the policy and defense surfaces intersect, the defender can use policy to defend the system. Where all three surfaces intersect, the attacker and defender can make moves and counter-moves against each other. They can also alter policy to enable their moves, and block those of their adversary.

assumptions, and as Singer and Bishop argued [47], the basis for trust is that set of assumptions that cannot be checked. As for local constraints, those that can be encoded into rules or other analysis mechanisms can be verified to hold in the system, and those that cannot be must be trusted to hold.

For our example of a network port, the policy surface would state whether the system should reply. How this is implemented is not relevant to this surface. As for the user interface that does not check for buffer overflow, the policy surface would constrain the user interface so as to prevent actions beyond those intended for the user interface. Again, the implementation is embodied in the defense surface.

We illustrate the relationships among the three types of surfaces in Figure 2.

3.4 Combining the Surfaces

The surfaces are related to one another, as hinted above. We now examine those relationships.

3.4.1 Relationship of Attack and Defense Surfaces. We can use attack and defense surfaces to determine whether it is possible for either the attacker, or the defender, to achieve their goals. If a defense surface is applied, and a path still exists from the attacker state to the attacker's goal state, then the attacker wins. Otherwise, the defender wins.

We can measure the vulnerability of a system by finding the intersection of the attack and defense surfaces. The portion of an attack surface that does not intersect with the defense surface contains the set of entry points, exit points, untrusted data, and rules — in short, the capabilities — that the attacker can use to achieve their goal. The portion of the defense surface that does not intersect the attack surface is not useful against the attacker's capabilities (although they may deter the addition of new capabilities). The portion of the two surfaces that intersects is the set of attacks that the defender can defend against.

In terms of preconditions, the defense surface should be the complement of the attack surface, thus ensuring the mechanisms block the attacks.[6] So if the attack surface is $A_S = p_1 \vee p_2 = (c_{11} \wedge c_{12} \wedge c_{13}) \vee (c_{21} \wedge c_{22} \wedge c_{23})$, then the ideal defense surface should simply be $D_S = \neg A_S$. For this to be true, the policy must have two parts: $\neg p_1 \wedge \neg p_2$. This means $\neg p_1 = (\neg c_{11} \vee \neg c_{12} \vee \neg c_{13})$and similarly for $\neg p_2$. And as before, this reduces to a satisfiability problem.

Example 1 (*continued*). Here, the surfaces intersect at the point where the defense is active and the attacker attempts to enter the system. The action of the defense mechanism would block the attacker, or would detect the attacker and give appropriate warning.

Example 2 (*continued*). The attack surface and the defense surface are related by the entry point rules and actions. Here, the entry points are the receptors of the CAVs, and the rule is to spoof a legitimate CAV. The attack entry point is the receptors that receive the messages guiding the platoon. The defense entry point is the use of cryptography and digital signatures to enable detection of bogus messages.

3.4.2 Relationship of Attack and Policy Surfaces. The policy surface is a representation of the security policy of the system. It defines the resources (entry and exit points, channels, and untrusted data) relevant to the security state of the system. In theory, it is complete; that is, it partitions all states into allowed and disallowed states. Were it complete, it would define the attack surface. The resources for both are the same; the requirement is that the attacker make some set of those resources not satisfy the constraints of the policy surface. Thus, the policy and attack surfaces are related by the constraints. The constraints defining the rules of the attack surface are the complement of those defining the policy surface, so every attack represents a violation of the policy.

The key word is "in theory". In practice, policies are set up initially, and then they evolve as new compromises of the system are discovered. One way to think of this is that a policy has two

[6] Note that, in practice, this is exceptionally rare.

parts, one explicit, the other implicit. New attacks compromise the implicit policy, after which those parts become explicit. Thus, in practice, the policy surface does not cover all resources in the attack surface.

3.4.3 Relationship of Defense and Policy Surfaces. Given a generic policy surface, there is no algorithm to construct a generic defense surface [24]. So defense surfaces must be constructed based on a specific (or a class of specific) policy surfaces.

The policy surface composition drives that of the defense surface. The defense surface attempts to implement the policy surface constraints. The mechanisms it uses to do so are prevention, detection and recovery mechanisms. Prevention occurs when a resource cannot be used in an attack, and hence are not points of the attack surface, Detection and recovery mechanisms observe the resources. If settings of a resource violate the policy surface constraints, the detection mechanisms report a possible security breach, and the recovery mechanisms attempt to restore the valid settings and either remove or monitor the attacker's actions.

Example 1 (*continued*). The intersection of the policy surface and the defense surface lies in the actions and constraints upon those actions. The policy surface constraints define what actions the defensive mechanisms are allowed to exercise. Here, the policy constraint is that only legitimate messages will be accepted by the CAVs. The defense surface actions to enforce this constraint are to require initial authentication and subsequent verification of the digital signatures.

Example 2 (*continued*). As in Example 1, the policy surface and the defense surface are related by the constraints and the defense mechanisms enforcing those constraints. The policy constraints relevant here are the ones that require the CAVs are legitimate members of the platoon, and send signed, unforgeable messages within the platoon. The defense actions are the use of cryptography and digital signatures to enable detection of bogus messages.

3.4.4 Relationship of the Three Surfaces. The simplest way to explain this is that the policy surface consists of resources and constraints; the defense surface describes what resources, settings, and mechanisms are involved in ensuring the settings of those resources meet the constraints of the policy surface; and the attack surface consists of those gaps where the policy and defense surfaces do not overlap.

More precisely, let A_S be the attack surface. Then the defense surface D_S relies on a set of mechanisms, M, that enforce preconditions. Thus, the set of attacks that will fail is $A_S - D_S$, that is, the resources used by the attack surface that are not protected by the defense surface. Ideally, the policy surface $P_S = D_S$; in practice, $P_S \cap D_S \neq \emptyset$, and that intersection describes the vulnerabilities.

3.4.5 Information Asymmetry in Surfaces. An information asymmetry exists between the attackers and defenders. The attackers know their capabilities and probe systems for weaknesses. They need find only one. The defenders know (or believe they know) the capabilities of the defense mechanisms, but they do not know where all the weaknesses are. And to prevent any attacks from succeeding, the defenders must defend all weaknesses.

Using policy surfaces, part or all of a surface may be rendered hidden, vulnerable, or invulnerable, to sets of attackers or defenders. This highlights the importance of establishing where the policy surface and the attack surface overlap. The part of the attack surface not being overlapped represents the weaknesses that the attacker can exploit. This shows a flaw in the policy, and presumably the requirements.

The same is true for the defense and attack surfaces. Here the danger is more immediate, as that part of the attack surface that is not overlapped represents ways the system can be attacked without defense mechanisms interfering.

By manipulating the overlaps of the three surfaces, the attacker can identify and implement attacks that the defender either can not, or will choose not, to defend against. In doing so, the attacker may not only reach their desired goal state, but can also force the defender to consume their resources to defend against attacks that the attacker does not plan to implement. Using the same manipulation, the defender can attempt to predict, identify, and block attacks, make parts of their system invisible or inaccessible, or change the cost or value to the attacker of implementing attacks.

This illustrates information asymmetry between attackers (who have perfect knowledge of their own capabilities, but incomplete system knowledge) and defenders (who have perfect system knowledge but incomplete knowledge of current and potential attacker capabilities). Manipulating these surfaces alters this information asymmetry, and both attackers and defenders can use this to their advantage. Specifically, to be more in their favor, attackers can learn of new paths and capabilities, while defenders can detect, neutralize, and respond to threats more efficiently.

3.4.6 Selecting a Defense Action. An attack becomes possible if some satisfiable attack path exists from the attacker's current point on the attack surface to some point that violates policy. If the attacker is not yet at the violation point, the defender might predict that the attacker will reach it by determining whether any attack paths exist from the attacker's current point to the violation point. The defender can then attempt to block the attacker by removing capabilities or negating preconditions, such that the attacker can not satisfy the requirements to traverse the attack path. On the other hand, if the attacker is already at the violation point, then the defender needs to find a way to move them out of that point, and into a point that does not cause a policy violation. One way to do this is to remove capabilities from the attacker, such that they no longer have the ability to remain at the violation point. The other option is to force the attacker to transition to some new point that does not violate policy by changing the state of the system through a change in policy, capabilities, or resources. Such a transition should, if possible, also not lead to a point that allows the attacker to directly transition to a violation point.[7]

Assume an attack $A \in \mathcal{A}$ has postcondition po that violates a policy rule R. There exists a defense action D with precondition D_{pr} and post-condition D_{po} such that $A_{po} \wedge R \Rightarrow D_{pr}$ and $D_{po} \Rightarrow notA_{po}$ and D_{po} does not violate any policy (that is, $D_{po} \wedge \text{policy} = False$). Then any D that satisfies these conditions is an acceptable defense action.

[7]We believe that model checking [9, 38], which is efficient for reasonably small problems, is likely to provide the best solution. This is an area of future research.

4 STRATEGIES FOR ATTACK AND DEFENSE

Our model can be used to develop attack and defense strategies for given systems. We have identified some of these strategies.

4.1 Information Asymmetry in Policy Surfaces

An information asymmetry exists between the attackers and defenders. The attackers know their capabilities and probe systems for weaknesses. They need find only one. The defenders know (or believe they know) the capabilities of the defense mechanisms, but they do not know where all the weaknesses are. And to prevent any attacks from succeeding, the defenders must defend all weaknesses.

Using policy surfaces, part or all of a surface may be rendered hidden, vulnerable, or invulnerable, to sets of attackers or defenders. This highlights the importance of establishing where the policy surface and the attack surface overlap. The part of the attack surface not being overlapped represents the weaknesses that the attacker can exploit. This shows a flaw in the policy, and presumably the requirements.

The same is true for the defense and attack surfaces. Here the danger is more immediate, as that part of the attack surface that is not overlapped represents ways the system can be attacked without defense mechanisms interfering.

By manipulating the overlaps of the three surfaces, the attacker can identify and implement attacks that the defender either can not, or will choose not, to defend against. In doing so, the attacker may not only reach their desired goal state, but can also force the defender to consume their resources to defend against attacks that the attacker does not plan to implement. Using the same manipulation, the defender can attempt to predict, identify, and block attacks, make parts of their system invisible or inaccessible, or change the cost or value to the attacker of implementing attacks.

This illustrates information asymmetry between attackers (who have perfect knowledge of their own capabilities, but incomplete system knowledge) and defenders (who have perfect system knowledge but incomplete knowledge of current and potential attacker capabilities). Manipulating these surfaces alters this information asymmetry, and both attackers and defenders can use this to their advantage. Specifically, to be more in their favor, attackers can learn of new paths and capabilities, while defenders can detect, neutralize, and respond to threats more efficiently.

4.2 Attacker Strategies

If the cost of a defense is too high, the defender will be unwilling or unable to carry it out. The attacker can raise the defense cost to a level that is higher than the defender is willing to commit to the defense of the system. In particular, if the cost of the defense is more than the losses resulting from the attack being successful, the defender may choose to allow the attack to succeed.

The attacker may wish to exhaust the available resources of the defender, in order to increase the feasibility of future attacks. The attacker can drive up the defender's costs by focusing on the paths with the highest cost of defense. However, because the attacker may have only partial information about the cost of defending those paths, the attacker may not have sufficient information to implement this strategy successfully.

Finally, the attacker can force the defender to reveal a subset of their defensive capabilities by finding attack paths in which the defender must either block the attacker by revealing a capability, or allow the attacker to reach some intermediate state. (This is similar to a "fork" in chess.)

4.3 Defender Strategies

4.3.1 Root Cause Analysis. In order for a given failure to occur, the preconditions for that failure must be satisfied. If the failure can occur if one or more independent failures occur, then at least one of the preconditions must be satisfied. Given a particular failure, we can perform a root cause analysis by tracing back from the failure, examining which capabilities were (or must have been) utilized in order for the failure to occur. We can continue to trace back until the root cause of a failure is identified. A vulnerability exists if a failure has not yet occurred, but the preconditions for that failure are satisfiable.

In the case of recursive surfaces, this may involve starting with a failure at a higher level, then tracing through to potential lower level causes. Or, we may start with a low level failure, then examine the dependencies for the subsystem that failed, moving through system layers until we reach all possible consequences of the failure.

4.3.2 Utilizing Information Asymmetry for Defense. The attacker may not know all possible policies, and may not know what defenses have been implemented. To avoid wasting resources or revealing their capabilities prematurely, the attacker must choose to attack only those paths for which they believe that they have sufficient capabilities to achieve their goal.

The defender can force the attacker to reveal capabilities. In the absence of sufficient information, the attacker may decrease their capability set by using them in unsuccessful attacks, or even by revealing them in successful attacks. (As an example, a zero-day attack ceases to be a zero-day attack once it has been used, because the capability that it utilizes has been revealed to the defender.) Defenses that force the attacker to reveal capabilities allow the defender to block the use of those capabilities.

The defender can drive up the cost to the attacker of reaching a certain goal state by eliminating the attacker's ability to traverse some possible paths leading to that goal. The defender must defend all possible attack paths. The attacker need traverse only one valid path. However, the defender may have only partial information about the cost of attacking those paths.

4.3.3 Defender Path Manipulation. Finally, in some cases, two paths to a given goal state may merge at a given node. (i.e. There may be two ways for the attacker to reach some intermediate node.) The defender can block both paths simultaneously by preventing satisfaction of the preconditions for any required state transitions out of the intermediate node.

5 DISCUSSION AND FUTURE WORK

In this section, we discuss how our model might be applied to system design and operation, and to attack detection and system defense. We also analyze some of the challenges that we intend to address in future versions of the model.

5.1 System Design and Operation

The first question is how to use our model to design systems that have resilient architectures, and how to generate requirements for those architectures.

5.1.1 Resilient Architectures. Some systems are brittle when attacked. They may enter a state in which they can no longer achieve their intended objectives (for example, if a CAV crashes, it can no longer achieve its goal of safely transporting its passengers to their destination). They may experience degraded performance (the CAV gets the passengers to their destination, but provides an unsatisfactory experience while doing so). Or, they may not be vulnerable to the capabilities and attack strategy used by the attacker. Other systems are resilient [5]. They maintain *sufficient* functionality to achieve *some* goal of the system, even if it was not the system's original or primary goal. For example, a resilient CAV may have the primary goal of transporting its passengers to their destination safely within certain time and cost constraints. But if an attack occurs, the CAV may switch goals (coming to a safe stop in a protected area instead of reaching the specified destination or switching passengers to a different vehicle), or may alter its constraints (degrading service quality, timeliness, or value for money) in order to provide the best service that it still can while avoiding system failure. Using our model, the system can dynamically (re)-evaluate possible actions, given its current capabilities and reachable set of goals. If, starting in its current state, the system can not find a sequence of moves that would allow it to reach a specific goal state, that goal state can be pruned from its set of potential options. The system can then evaluate the costs of the paths to the remaining goal states, add them to the rewards associated with those goal states, and select the goal with the highest net reward.

5.1.2 Generating Requirements. Requirement specification allows system architects and engineers to determine whether or not their designs meet legal, social, technical, operational, or other requirements. While requirement generation is an area for future work, one potential approach is to make (a subset of) requirements equivalent to satisfaction of state transition preconditions. For a transition to occur, the right preconditions, capabilities, and policies must have been satisfied by the relevant user, or by the system itself. Requirements could be translated into policies, lists of capabilities, and design constraints to be applied in different circumstances. For example, a requirement to protect CAV passenger safety might be translated into a set of safety capabilities that would satisfy that requirement. The CAV's surface graphs could then be analyzed to determine whether required goal states were reachable with those capabilities in place, or in the event that one or more capabilities were disabled.

5.1.3 System Operation. Modeling of attack, defense, and policy surfaces improves the ability of system operators to track and predict the current and future states of a system, respectively. As capabilities and system states change, those changes can be applied to the models of each surface, and any new or updated paths through the surfaces can be (re)evaluated. Note that while these models can be detailed, they likely can never be complete for systems of non-trivial complexity. While model completeness is a potential limitation, a lack of complete knowledge always affects attackers and defenders. Both attackers and defenders have limited knowledge of each other's capabilities, strategies, policies, and state. If either an attacker or defender does not know that they can make a specific move, it is very unlikely that they will execute that move. Thus, an incomplete model reflects what is both *possible* and *known* to the attacker and defender, rather than all possible moves.

5.2 Defending Systems

Another question is how to use the model to improve the ability of defenders to protect their systems. Our future work on system defense will focus on the application of our model to three areas — attack detection, attack diagnosis, and resilient response.

5.2.1 Attack Prediction and Detection. Traditionally, attack detection focuses on the identification of attacks that either have already occurred, or that are currently ongoing. But our surface models provide an additional alternative — the possibility of predicting both attack preconditions and attack effects, given available capabilities and current or future system states. One option may be to identify sets of preconditions required to gain entry to paths to specific goals, and then to monitor systems to determine whether those preconditions have been satisfied. Another option is to predict the effects of potential attacks, as represented by changes in available capabilities, satisfaction of transition prerequisites, and traversal of users or systems across paths to specific goals. Those predicted effects can then be used as signatures to detect ongoing and past attacks. In future work, we will analyze these options, and demonstrate how surface modeling can make them feasible to implement.

5.2.2 Attack Diagnosis. Attack detection and attack diagnosis are distinct. While attack detection focuses on determining whether or not an attack is occurring, attack diagnosis focuses on identifying the nature of current and past attacks, including classification of attack family, Tactics, Techniques and Procedures (TTPs), attack stages, and attack goals. One potential advantage to our model is that it can track and relate multiple, simultaneous attacks. Consider two attacks that are targeted towards achieving the same goal, but that appear to be implemented by two different external attackers. In traditional attack detection, it is possible to detect each attack, but it is difficult or infeasible to relate the two attacks. Using our surfaces, it is possible to track whether capabilities gained in one attack are applied in other attacks, and whether two attacks may ultimately be converging towards the same (intermediate) goal along intersecting paths. In future work, we will demonstrate relation finding between attacks that would otherwise appear to be distinct.

5.2.3 Example: Honeypots. A security policy may contain entry points for attackers to exploit something in the system. This occurs in honeypots, where the goal is not to prevent attacks, but encourage them — so they can be monitored. In this case, the security policy either states or implies a certain class of attacks do not violate the policy. Here, the defense surface does not overlap the policy surface at those points, but it *does* overlap the attack surface. Here, though, the mechanism is to monitor the steps in the attack.

5.3 Analysis

Finally, we explore ways in which our model could be enhanced to improve reasoning about security and strategic analysis, to improve modeling of surface interactions, address temporal and probabilistic constraints, and to compute metrics with respect to different surfaces.

5.3.1 Reasoning About Security. Our model may make it easier to reason about the security of a system, from the perspectives of both attackers and defenders. Attackers can evaluate the cost, number of steps, required capabilities, or other parameters for both simple and complex attacks, including the coordination of multiple attack strategies. Defenders can do the same with respect to their defenses. Comparisons can be made between different strategies — not just of cost, but also of probability of success. Developing techniques for automating this reasoning capability is an area for future work.

5.3.2 Edge Cost Annotation. Each step in an attack may cause the attacker to incur one set of costs, and the defender to incur another. If an attack causes dependencies within a system to fail or degrade in functionality, the degradation of those dependencies may also result in a cost — even if the attacker does not attack the dependencies directly. While it is very difficult to accurately estimate the cost of an attack to either the attacker or defender, it may be much easier to do so if the costs for transitions to new states, including potential dependencies, are annotated along each path edge. Finding the best way to derive these costs is an area for future work. However, if edges have been annotated with costs, it should be possible to make direct comparisons of the costs of different attack and defense strategies, and to use these comparisons in strategy evaluation. For example, a defender might dynamically adjust the cost of reaching certain goals by changing policies and capability requirements in order to make reaching those goals too costly for an attacker. Likewise, an attacker might alter their strategy in order to consume or misdirect most of a defenders resources, thus making it more difficult to defend the attacker's actual target.

While it is also possible to annotate an edge with the probability of an attacker move, this is problematic for several reasons. One reason is that there may be many attackers (some of whom may collude). The probability of a given move may be very different for each attacker, depending on their goals, capabilities, and ability to see different parts of the attack surface. Another reason is that individual attackers may need to change strategy and goals as those same things change. While a probability of any user following a certain edge or path can be computed, individual attackers have different goals and behavior than ordinary users, and possibly from each other. Thus, while probabilities can be assigned to edges, they may be inaccurate, or even misleading.

5.3.3 Uncertainty. A key limitation of our work is that in many cases, it is not possible to know the probability with which certain events will occur in the future. This limitation stems from the lack of complete information that both attackers and defenders must contend with. The attacker's view of the target system is limited. The defender, while potentially able to see the entire target system, has a limited view of attacker capabilities, as well as limited insight into attacker incentives and motivations. This is further complicated by the fact that many systems have sufficient complexity that it is not feasible to map out all possible moves, dependencies, state transitions, or impacts on dependent subsystems. Discrete event systems [6, 43] may provide a solution to this problem. Their application to our model is an area of future work. Discrete event systems [6, 43] may provide a solution to this problem. Their application to our model is an area of future work.

5.3.4 Underspecification. Neither the attacker nor the defender is likely to have a complete view into any non-trivial system. From the perspective each, the system being attacked or defended will be underspecified, which may result in emergent behaviors [19]. The manipulation of behaviors emergent to an adversary is a potential tool for both attackers and defenders, who may manipulate the visibility and accessibility of points and paths within a system's surfaces. Whether or not this can be used to make a *known* path to goal unreachable, or to render acquisition of the goal irrelevant, is an area for further study.

5.3.5 Dynamic Surfaces. Systems do not stay static. Their configurations change over time. While some portions of a surface may remain highly stable, others may change frequently. Some may even incorporate feedback loops and control mechanisms. By focusing on stable portions of a surface, attackers can increase their odds of reaching a desired goal without their chosen attack path being interrupted. Likewise, defenders may use surface instability as a defensive measure. This is effectively a moving target defense [8]. Focusing monitoring on stable surfaces may improve the ability of both attackers and defenders to achieve their goals. In future research, we will address the value to the defender of maintaining points requiring defense within stable regions of surfaces, and to the attacker of using surface region instability to improve attack stealthiness, and to find or manipulate windows in which attack goals can be reached.

5.3.6 Complex Attacks and Multiple Attackers. An attack may have multiple starting points on a surface, or across multiple surfaces. This allows the attacker to gain additional capabilities, potentially using a capability gained in one attack to aid in another. This enables the attacker to make it more difficult to detect and defend all active attack paths. For example, the use of multiple attack paths may prevent the defender from correlating and associating specific actions, or the acquisition and use of particular capabilities. Unfortunately for the attacker, the defender can add uniqueness to capabilities in order to detect use across attack paths. For example, the defender could require the use of a token to traverse a given edge. If that token incorporates a non-forgeable unique identifier, then using it would allow the defender to create a binding between the acquisition and use of the token along different paths.

An attack may also have multiple (potentially unrelated) goals with the same starting point. Using our model, an attack could be traced as it moves from one point to the next within a surface. These paths could be traced back to their common starting point, allowing them to be associated.

Multiple attacks may also occur simultaneously. Sometimes, these attacks will be related. In this case, if there is any sharing of capabilities or path usage, then an association between the attacks can be made. In the case where the attacks are not related, the defender should still be able to track and counter each of the attack.

In each of these cases the attacker can also attempt to evaluate whether making a countermove to block the use of one attack path would cause an attack on a different path to gain a capability on a subsequent move. The defender can evaluate the potential consequences of possible responses to each attack, making trade-offs between them to achieve a better overall outcome.

5.3.7 Metrics. The effectiveness of the defense mechanisms can be measured in one of several ways. First, how much of the attack surface does the defense surface cover? If there are parts of the attack surface that the defense surface does not cover, the system is vulnerable. The ratio of coverage versus non-coverage for a particular institution's policy and defenses can be measured by these gaps. Note the defense surface may not intersect with the attack surface at some points, indicating the defense mechanisms may block legitimate work. This can be factored into any metric.

A second useful measure would be the amount of the policy surface that the defense surface covers. Again, if there are gaps, those gaps identify parts of the policy not enforced by the defense mechanisms. The places where the defense surface does not intersect with the policy surface may indicate the defense mechanisms prevent authorized actions. Perhaps this can be handled in the same way as the attack and defense surfaces; this is an area ripe for exploration.

5.3.8 Avoiding Cycles. Paths between any two nodes must terminate, but it is possible that a path could be created that contains a cycle, resulting in an infinitely long path. To prevent this, any algorithm that finds or evaluates paths between nodes must incorporate cycle detection, and must terminate if a cycle is detected. One possible way to do this may be to assign each node a unique identifier (UID), and to track whether a node with a given UID has already been visited by a given algorithm. However, it might be possible for an attacker to engineer an attack that would cause cycle detection to fail. Finding ways to preclude such an attack from becoming possible is an area for future work.

5.3.9 Predicates. If we represent the attack and policy surfaces as a disjunction of predicates, with each predicate being derived from a requirement, we can combine the policy and defense surfaces. The defense surface would characterize possible values of the variables in the predicates. Then if the attack surface and policy surface contradict each other, the defense must be changed in order to enforce the policy. Under this view, one would need to include positive requirements (what is allowed) as well as the more usual negative requirements (what is not allowed). If there is a successful attack, then there is no contradiction between the policy surface and the attack surface for the values that allow the attack to match the policy, and thereby succeed. Finding the values of the variables that would allow either of these alternatives is a problem in satisfiability. One could also attach varying weights to the values of the predicates to obtain a measurement of the effectiveness of the defense and policy surface. How to do this, and what measures would be useful, probably depend on the specific organization targeted in a particular attack. Whether this view of the surfaces, or a more traditional view, is more effective is an area warranting further work.

5.3.10 Time Constraints. Some systems must detect and respond to attacks within hard time constraints, in order to avoid adverse effects such as system crashes. For a CAV to avoid a physical crash, response times must be on the order of 100ms. Our representation allows for the possibility of pre-computation of a subset of potential attacker and defender moves, before those moves become critical. In cases where edges or surfaces are re-used, it may also be possible to re-use computational results throughout the represented system, further decreasing overall computation time.

5.3.11 Attacks on Machine Learning Models. Some attacks are executed against machine learning models, rather than traditional "systems". Two examples of this are model inversion attacks [16] and data poisoning attacks [1]. In a model inversion attack, an attacker attempts to reconstruct the classifier's training set – deriving the distribution of classes within the training set, and potentially variations within those classes [52]. In a data poisoning attack, the attacker manipulates the behavior of a machine learning model by inserting invalid examples into the model's training set, thus causing misclassifications to occur [48].

In the first example, the goal is the set of classes (and possible variants) in the training set. To get to the goal, the attacker must gain access to the trained model, and potentially also to outputs from that model. The capabilities required are those needed to access and execute the model, and to access the model outputs. In the second example, the required capabilities are the ability to access and modify (add to) the training set, and (potentially) the ability to cause the model to be trained on the modified training set.

A subset of data poisoning attacks, called "dirty label attacks" [45], causes the classifier to misclassify when exposed to a specific trigger, such as a stop sign that has had a sticker attached to it [18]. In this case, there are *two* attack paths that the attacker must traverse to gain the required capabilities. The first path is the same as before – achieving the goal of poisoning the classifier's training set. The second path has the goal of generating the trigger (placing the sticker on the stop sign). If both attacks are executed by the same attacker, then once the attacker has generated both capabilities, they may then transition to their final goal of inducing a misclassification of the target by the classifier. If the attacks are executed by different attackers, then if one attacker has already successfully executed the data poisoning attack, that precondition is already satisfied for the second attacker, who then only needs to obtain and execute the capability to generate the trigger.

5.3.12 Trust of Surface Data. Some systems, such as vehicle platoons, are very complex, and may even be ephemeral. The composition of a platoon may vary from second to second as vehicles join, leave, maneuver, and interact. Rather than using a static architecture, a system such as a platoon may need to be dynamically defined. In some cases, each subsystem may need to report on the configuration of its own surfaces. In other cases, subsystems may appear as black boxes from the perspective of the larger system, even though they represent complex systems internally. This raises the question of how each subsystem can trust the others, and how the overall system can trust information reported to it by each subsystem. We are studying this problem. The Solar Trust Model provides the basis for a possible solution, as it is designed to model ad-hoc trust relations between entities that lack prior relationships.

5.3.13 Building Surfaces. It is important to define algorithms and techniques for efficiently identifying and updating the points and edges on each surface. As previously indicated, neither the attacker nor the defender are likely to have a complete view of their system. This view may be limited by resources and capabilities available to map and monitor the system state. Currently, we envision the use of libraries of surface models, much like object oriented software is typically composed using libraries of previously defined components. However, finding more optimal ways to generate surface models is an area of future work.

6 CONCLUSION

We have presented several new paradigms, including recursive surfaces, defense surfaces, and policy surfaces as extensions of the extensively studied concepts of attack surfaces. Through these three kinds of surfaces it is possible to specify and analyze attacks and defenses on complex systems in order to determine if a policy is respected. Defining these surfaces using recursion, efficient representation of attacks and defenses is achieved, for example when configurations are replicated within a system. We believe the surface definitions can also be used to enhance system resilience, to derive better attack and defense strategies, to model system entropy, attack and defense costs. They can also be used to support root cause analysis, in order to help to eliminate vulnerabilities, or find the likely causes of prior failures.

Acknowledgements. Matt Bishop gratefully acknowledges the support of the National Science Foundation awards DGE-1934279 and DGE-2011175 to the University of California and funding from Toyota InfoTech Labs. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation or the University of California.

REFERENCES

[1] Battista Biggio, Blaine Nelson, and Pavel Laskov. 2012. Poisoning attacks against support vector machines. *arXiv preprint arXiv:1206.6389* (2012).
[2] Matt Bishop. 2019. *Computer Security: Art and Science* (second ed.). Addison-Wesley, Boston, MA, USA.
[3] Matt Bishop, Heather M Conboy, Huong Phan, Borislava I Simidchieva, George S Avrunin, Lori A Clarke, Leon J Osterweil, and Sean Peisert. 2014. Insider threat identification by process analysis. In *2014 IEEE Security and Privacy Workshops*. IEEE, Piscataway, NJ, USA, 251–264.
[4] BishopFox. 2022. BishopFox: Attack Surface Management. https://bishopfox.com/platform/attack-surface-management
[5] Fredrik Bjorck, Martin Henkel, Janis Stirna, and Jelena Zdravkovic. 2015. Cyber resilience–fundamentals for a definition. In *New contributions in information systems and technologies*. Springer, 311–316.
[6] Christos G Cassandras and Stéphane Lafortune. 2008. *Introduction to discrete event systems*. Springer.
[7] Stephen Checkoway, Damon McCoy, Brian Kantor, Danny Anderson, Hovav Shacham, Stefan Savage, Karl Koscher, Alexei Czeskis, Franziska Roesner, and Tadayoshi Kohno. 2011. Comprehensive experimental analyses of automotive attack surfaces. In *20th USENIX security symposium (USENIX Security 11)*.
[8] Fred Chong, Ruby Lee, A Acquisti, W Horne, C Palmer, A Ghosh, D Pendarakis, W Sanders, E Fleischman, H Teufel III, et al. 2009. National cyber leap year summit 2009: Co-chairs' report. *NITRD Program* (2009).
[9] Edmund M Clarke. 1997. Model checking. In *International Conference on Foundations of Software Technology and Theoretical Computer Science*. Springer, 54–56.
[10] Michael Clifford. 2002. Networking in the solar trust model: Determining optimal trust paths in a decentralized trust network. In *18th Annual Computer Security Applications Conference, 2002. Proceedings.* IEEE, Piscataway, NJ, USA, 271–281.
[11] Michael Clifford, Charles Lavine, and Matt Bishop. 1998. The solar trust model: authentication without limitation. In *Proceedings 14th Annual Computer Security Applications Conference (Cat. No. 98EX217)*. IEEE, Piscataway, NJ, USA, 300–307.
[12] Michael Allen Clifford. 2012. *The Solar Trust Model, Identity, and Anonymity*. University of California, Davis, Davis, CA, USA.
[13] Clifton A Ericson et al. 1999. Fault tree analysis. In *System Safety Conference, Orlando, Florida*, Vol. 1. 1–9.
[14] Benjamin Eriksson, Jonas Groth, and Andrei Sabelfeld. 2019. On the Road with Third-party Apps: Security Analysis of an In-vehicle App Platform.. In *VEHITS*. 64–75.
[15] Ian Foster, Andrew Prudhomme, Karl Koscher, and Stefan Savage. 2015. Fast and vulnerable: A story of telematic failures. In *9th USENIX Workshop on Offensive Technologies (WOOT 15)*.
[16] Matthew Fredrikson, Eric Lantz, Somesh Jha, Simon Lin, David Page, and Thomas Ristenpart. 2014. Privacy in pharmacogenetics: An {End-to-End} case study of personalized warfarin dosing. In *23rd USENIX Security Symposium (USENIX Security 14)*. 17–32.
[17] Nir Friedman, Lise Getoor, Daphne Koller, and Avi Pfeffer. 1999. Learning probabilistic relational models. In *IJCAI*, Vol. 99. 1300–1309.
[18] Tianyu Gu, Brendan Dolan-Gavitt, and Siddharth Garg. 2017. Badnets: Identifying vulnerabilities in the machine learning model supply chain. *arXiv preprint arXiv:1708.06733* (2017).
[19] Heather M Hinton. 1998. Under-specification, composition and emergent properties. In *Proceedings of the 1997 workshop on New security paradigms*. 83–93.
[20] Michael Howard. 2003. Fending off future attacks by reducing attack surface.
[21] Kyle Ingols, Richard Lippmann, and Keith Piwowarski. 2006. Practical attack graph generation for network defense. In *2006 22nd Annual Computer Security Applications Conference (ACSAC'06)*. IEEE, 121–130.
[22] Sushil Jajodia, Steven Noel, and Brian O'berry. 2005. Topological analysis of network attack vulnerability. In *Managing cyber threats*. Springer, 247–266.
[23] Somesh Jha, Oleg Sheyner, and Jeannette Wing. 2002. Two formal analyses of attack graphs. In *Proceedings 15th IEEE Computer Security Foundations Workshop. CSFW-15*. IEEE, 49–63.
[24] Jones Anita K. and Richard J. Lipton. 1975. The Enforcement of Security Policies for Computation. *ACM SIGOPS Operating Systems Review* 9, 5 (Nov. 1975), 197–206. https://doi.org/10.1145/1067629.806538
[25] Peter E Kaloroumakis and Michael J Smith. 2021. Toward a knowledge graph of cybersecurity countermeasures. *Corporation, Editor* (2021).
[26] Kyounggon Kim, Jun Seok Kim, Seonghoon Jeong, Jo-Hee Park, and Huy Kang Kim. 2021. Cybersecurity for autonomous vehicles: Review of attacks and defense. *Computers & Security* 103 (2021), 102150.
[27] Daphne Koller. 1999. Probabilistic relational models. In *International Conference on Inductive Logic Programming*. Springer, 3–13.
[28] Barbara Kordy, Sjouke Mauw, Saša Radomirović, and Patrick Schweitzer. 2014. Attack–defense trees. *Journal of Logic and Computation* 24, 1 (2014), 55–87.
[29] Sidney La Fontaine, Naveen Muralidhar, Michael Clifford, Tina Eliassi-Rad, and Cristina Nita-Rotaru. 2022. Alternative Route-Based Attacks in Metropolitan Traffic Systems. In *2022 52nd Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W)*. 20–27. https://doi.org/10.1109/DSN-W54100.2022.00014
[30] Wen-Shing Lee, Doris L Grosh, Frank A Tillman, and Chang H Lie. 1985. Fault tree analysis, methods, and applications — a review. *IEEE transactions on reliability* 34, 3 (1985), 194–203.
[31] Xiangxue Li, Yu Yu, Guannan Sun, and Kefei Chen. 2018. Connected vehicles' security from the perspective of the in-vehicle network. *IEEE Network* 32, 3 (2018), 58–63.
[32] Richard Lippmann, Kyle Ingols, Chris Scott, Keith Piwowarski, Kendra Kratkiewicz, Mike Artz, and Robert Cunningham. 2006. Validating and restoring defense in depth using attack graphs. In *MILCOM 2006-2006 IEEE Military Communications Conference*. IEEE, 1–10.
[33] Richard P Lippmann, Kyle W Ingols, Chris Scott, Keith Piwowarski, Kendra Kratkiewicz, Michael Artz, and Robert Cunningham. 2005. *Evaluating and strengthening enterprise network security using attack graphs*. Technical Report. Citeseer.
[34] Nancy A Lynch and Mark R Tuttle. 1988. *An introduction to input/output automata*. Laboratory for Computer Science, Massachusetts Institute of Technology.
[35] P. Manadhata and M Jeannette. 2004. *Wing. Measuring a system's attack surface*. Technical Report. Technical report, DTIC Document.
[36] Pratyusa K. Manadhata and Jeanette M. Wing. 2011. An Attack Surface Metric. In *IEEE Transactions on Software Engineering*, Vol. 37. IEEE, Piscataway, NJ, USA, 371–386. https://doi.org/10.1109/TSE.2010.60
[37] Pratyusa K Manadhata and Jeannette M Wing. 2011. A formal model for a system's attack surface. In *Moving Target Defense*. Springer, 1–28.
[38] Makai Mann, Ahmed Irfan, Florian Lonsing, Yahan Yang, Hongce Zhang, Kristopher Brown, Aarti Gupta, and Clark Barrett. 2021. Pono: a flexible and extensible SMT-based model checker. In *International Conference on Computer Aided Verification*. Springer, 461–474.
[39] Carsten Maple, Matthew Bradbury, Anh Tuan Le, and Kevin Ghirardello. 2019. A connected and autonomous vehicle reference architecture for attack surface analysis. *Applied Sciences* 9, 23 (2019), 5101.

[40] Ben Nassi, Dudi Nassi, Raz Ben-Netanel, Yisroel Mirsky, Oleg Drokin, and Yuval Elovici. 2020. Phantom of the adas: Phantom attacks on driver-assistance systems. *Cryptology ePrint Archive* (2020).
[41] Department of Homeland Security. 2022. Secure Cyberspace and Critical Infrastructure. https://www.dhs.gov/secure-cyberspace-and-critical-infrastructure. [Online; accessed 12-December-2022].
[42] Penetra. 2022. Penetra Automated Security Validation Platform. https://penetra.io
[43] Peter JG Ramadge and W Murray Wonham. 1989. The control of discrete event systems. *Proc. IEEE* 77, 1 (1989), 81–98.
[44] Bruce Schneier. 1999. Attack trees. *Dr. Dobb's journal* 24, 12 (1999), 21–29.
[45] Shawn Shan, Arjun Nitin Bhagoji, Haitao Zheng, and Ben Y Zhao. 2022. Poison forensics: Traceback of data poisoning attacks in neural networks. In *31st USENIX Security Symposium (USENIX Security 22)*. 3575–3592.
[46] Oleg Sheyner, Joshua Haines, Somesh Jha, Richard Lippmann, and Jeannette Wing. 2002. Automated Generation and Analysis of Attack Graphs. *Proceedings of the 2002 IEEE Computer Society Symposium on Research in Security and Privacy*, 273– 284. https://doi.org/10.1109/SECPRI.2002.1004377
[47] Abe Singer and Matt Bishop. 2020. Trust-Based Security; Or, Trust Considered Harmful. In *Proceedings of the 2020 New Security Paradigms Workshop*. ACM, New York, NY, USA, 76–89. https://doi.org/10.1145/3442167.3442179
[48] Jacob Steinhardt, Pang Wei W Koh, and Percy S Liang. 2017. Certified defenses for data poisoning attacks. *Advances in neural information processing systems* 30 (2017).
[49] Blake E Strom, Andy Applebaum, Doug P Miller, Kathryn C Nickels, Adam G Pennington, and Cody B Thomas. [n.d.]. Mitre att&ck: Design and philosophy.
[50] Steven J. Templeton and Karl Levitt. 2000. A Requires/Provides Model for Computer Attacks. In *Proceedings of the 2000 New Security Paradigms Workshop*. ACM, New York, NY, USA, 31–38.
[51] Christopher Theisen, Nuthan Munaiah, Mahran Al-Zyoud, Jeffrey C Carver, Andrew Meneely, and Laurie Williams. 2018. Attack surface definitions: A systematic literature review. *Information and Software Technology* 104 (2018), 94–103.
[52] Kuan-Chieh Wang, Yan Fu, Ke Li, Ashish Khisti, Richard Zemel, and Alireza Makhzani. 2021. Variational Model Inversion Attacks. *Advances in Neural Information Processing Systems* 34 (2021), 9706–9719.
[53] John H Wensley, Leslie Lamport, Jack Goldberg, Milton William Green, Karl N Levitt, Peter Michael Milliar-Smith, Robert E Shostak, and Charles Burr Weinstock. 1978. Sift: Design and analysis of a fault-tolerant computer for aircraft control. *Proc. IEEE* 66, 10 (Oct. 1978), 1240–1255. https://doi.org/10.1109/PROC.1978.11114