Reap the Rewards: Bridging Risk Assessment and Secure Application Development Practices

Abstract

Integrating cybersecurity requirements into complex systems during early design stages is essential for building resilient applications, but it remains a challenging task for both application owners and cyber operators. In this paper, we present a model-based systems engineering (MBSE) framework that supports collaborative, security-aware design by enabling application owners and cyber operators to jointly assess and prioritize cybersecurity controls. First, we introduce a framework for integrating cyber into software development during the early design stages and provide a scalable approach to ensure completeness in the allocation of cybersecurity requirements to a system. Central to our approach is REAP, a lightweight plug-in that performs rapid, RPO/RTO-informed risk assessments to guide the selection and allocation of security requirements. REAP improves transparency and decision making by helping stakeholders identify critical controls early, quantify trade-offs, and ensure system-wide coverage. We evaluated our framework in a representative network scenario, demonstrating its ability to provide meaningful security insights while seamlessly integrating into existing MBSE workflows. Our key contribution is a usable and measurement-driven method for incorporating cybersecurity into the system design process in a way that is both scalable and actionable.

CCS Concepts

• **Software and its engineering → Software verification and validation**; **Software design techniques**; • **Security and privacy → Software security engineering**.

Keywords

Cybersecurity, model-based system engineering, cyber engineering, secure-by-design, risk assessment

ACM Reference Format:

. 2025. Reap the Rewards: Bridging Risk Assessment and Secure Application Development Practices. In *Proceedings of Computer and Communications Security (CCS '25).* ACM, New York, NY, USA, 12 pages. https://doi.org/XXXXXXX.XXXXXXX

CCS '25, October 13–17, 2025, Taipei, Taiwan

ACM ISBN 978-1-4503-XXXX-X/18/06
https://doi.org/XXXXXXX.XXXXXXX

I Introduction

As computer systems become more complex, ensuring robust security and availability is a fundamental requirement for industries such as healthcare, energy, finance, and national defense. Cybersecurity threats pose significant risks to operational continuity and sensitive data, underscoring the need to incorporate security during the software development process. When outages occur, it is extremely costly: in the Colonial pipeline ransomware attack, the requested ransom was $4.3 million [13] and the Crowdstrike outage resulted in losses estimated between $300 million and $1 billion [1].

There are a multitude of frameworks and methodologies that application owners use to address these concerns. A popular framework is the Risk Management Framework (RMF), published by NIST [2], which guides users in selecting, implementing, evaluating and monitoring security controls. In healthcare, there are regulations like HIPAA that require healthcare applications to meet specific requirements which include, but are not limited to, fully functional cross-region backups and the encryption of data at rest. More generally, application owners create business continuity plans (BCPs) for their applications, which are used to plan a rapid recovery from disasters such as cyberattacks on critical applications. In these BCPs, a Recovery Time Objective (RTO), the maximum amount of time it should take to restore normal operations after a data loss or outage, and a Recovery Point Objective (RPO), the maximum amount of data an organization can accept losing, are defined, which shape the security requirements and controls system and cyber engineers need to add to these applications. There are a number of ways that system and cyber engineers can proactively combat cyberattacks, such as adhering to the software development practice known as *secure-by-design*, which requires developers to follow secure practices throughout the development lifecycle, instead of exclusively relying on reactive approaches such as sending out patches, to meet RTO and RPO objectives[7][9][14]. However, it is often unclear to these engineers and application owners whether or not their proposed security requirements and controls actually meet their RTO and RPO objectives.

One reason why it is often difficult to determine whether a system meets the RTO and RPO objectives is the communication gap between system engineers and cybersecurity engineers that makes it difficult not only to determine what security requirements are necessary for the system, but also to implement security controls on a system effectively. System engineers frequently struggle to understand how security controls will affect system performance, while cybersecurity engineers are often stuck trying to understand all the system's use cases and critical operations in order to determine what security requirements and subsequently controls are sufficient. Documentation of existing systems can also often be

difficult to understand, incomplete, or in dispute, as system engineers may have a hard time understanding how all the pieces of a massive, interconnected system fit together. To make it easier for both system and cyber engineers to document their work, communicate about requirements and how to implement them, and verify that requirements enable meeting specified RTO and RPO objectives, we borrow an approach from system engineers called *Model Based Systems Engineering* (MBSE). Unlike traditional documentation-centric methods, MBSE integrates and visualizes system components and their interactions into a version-controlled, shared, and unified model that can be viewed in multiple ways by the various stakeholders. The model reflects the current state of development and provides one central repository for all system information. As information is centrally managed, the model elements are interconnected, allowing automatic propagation of design changes, consistency checks, and error checks [10]. MagicDraw is an MBSE tool that enables the creation, analysis, and validation of system models to support complex system design and documentation. Despite MagicDraw's popularity with system engineers for designing cyber-physical systems, it is not commonly used to build secure software systems. Therefore, we leverage MagicDraw to create an MBSE plugin that would help fix this gap.

In this paper, we describe a framework that fits within an MBSE model to systematically improve and ease communication between cyber and system engineers, making it easier to design an architecture that is more "secure-by-design". By giving both parties a clear and intuitive way of understanding each other's problems and priorities, we can ease the process of crafting security requirements, specifying implementation details for security controls, and verifying their effectiveness through tests. Furthermore, all these tasks can easily be done using MBSE software, which has the added benefit of providing traceability - anyone looking at the model can easily trace a diagram detailing control flow for a security control to the requirement that necessitated it and to its tests for verification - and making explicit ties between security controls and system use cases. Verification tests are also intended to be used to ensure that the system meets specified RTO and RPO objectives, which the system and cyber engineers can communicate to application owners.

The overall framework for integrating cyber into MBSE is made in two parts. First, we discuss a Test-Driven Cyber Integration methodology, which allows application owners to ensure completeness in allocating cyber requirements for their system, as well as the ability to verify integration of secure technologies satisfying those cyber requirements. As the complexity of the system increases, so does the scope and volume of cybersecurity requirements. To support early-stage decision-making, we introduce the Risk Evaluation Assessment Plugin (REAP), a MagicDraw plugin that helps application owners perform rapid cyber risk assessments within their existing MBSE workflows. REAP enables users to explore the effectiveness of different combinations of security controls by simulating adversary behavior including lateral movement and system compromise and evaluating the resulting impact on RTO and RPO. By presenting actionable insights into which controls are most critical for resilience, REAP facilitates more informed and measurable security design decisions during system development. The contributions for this paper are as follows:

- A scalable method to assure completeness in allocating cyber requirements to a system
- A systems engineering approach for testing that cyber functionality satisfies requirements
- An initial method for dynamically assessing the cyber risk of the system as designs change, enabling system developers to better understand which security controls are crucial for meeting RTO and RPO objectives
- MBSE profiles and plugin, which aid the system engineer in performing the above

The rest of this paper reads as follows. In Section II, we discuss related works. In Section III, we discuss how to apply MBSE to software development so that application owners can build security into their applications. Next in Section IV, we discuss how to determine which security controls to prioritize. In Section V we discuss the MagicDraw plugin REAP, its design choices, and experiments on an example computer network with various security settings that can help system and cyber engineers and application owners understand the relationship between their security settings and the risks of adversary infiltration and data loss, allowing them to make informed decisions about which security settings to prioritize for their system. Finally, in Section VI, we conclude the paper and offer directions for further work.

II Related Work

A popular prior approach to integrate cyber with MBSE is that of *SysML-Sec* [3], which was developed for the design and development of secure embedded systems. The SysML-Sec approach has three phases: system analysis, software development, and validation. In stage 1, users co-design security and functional architecture, identify security concerns, and perform a risk analysis. Our approach differs in that it focuses on the effect of an adversary hacking into the system, rather than on targeting specific vulnerabilities. SysML-Sec models attacks using attack trees, but this approach becomes cumbersome when taking into account lateral movement; in contrast, our approach makes it easy to take into account an adversary chaining together multiple exploits. In stage 2, SysML-Sec asks users to make security requirements more precise and verify they're satisfied by the system design. Stage 3 uses formal methods to verify the confidentiality and authenticity properties. Our approach does not use formal methods; instead, our pipeline requires cyber and system engineers to specify functional tests for their security controls to verify their effectiveness. We touch on more about how we could integrate a formal methods approach in Section VI, when we offer directions for future work.

In [9], the authors propose an MBSE method that enables systems engineers and cyber security experts to start to incorporate threat mitigation strategies in the initial design phase. Where our approach differs from theirs is, again, our focus on the effect of an adversary hacking into the system, rather than on targeting specific vulnerabilities. This approach also asks developers to line up all the exploits needed to carry out the attack in order to map the threat to system architecture, whereas our approach allows us to model an adversary taking all possible routes through a network, as adversaries are likely to try multiple routes through a computer network in order to reach their goal.

In [8], the authors provide a method for modeling the propagation of failure through a system in the context of a Failure Modes and Effects Analysis, which is a structured method for identifying and addressing potential failure points before they occur. Their method is automated and provides results beyond the usual Failure Modes and Effects Analysis that helps support not just safety and reliability but also cybersecurity issues. They also provide a plug-in. Similarly to our approach, the authors consider all possible paths along which failure (used by the authors to refer to both failures and cyberattacks) can propagate. Where our approach differs from theirs is the amount of work needed to create the necessary models and metamodels before beginning the analysis. In their approach, the authors require that users create a metamodel that defines, among other things, what types of system components have failures, how these failures can propagate and transform, among other things. From there, users must add quite a bit of information to the block definition diagrams used to describe system architecture, for example, every possible input to a device that could lead to failure (two of the components in their example system both have nine different malicious inputs specified) and how every possible malicious output from one component could lead to a malicious input in a different component. REAP requires less work from the user: users need only specify potential methods of access for the components in their system, and REAP will model how malicious outputs from one component become malicious inputs to another.

In [5], the authors present an MBSE plugin specifically for cloud security architecture design. Their plugin, Security Pattern Synthesis, is designed to automate the Threat Analysis and Risk Assessment (TARA) process within cloud architecture design, and all its security recommendations are specific to cloud architectures only and no on-premise architectures. REAP, on the other hand, is designed to be generalizable to any system that can be modeled using MBSE software.

Finally, there have been several efforts to model the spread of computer viruses through a network similar to the way mathematicians and biologists model the spread of disease through a network, which is how we modeled lateral movement when developing REAP [4][11][6]. One key difference between our work and that of the aforementioned authors is that they focus on understanding under what conditions the system reaches equilibrium in the long term (no device changes from being compromised to uncompromised or vice versa), while our work focuses on understanding which devices either switch between being compromised and uncompromised over time or stay compromised the entire time.

III Test-Driven Cyber Methodology in MBSE

As mentioned earlier, MBSE is a popular choice for collating a platform's use cases, requirements, and parameters of success into a single cohesive model for a wide variety of stakeholders like software engineers and cyber engineers. MBSE approaches have shown success for tracing functional requirements to systems, but the cyber domain is underexplored in part due to a fundamental communication gap between systems engineers and cyber engineers.

It has been established that overall integration costs decrease when considering cyber security during the design phase of a system, a practice commonly referred to as secure by design [15]. However, there are still challenges during the design phase when it comes to defining cyber threats, mapping threats to system components, and applying cyber security mitigation strategies, especially in a way that keeps all stakeholders in agreement and coordination. To address these concerns, this section will discuss an overall methodology for addressing cyber security during system design using modern MBSE approaches. The end result of this methodology is to develop the derived cyber requirements as well as functional and integration tests for cyber technologies. We develop these tests so that we can verify the cyber requirements and develop these cyber requirements to establish overall security properties that can be proven correct about the system.

Secure-by-design is certainly not a new concept, nor is trying to use MBSE practices to do so, as can be shown in plenty of works [7][9][14]. This paper is not trying to reinvent secure-by-design best practices. However, what we note is that (to our knowledge) there has not been a proposed method that shows a fully traceable proof of secure design. Our paper is focused primarily on that front, especially in early-design of the system when defining security is particularly nebulous.

Defining cyber threats typically involves identifying a specific vulnerability (e.g., a bug in software running a user account database) and mapping it to an effect that causes harm to the system (e.g., the bug causes a user's password to leak). This definition has the implicit assumption of knowing precise implementation details about the system, which may not necessarily exist during the modeling and design phases. Therefore, we intend to focus more on the latter half: the effect that an adversary can use to cause harm to the system [15]. For this methodology, a cyber effect has effectively two parts: how does the adversary get in (i.e., a cyber kill chain of components) and the cyber-induced failure (i.e., loss of confidentiality, integrity, or availability in a component).

III.I Example Network and Methods of Access

To produce the cyber kill chain, we need to look at the various ways an adversary can get into the system. In Figure 2 we have what is called an *internal block diagram*, or IBD, that shows the various methods of access each of our devices on the example network has (this is the example network on which we will run all simulations in Section V). For example, a laptop on a network can be composed of Software (applications and an operating system), Hardware (workstation and removable media), and Subsystems and Data Busses (Wi-Fi, Bluetooth, etc.). Depending on the use of that laptop, it could also have Maintenance and Debrief Data if it is a maintenance laptop, or Mission Plan and Operations Data if it is part of the network's day-to-day operations. These tangible and intangible components are from the Wheel of Access, which itself is from the "DoD Cyber Tabletop Guidebook" (shown in Figure 1).

The benefit of utilizing the Wheel of Access in our approach is that it makes our approach generalizable. Utilizing the Wheel of Access allows us to focus on particular methods of access an adversary might use to gain entry into a system component, like data or voice links, instead of specific threats that are tied to specific software vulnerabilities or specific pieces of hardware. By keeping our focus on something more abstract, we can easily use this framework across many different kinds of systems without having to

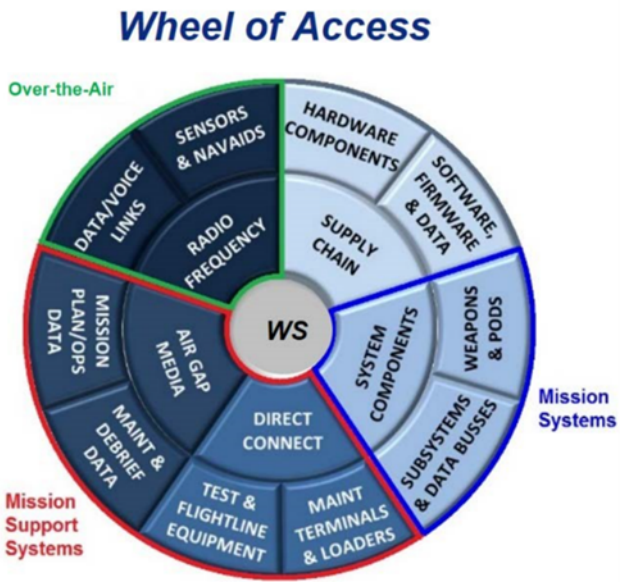

Figure 1: Wheel of Access from the "DoD Cyber Tabletop Guidebook". "WS" stands for "Weapon System." System block diagrams are decomposed into a subset of the components on the outer ring (e.g., hardware components, data/voice links, etc.).

list out specific threats for each new system or change the threats we focus on with changes in implementation details or software installations. Keeping our focus more high-level rather than fine-grained also allows us to start thinking about cybersecurity in early design phases, when implementation details might not exist yet or are subject to change. It also enables us to focus less on *exactly how* an adversary gains access, since that space is theoretically infinite as threat actors are constantly looking for new ways to gain access, but more on the fact that *they did* and what security requirements could be useful in preventing it.

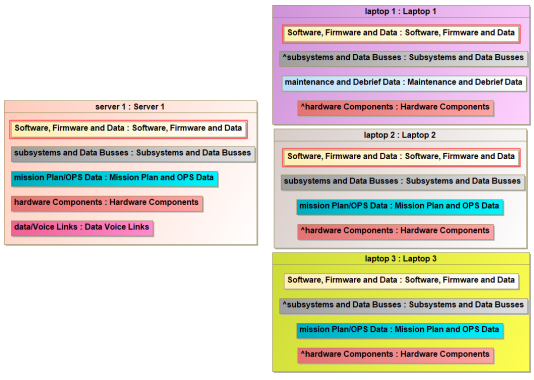

Figure 2: Example network with systems decomposed into the Wheel of Access abstractions.

III.II Allocating Security Requirements System-wide

We represent cyber-induced failures simply by applying cyber properties to each *block definition diagram*, or BDD, which is shown in Figure 3. We use this diagram to clearly lay out how all devices in the network are connected to each other; we use *blocks* to represent the devices, one block per device. We will refer to this diagram and discuss it in greater detail in Section V when we talk about the role this diagram plays in REAP simulations. To each block, we add *value properties*, which in MBSE are used to specify quantifiable properties about a particular block, representing Confidentiality, Integrity and Availability. For our approach, it makes sense to represent Confidentiality, Integrity, and Availability as value properties,since we will use binary values to indicate their status (i.e. 0 indicating a failure in the associated cyber property).

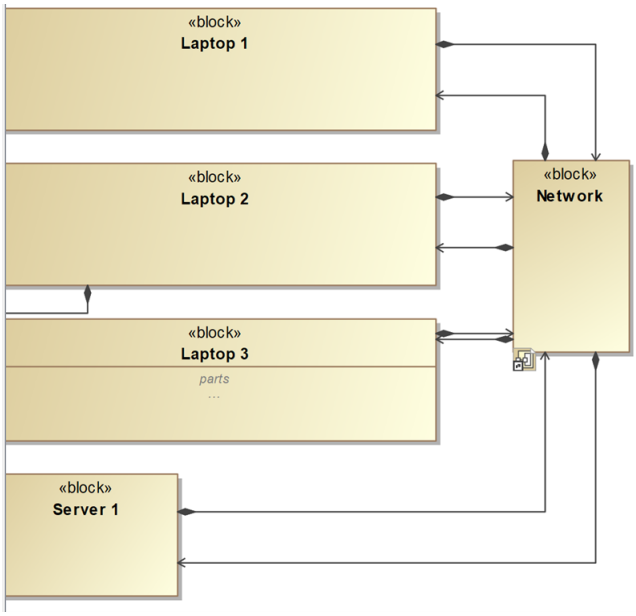

Figure 3: Example BDD of the network layout.

To start mitigating against possible cyber effects, our approach applies security controls to each type of access from the Wheel of Access. The logic here is that, while it is intractable to claim that a block component is absolutely secured, we can have a process to verify that a block has all of the recommended security guidance applied to it. In this work, we used the Risk Management Framework, or RMF [2], as an example suite of cyber requirements. An *allocation matrix* is shown in Figure 4 that shows the allocations of the RMF controls to the Wheel of Access component abstractions. The rows in this allocation matrix represent the RMF controls, while the columns represent the access methods. The arrows in the cells indicate that the RMF control in that row is a security requirement for the particular access method in that column; in other words, the control has been *allocated* to that method of access. What falls out of this allocation is a scalable approach to applying security requirements to the system.

We leveraged the security requirements set out in the RMF to provide a set of controls for the example network that we analyze later in this paper. We discuss this in more detail at the end of the paper, but note that users are not subject to utilizing RMF for their security controls. Any set of security controls that address the various methods of access as sufficiently and completely as possible could work. By allowing for this freedom of choice, users can pick the sets of security controls most relevant to their systems, increasing the versatility of our framework. We also note that REAP's current functionality utilizes RMF controls in its design, but we are hopeful that in future iterations of REAP we can allow for more freedom of choice in security controls (more discussion on this in later sections).

As a system grows in size and complexity, and system developers add new blocks to represent new devices or change blocks to represent changes in devices, they will continue to specify for each new or modified device potential methods of access from the Wheel of Access. The RMF security controls are automatically applied to the methods of access, which means they're automatically applied to the new devices (since the methods of access are constituent parts of the device). This automatic allocation happens as a result of the allocation matrix we created earlier and is what allows us to easily apply security requirements to our entire system. Our approach is one that scales well with system growth.

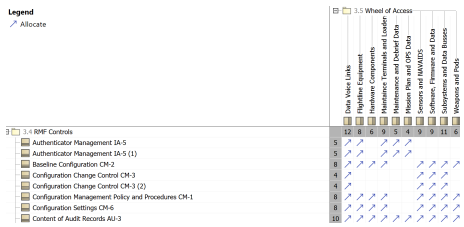

Figure 4: Example allocation of RMF security controls allocated to Wheel of Access abstractions.

With RMF controls automatically allocated to the methods of access on each block as they are defined, we can ensure that there is a complete coverage of security controls and requirements applied to system components. The next step is to verify that these requirements can be implemented in the system. As shown in Figure 2, the laptops and server in our example network have all been given software and hardware components. All of these components have been allocated RMF Control CM-02 Baseline Configuration as per Figure 4. This control is partially defined as: "Baseline configurations for systems and system components include connectivity, operational, and communications aspects of systems. Baseline configurations are documented, formally reviewed, and agreed-upon specifications for systems or configuration items within those systems." The implementation of a baseline configuration for the server's hardware will be a different implementation than that of the laptop's software. The way to address this is through unit tests at the component level. Modern software development commonly drives toward passing functional tests as it is a scalable and robust way to ensure the system can pass functional requirements. We propose integrating system security design into current practices of a test-driven approach to also meet the security requirements of the system so that security design also gets these benefits.

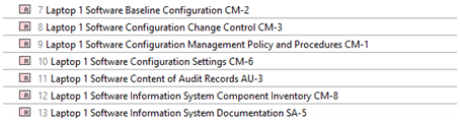

Figure 5: Example derived cyber requirements for Laptop 1's software components.

A simple example of a cyber test can be shown in Figure 6, which shows a sample test for Laptop 1. In this case, the test is to verify the requirement that Laptop 1's software has a baseline configuration. The behavior of the test would start with a central authority ("Manager" in the example) that would request the software baseline of Laptop 1. Laptop 1 would respond with a list of all software and their version numbers currently installed. The central authority then compares that list with an approved set of software. If Laptop 1's software maps to the approved software, then the test is successful and the requirement can be verified.

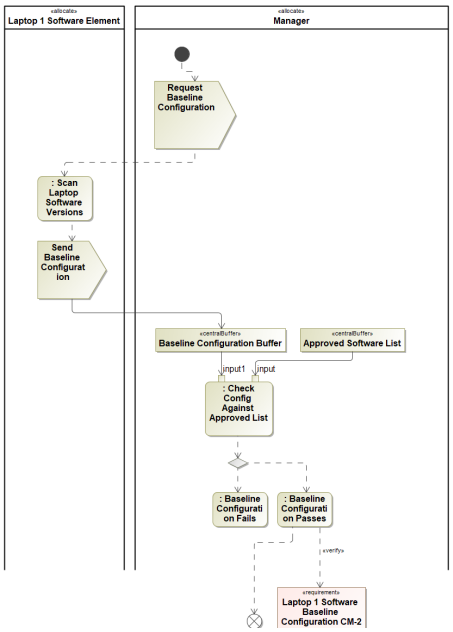

Figure 6: Example cyber test that verifies the requirement for Laptop 1's software to have a baseline configuration.

The benefit of systematically going through this test-driven approach to security is that it assures (1) completeness in cyber requirements by having system components decompose to a core set of abstractions, and (2) the requirements can be verified through the tests in a more systematic way. However, manually going through this process is tedious and would involve too much added work. The aid of MBSE allows for easier allocation of security requirements, integration into the system as components are being defined, and reapplication of security tests when there are similarities. Although

this approach guarantees completeness, it is unclear for a system engineer where to begin with the integration and design of security requirements. The next section will discuss a way to perform a criticality analysis of components in the system so that a system engineer can better prioritize which requirements to address first.

IV Cyber Criticality Analysis

With the prior section establishing a means to derive a complete set of cyber security requirements for a system, the next problem arises: which requirements should be addressed first? This question becomes increasingly difficult to answer as the systems grow in size and complexity. This section aims to start answering that question by providing a method for comparing how effective different sets of security controls on different devices are at meeting defined RTO and RPO goals. It provides a useful, easy alternative to manually digging through the system model with other system developers or cyber-engineers weighing different security controls against each other. As the process is described, we highlight ways in which MBSE approaches can help streamline this assessment and allow for dynamic re-assessment as the system design, system use cases, or threats to the system change throughout the design life cycle.

Our approach to analyzing cyber impact to a networked system is by modeling the spread of viruses (or adversaries) through the computer system similar to how mathematicians and biologists model disease spread through human populations; in Section II we discussed similar approaches and where ours differ. When modeling disease spread, it is assumed that each person in the network has some probability of infecting the people they interact with, so the disease moves through the network by copying itself from person to person [12]. In a similar fashion, adversaries create cyber-kill chains by moving laterally through a system and infecting a series of components and hosts. By running simulations of a threat moving laterally throughout the network by jumping from device to device, we can simulate an adversary chaining together multiple vulnerabilities, or creating a cyber kill chain, by exploiting a vulnerability in one device to get to another. In many cases, an adversary gets access to critical components through vulnerabilities found in tertiary components, which is not immediately obvious to the system designer at a glance.

Our work gives users a way of comparing the impacts different security controls have on their system by providing users data from simulations of adversaries successfully or unsuccessfully compromising network components, some of which have security controls and some of which do not, some of which will get patched and some of which will not, on a level close to day-to-day. By adjusting the presence of security controls on various components in the system and the probability of successful patching and comparing the data between different simulation runs, users can start to make decisions about which security controls to prioritize for their system and which components to add security controls to. We will provide an example of six different sets of simulation runs on our example network, their results, and a discussion of the results in Section V.

V Risk Evaluation Assessment Plugin (REAP)

The analysis approach we take will assume that the probability of compromise is the same for all types of access from the Wheel of Access shown in Figure 1; that is, an adversary is not more likely to choose one method of access over any other. The argument is that, since the system being analyzed is still in development, it is infeasible to appropriately weigh one method of system access as "easier to corrupt" than any other. Once the system has begun its implementation, a system engineer could modify this methodology by applying weights based on vulnerability assessments and threat intelligence as appropriate to increase the efficacy of this approach.

V.I Design Choices

The primary functionality that REAP provides is the ability to run simulations of a cyber adversary attempting to gain access to the system and move laterally within it. The core features of these simulations, which are designed to run over multiple timesteps (for ease of comprehension, each timestep represents one day), are:

(1) Assigning a probability p that the adversary will compromise any component that currently has not been compromised.
(2) Assigning a probability q to model events such as patch updates or IT kicking adversaries out of certain devices; compromised devices that recover can become compromised again. This probability is a parameter that REAP users can set before the simulation runs start.
(3) Deciding which sets of security controls to implement in which system components; the more security controls users add, the more p decreases. Users decide which components have controls, if any, before the simulation runs begin.
(4) Deciding what the RTO and RPO objectives are (for example: only 1 day of data loss is acceptable, or each component must recover within 2 days).
(5) Flipping a (potentially weighted) coin at the start of each timestep (each day) of the simulation to see if any compromise attempt of any component is successful.
(6) Seeing which components have been infected and changing scope dynamically to emulate lateral movement of an attacker as they propagate cyber effects throughout the system.

We note that there are elements of adversary behavior, recovery mechanisms, and network topology and features that our model currently does not capture, as the focus of our analysis is to simulate adversaries moving laterally through a network by lining up exploits and how the addition of security controls and patching can inhibit that movement. Adding, for example, more complicated recovery mechanisms that acknowledge how adversaries can still run exploits even without access to components they previously compromised adds a layer of complexity that would be interesting to explore in future iterations of REAP but would detract from what we want to focus on in this paper and also make it more difficult to compare the results of different simulation runs. With these core features in place, we can apply security controls and requirements to individual component blocks to reduce the probability that the block becomes compromised. REAP provides users with data at the end of all simulation runs, which shows which blocks had what controls and which blocks got compromised and how quickly, how frequently, and for how long. Each time a block gets compromised, its data is lost, which will affect the system's ability to meet RPO thresholds. We also keep track of how many

days blocks are compromised and how many times, which allows us to determine whether or not our system is able to meet the RTO objectives. We explain how we produce these metrics in the next section.

V.II REAP Simulations

One benefit of running simulations with REAP is that we can change many aspects of our system – like the probability of successful patching q – and see how our end results change, allowing us to consider a large variety of threat and recovery models. This flexibility allows us not only to compare infection rates and duration in a number of different situations, but also to focus on the impact threats and security controls have on our system, rather than the specifics of how a cyber threat or security control works. REAP makes it easier for application owners to understand why certain cyber threats are dangerous and to compare how different security controls can mitigate those threats. The initial functionality of REAP is to start allowing system engineers to see the immediate effects cyber security controls and technologies have on the system design. REAP was built to be modular, so it would be completely feasible to include more sophisticated threat models and defense models as the knowledge about the system grows. This paper aimed to show an initial design of the plug-in.

During simulation runs, REAP keeps track of which devices were compromised on which day and on which day they recovered (if at all). At the end of the runs, users will get data on the status of every device every day, as well as summary statistics regarding the number of devices compromised, the number of days each of the devices were compromised, how many times those devices were patched, whether or not the RTO and RPO objectives are met, and more. As such, by running simulations under multiple conditions – from no security controls on any block to every possible security control on every block, and from low patching success rates to high success rates – users can use the summary statistics to compare different sets of security controls on different blocks. These summary statistics will help users decide which security requirements and components to prioritize. Since users will also get raw data, they can also compute their own statistics for further analysis. We will provide a detailed explanation of exactly how we run these simulations along with the results of running REAP on the example network shown in Figure 3, complete with summary statistics, later in this section. On the right, we have a block defined for our network as a whole, and immediately to the left, we have 1 block for each of the devices that constitute our network: Laptops 1, 2, and 3 and the server.

REAP uses BDDs to build a graph so we can run simulations. We think of each block diagram as a directed graph, where each device is a node, and an edge from device A to B means that A can talk to B. In our example model, the server and three laptops all have edges pointing toward the network node, and there are also edges pointing from the network to each device, as we assume all these devices can talk to the network and vice versa.

For the probability p that any individual device gets compromised, the presence of security controls is the only factor that can decrease it. Currently, each device has three possible sets of security controls that the user can implement. For every set applied to the device, p decreases by 80%. We note that these numbers are not based on actual data; see Subsection V.IV as well as the discussion section for more details.

Once we have this directed graph, we can begin our analysis. As mentioned previously, each simulation run consists of multiple timesteps (for ease of comprehension, we imagine that each timestep is analogous to a day); the idea is that on each day the adversary will attempt to compromise all nodes to which the currently connected to compromised nodes with probability p. As an example, if Laptop 1 is compromised, then the adversary will attempt to compromise Network. If the node that the adversary is trying to compromise has no security controls, the probability of successful compromise is p; if the node has security controls, then the probability is less than p. This phenomenon models the adversary crawling through the network: an adversary cannot see the whole network, they can only see the neighbors of the device they are currently compromising, and they use this information to move through the network. As the adversary can only see neighbors, REAP adjusts the scope of what nodes the adversary can access based on what node it is currently compromising (if the adversary was compromising Laptop 1 its scope would be limited to just Network, but if's currently compromising Network its scope would be all remaining devices). The nodes in our network the adversary successfully compromises are known as *infected* nodes while all remaining uncompromised nodes are called *susceptible* nodes. Each infected node has the same fixed probability q of recovering (that is, a system administrator applies a patch that fixes a known vulnerability) and immediately transitions back to the susceptible group (where an adversary can compromise it again). This model of nodes moving between susceptible and infected is known as the Susceptible-Infected-Susceptible (SIS) model [12].

V.III REAP Inputs

Before we begin running simulations, we first specify the entry points for the adversary: which devices can get infected on day 0? The user can choose any number of devices to be entry points. The user also specifies the number of timesteps (days) to run the simulation for and how many instances of those simulation runs they want (for example, a user could specify 10 instances of 100 days each, which we interpret as allowing an adversary to run amok in our system for 100 days, ten separate times). We also specify the value for q and how many RMF controls we want to apply to our system and on which devices. Lastly, we set the RTO and RPO thresholds (max number of days to return to service and maximum acceptable data loss) and data backup frequency (for example, once every day, once every 10 days, etc.). We note that every time a device is compromised, its data isn't backed up, and so the count of how many days of data loss that device suffers from begins there - if a device remains compromised for 12 days, recovers for 29, and then is compromised for another 10 days, REAP records its total amount of data loss as 12 + 10 = 22 days' worth. REAP provides a GUI that allows users to specify the values of these parameters.

We began each simulation on day 0 where each entry point specified by the user has probability p (if it has no security controls) or probability less than p (if it has security controls) of starting off infected (recall that for every set of security controls applied to

a component, its probability of infection p gets multiplied by 0.2 since each set of controls leads to an 80% reduction). From there, what happens on subsequent days is the same as with the SIS model of disease spread that we described earlier: each infected device will try to infect other susceptible devices with probability p or less than p. For all experiments, we used REAP on an example network model shown in Figure 3.

V.IV Sensitivity Testing and Risk Measurement

We will discuss each round of simulation results separately and provide the rationale for how we chose the inputs for each.

V.IV.I Round 1 of Experiments. In Table 1 we have provided a list of all inputs used for the first round of experiments that we performed. These input values were chosen to test a wide range of input values and are not based on any actual threat or recovery models; as in early design phases with limited implementation details or specifications, it is infeasible to accurately estimate, for example, how successful patching could be.

The first three batches were used to simulate a wide variety of patching efficacy rates; the middle three were used to compare the effects of varying how many sets of security controls we add to our system, while applying it uniformly to all devices; and the last was to see how infection rates and our ability to stay below RTO and RPO thresholds are affected when we only give some devices security controls. For all 9 experiments, $p = 0.5$, Laptop 1 was the sole entry point, there were 30 runs of 300 days each, RPO = 20, RTO = 10, and the data backup frequency was once every 10 days.

We set the value of $p = 0.5$ for all these experiments so that we could test out a wider range of patching success rates q in the hopes that each value of q would provide different results: if p was closer to 1, then there would probably not have been much difference between $q = 0.3$ and $q = 0.5$. Similarly, if p were closer to 0, then there would probably not have been much difference between $q = 0.5$ and $q = 0.8$. Setting p at 0.5 makes it easier for us to have one value of q much higher than p (modeling the scenario in which we are able to successfully kick the adversary out of our system frequently), one value of q where they're equal (modeling the scenario in which our patching success rate is not as effective), and one value of q much smaller than p (modeling the scenario in which our patching success rate is low enough that it might not be effective at all). In future work, we would ideally base p on actual data or test a much wider range of values. We acknowledge that only testing one value of p limits adversary behavior, which limits the generalizability of our results.

As mentioned in the previous section, every time we add a set of security controls to a block, its probability of compromise decreases by 80%. This specific value was chosen to reflect how stringent and comprehensive RMF controls are and is not based on actual data. Trying to quantify and subsequently measure how much cyber risk decreases with the presence of security controls is an area of active research.

It is beyond the scope of this paper to determine appropriate input values. We encourage REAP users to consider threat and recovery models that make sense for their systems and also note that as system-specific implementation details emerge, they can also use that information when setting parameter values. REAP is designed to allow for a wide range of possible assumptions that a user would employ in setting parameter values.

Table 1: Table of initial experiments done, p = 0.50, Laptop 1 is sole entry point, 30 runs of 300 days each, RPO = 20, RTO = 10, Backup freq = 10

Scenario	q	Sec Ctrls	Device w/Ctrls
Default	0.50	0	None
Best Patch	0.80	0	None
Worst Patch	0.20	0	None
1 Set	0.50	1	All
2 Sets	0.50	2	All
3 Sets	0.50	3	All
3 Sets, Entry	0.50	3	Laptop 1
3 Sets, Critical	0.50	3	Network
3 Sets, Both	0.50	3	Laptop 1 and Network

Now we will discuss some results for this initial set of 9 experiments. Table 2 contains the average recovery time and the average number of days of data loss for the four experiments that we will discuss in this subsection. The values highlighted in pink are values that exceeded thresholds; the values highlighted in light green are values that stayed below or met thresholds.

In *Default*, all the devices in our network exceeded both the RTO and the RPO thresholds at least once during most simulation runs, with the exception of the entry point Laptop 1, which exceeded the RTO and RPO thresholds at least once during *every* simulation run. In particular, we note that across all 30 simulation runs, Laptop 1 took on average 98.9 days to recover after getting compromised, and the average of the total number of days of data loss it suffered was 217.8 (Laptop 1 was compromised multiple times per simulation run with brief recovery periods in between, which is why the average total data loss is greater than the average recovery time). We note that even though each device had a 50% chance of being patched every day, the fact that there were no controls at all meant that every device also had a 50% chance of being compromised (and thus losing data) every day. Plus, once a device is compromised, it tries to infect *all* of its neighbors every single day that it is compromised, which means once Laptop 1 is able to successfully infect the Network block, the adversary now has access to every device for as long as it is able to successfully infect Network since Network is connected to all components.

Compare these results to *3 Sets, Entry* where we added all possible security controls to the entry point Laptop 1 and none anywhere else. All other parameter values are the same as in *Default*; also recall that for every set of security controls added to a device, its probability of infection decreases by 80% (or it gets multiplied by 0.2), so because Laptop 1 has all three sets, its probability of infection has decreased to $0.5 \times (0.2)^3 = 0.004$. In this experiment, Laptop 1 was still compromised, but far less frequently and for far shorter lengths of time. In fact, it met the RTO and RPO thresholds *throughout every simulation run*, since on average it only took 1.667 days to recover from being compromised and on average only suffered 4.4 days of data loss! For Network, it was only compromised for an average of 0.433 days across all simulation runs and only suffered an average of 1.4 days of data loss. These two experiments

make a strong argument for the value of having security controls on even one device in a system, even if the success of patching is only 0.5.

What if, instead of adding security controls to our device, we decide to simply improve our ability to kick the adversary out of a device by improving the success rate of our patching? That is what inspired the experiment *Best Patch*, where the probability of successful compromise p is 0.5 and the probability of successful patching q is 0.8. However, we can see that Laptop 1 suffers greatly if we only rely on patching - it takes on average 34.4 days to recover from compromise and loses an average of 125.9 days of data. Even though Network fared significantly better, with an average recovery time of 5 days and an average of 18.2 days of data loss, its averages are still an order of magnitude larger than Network in the prior experiment *3 Sets, Entry*. We speculate that the reason for the large discrepancy between Laptop 1 and Network in *Best Patch* is due to a design choice we made for REAP: for any device, if it gets compromised on day n, we first deploy a patch with probability of success q and then the device tries to infect its neighbors if patching was unsuccessful (this design choice is modeling the scenario in which someone notices something strange on their computer and alerts IT before the adversary is able to infiltrate another device). So, it is possible that the patch deployment fails, it tries to infect each of its neighbors with probability p, and some number of its neighbors are infected on day $n+1$. It is also possible that patching is successful and that it never gets the chance to try to infect any neighbors, so none of its neighbors is infected on day $n+1$. Thus, because the probability of successful patching q is 0.8, it is likely that there were many more instances in which Laptop 1 got infected and recovered on the same day without ever infecting any of its neighbors than instances where Laptop 1 got infected, failed to recover, and subsequently tried to infect its neighbor Network. What this experiment shows us is that patching can be effective in preventing lateral movement (assuming patches go through before the adversary is able to infiltrate more devices), but patching is not particularly effective in preventing the adversary from successfully compromising entry points.

Lastly, we will consider the case of only applying one set of security controls to every device. How does that affect the end results? Examine *1 Set*. We can see that compared to having all security controls just at the entry point and nowhere else (the experiment *3 Sets, Entry*), it takes much longer on average to recover from the compromise, and it also suffers more days of data loss on average. However, we note that the difference between Laptop 1 and Network in this experiment is much greater than the difference between Laptop 1 and Network in *3 Sets, Entry*: we can see that when we have all 3 sets of security controls on just the entry point, Laptop 1's average recovery time and amount of data loss is between three and four times greater than Network's. However, when we have 1 set of security controls on all devices, we see that Laptop 1's average recovery time and amount of data loss is about *ten times* greater than Network's - which means we saw much less lateral movement! In *3 Sets, Entry* about 25% of the adversary's attempts at infiltrating Network after gaining entry through Laptop 1 were successful, but in *1 Set* only about 10% of the adversary's attempts at infiltrating Network after gaining entry through Laptop 1 were successful. Of course, based on these simulation results, *3 Sets, Entry* is still preferable to *1 Set*, but *1 Set* shows the importance of having security controls at more than just the entry point! It is very effective in preventing lateral movement.

Table 2: Average Recovery Time (RT) and Total Data Loss (TDL) Measured in Days

Scenario and Device	Avg RT	Avg TDL
Default, Laptop 1	98.9	217.8
Default, Network	48.7	109.4
3 Sets, Entry, Laptop 1	1.6	4.4
3 Sets, Entry, Network	0.4	1.4
Best Patch, Laptop 1	34.4	125.9
Best Patch, Network	5.0	18.2
1 Set, Laptop 1	30.56	67.5
1 Set, Network	3.83	6.8

Table 3: Results for *3 Sets, Entry* With Different Patching Success Rates

Scenario	Average RT - Laptop 1	Average RT - Network
$q = 0.5$	1.6	0.4
$q = 0.4$	1.7	1.5
$q = 0.3$	2.0	4.2
$q = 0.2$	4.1	22.0
$q = 0.1$	7.9	160.7
$q = 0.05$	21.3	251.8
$q = 0.01$	65.0	269.6
$q = 0.0$	153.8	297.3

V.IV.II Round 2 of Experiments. We notice that *3 Sets, Entry* with a patching success rate q of 0.5 led to great results (recall that in this set of experiments, the probability of a successful compromise p was also 0.5). What if we had all the security controls on Laptop 1, kept all other parameters the same, but our patch success rate q was lower than the compromise success rate p? Do we still stay below our RTO and RPO thresholds? This question led to our next set of experiments, where we recreated *3 Sets, Entry* with a variety of values for patching success rates q, all less than 0.5. The q–values we tested can be seen in the first column of Table 3.

We note that the first row of Table 3 shows the results from the experiment *3 Sets, Entry* from Table 1. We added it to this table for ease of comparison. Although we initially decided to only decrease the patching success rate q in increments of 0.1, after discovering that $q = 0.5, q = 0.4, q = 0.3, q = 0.2$, and $q = 0.1$ all yielded low values for the average recovery time for Laptop 1 (we always stayed below the RTO threshold), we decided to try more values between 0.1 and 0. Note that in all of these experiments, the probability p of Laptop 1 getting infected is $0.5 \times (0.2)^3 = 0.004$ while the probability p of Network getting infected is still 0.5 since it has no security controls.

Notice that for Laptop 1, there is a remarkable jump in the average recovery time when going from $q = 0.1$ to $q = 0.05$. In fact, the average recovery time either triples or almost triples every time

we decrease the value of q starting at $q = 0.1$! This seems to be an inflection point for Laptop 1. Notice that for Network, its inflection point occurs when going from $q = 0.3$ to $q = 0.2$. It makes sense that there is an inflection point for both devices - that eventually, even though Laptop 1 has all three sets of security controls, the adversary is ultimately able to compromise both devices for an amount of time that far exceeds what has been deemed acceptable.

The way our model works is that if Laptop 1 and Network are infected on day N, then we first try to deploy a patch to both of them. If q is quite small, then it is possible that the patch only works for Laptop 1 and not for Network. But later *on the same day*, Network will try to infect its neighbors (since it failed to recover) - which includes Laptop 1! This means that Laptop 1 can recover and become compromised again on the same day. This phenomenon explains why, for sufficiently small patching success rates, Laptop 1 and Network just stay infected for long periods of time - they are constantly reinfecting each other! It also makes sense that Network suffers from higher recovery times than Laptop 1 much earlier on (its inflection point comes much earlier, for higher patching success rates): Network is connected to three other devices, so suppose Network is infected on day N, fails to recover (which, again, is likely since q is small), then infects Laptops 2 and 3. On day $N + 1$, if Network recovers but at least one of Laptops 2 or 3 don't, then one of Laptops 2 or 3 can now reinfect Network! In other words, this constant back and forth between recovering and getting reinfected happens much more frequently with Network because it is connected to more devices than Laptop 1! Laptop 1 can only be reinfected by Network.

Further investigation is needed to determine *why* the inflection point Laptop 1 is at $q = 0.05$ specifically, while the inflection point for Network is at $q = 0.2$ specifically.

What these results show us is the importance of patching in addition to security controls: otherwise, we see a constant cycle of recover-compromise-recover-compromise as devices reinfect each other. Security controls are needed to prevent the cycle from ever occurring in the first place; patching is crucial to breaking this cycle once it begins!

V.IV.III Round 3 of Experiments. Ideally, a system would have a low RTO value - its maximum allowed recovery time should be as low as possible, as otherwise a compromised device is online for much longer and has more time to compromise other devices. For five of our experiments, *3 Sets, 2 Sets, 1 Set, None,* from the first round and *3 Sets, q = 0.2* from the second round, we decided to investigate how low we could set RTO thresholds for each experiment and still have a majority or even all of the simulation runs stay below this threshold. For example, in *3 Sets* we know that Laptop 1's recovery times were in 10 days or less for all simulation runs - but how many simulation runs recovered in 8 or less? Or even 3 or less? These are the questions we are attempting to answer; see Figure 7 for the answers to these questions, which we will unpack in this section.

To be clear, we did not rerun the simulations; we simply calculated, for Laptop 1 in each experiment, what percentage of simulation runs recovered in 10 days or less, what percent recovered in 9 days or less, what percent recovered in 8 or less, so and so forth to what percent recovered in 2 days or less and finally what percent recovered in 1 day.

We can see for the experiment *3 Sets* that if we set the maximum recovery time to just 1 day, then only in 50% of the simulation runs did Laptop 1 recover within 1 day. Once we increase it to 5, however, we see that in well over 95% of the simulation runs Laptop 1 recovered in that time frame - which indicates that at least for Laptop 1, we could consider lowering the RTO threshold to 5. We would need to investigate the recovery times for the other devices in this experiment to consider lowering the RTO threshold for the entire system.

For the experiment *2 Sets* we see that in only 20% of simulation runs did Laptop 1 recover within 1 day. If we set RTO to 9, we do fare quite a bit better - in around 90% of simulation runs Laptop 1 recovered within that time frame. This result makes sense: fewer security controls means a higher probability p of infection on any given day, which means that if Laptop 1 can stay infected (for *3 Sets* and *2 Sets* the patching success rate q was 0.5) it can infect its neighbor Network, also with higher probability p of infection, and Network can reinfect Laptop 1 if it stays infected (the recover-compromise-recover-compromise cycle).

For both *1 Set* and *0 Sets*, there are no simulation runs that stay below any of the RTO thresholds (the purple and pink line on the graph overlap) as shown in Figure 7.

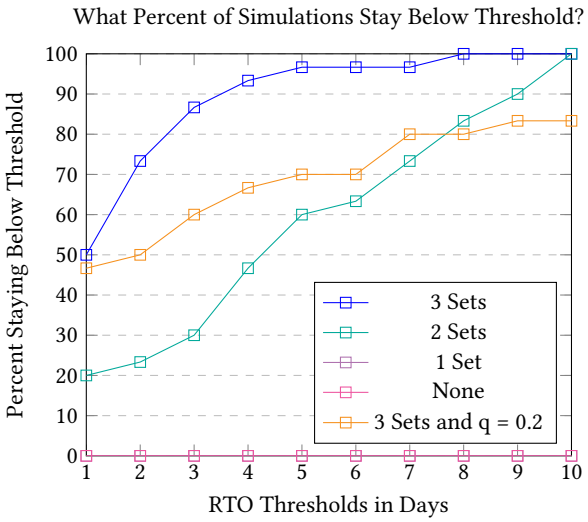

Figure 7: Plotting RTO Compliance versus RMF Controls

Lastly, consider *3 Sets and q = 0.2*. Unlike in *3 Sets* and *2 Sets*, we never reach a point where 100% of our simulation runs stay beneath a certain maximum recovery time - at best, we are slightly above 80%. The lower patch success rate $q = 0.2$ and the fact that only 1 device, the entry point Laptop 1, has security controls means that whenever Laptop 1 successfully infects the Network, it can easily reinfect Laptop 1 whenever Laptop 1 recovers, since the Network is not particularly likely to recover. Also, Network can easily infect its other neighbors, its other neighbors are less likely to recover and so they're more likely to reinfect Network, which can then reinfect Laptop 1. It is again the recovery-compromise-recover-compromise cycle.

V.V Discussion

Our approach is proposed as groundwork to enable a more seamless integration of cyber as an engineering practice—especially during the start of a system design—to better enable the vision of *secure by design*. This section will add discussion points to the strengths and limitations of the current proposed approach:

Better Cyber Requirements - Cyber requirements as presented to a systems engineer are often opaque and speak more to the aspirations of security properties (i.e., "the system shall have a baseline configuration"). By driving the requirements down to individual components, our approach starts to make cyber requirements more tangible for engineers, without necessarily needing a proficient background in security. This approach also drives more toward a complete set of cyber requirements, making it easier to adhere to frameworks like RMF or meet requirements laid out from government regulations such as HIPAA. The derived nature of requirements for each component makes for an easier assessment of what has and has not been addressed.

Provable Security Properties — Addressing cyber requirements in the form of tests starts to build towards strong assertions of security properties with provable and repeatable results. A system engineer can concretely say, for example, "this system maintains integrity by meeting these access control requirements, and these tests show proper isolation is occurring." However, this work currently does not claim to build requirements into higher-level security properties through, say, a formal methods approach. Significant work must be done, ideally with formal proofs, to be able to make strong claims about specific requirements meeting security properties.

Easier Integration with Cyber Teams - A test-driven approach produces a natural set of metrics: how many cyber requirements has the system already met and how many are left to do? How many verification tests have been passed? These statistics significantly ease communication with other development teams (e.g. a cyber red team) so that they can effectively address the validity of the test's assertions (and in the case of the cyber red team, see if they can exploit security controls that lack verification tests or unimplemented security controls).

Simple Heuristic Approach for Cyber Risk Assessment - REAP provides a system engineer with the functionality to get heuristics and other statistics about the efficacy of security controls in the system design—making it easier to start prioritizing which security controls to adopt first. However, work needs to be done to improve the algorithms and calculations performed in this approach. For example, an 80% reduction (multiplying p by 0.2) in the probability of compromise after adding security controls is possibly too high of a reduction, and our simplifications of adversary behavior might miss key elements of how adversaries actually chain together vulnerabilities. By doing more research on how we measure cyber risk and studying how adversaries have behaved in the past, we can gain more insight into how to refine our model to more realistically represent an adversary chaining together vulnerabilities. However, we note that the threat space is theoretically infinite; adversaries are always finding new ways of chaining together vulnerabilities to hack into systems, so our threat models will never fully encapsulate adversary behavior.

Confirming BCP Requirements - REAP can help confirm if an application can meet its RPO and RTO requirements. The number of days compromised should ideally be less than expected RTO, and if it is not, REAP can help provide insight on how to meet RTO compliance, as users will know exactly how much they fall short of meeting said requirements. When it comes to complying with RPO requirements, since REAP provides results for how many devices are compromised each day of the simulation, using assumptions about how frequently data is backed up (say every 20 days), users can easily compute how many days of data loss the system suffers in simulation runs.

VI Conclusion and Future Works

In this paper, we address the needs of application owners who manage critical applications that require extensive security controls. We introduced an overall framework for integrating cyber into the early design phase of a system using MBSE approaches. We first discussed an approach to assure completeness in allocating cyber requirements to the system and verifying functionality using a Test-driven Cyber Integration methodology. We then demonstrated the Risk Evaluation and Assessment Plugin (REAP) and its ability to dynamically reassess cyber risk within a system model, helping prioritize the integration of appropriate cybersecurity functionality. This framework and tool helps application owners evaluate whether their design aligns with business continuity planning (BCP) objectives, such as RTO and RPO. In cases where requirements are not met, REAP's detailed summary metrics and simulation results provide actionable guidance, making it easier for application owners and system developers to identify and implement targeted design changes that improve system resilience.

There are many avenues for future work when it comes to the Test-Driven Cyber Integration framework. An actual application of our framework from the beginning to end of system design is needed to ensure it does ease communication between cyber engineers and system developers, allow for a complete coverage of cyber requirements, and lead to a test driven approach towards cyber. Ideally, testing our framework on a variety of cyber-physical systems would yield the greatest results. Replacing the cyber requirements from RMF with a different set of cyber requirements - say ones required by HIPAA - would also be interesting and also demonstrate the versatility and generalizability of our framework. Finally, since MBSE currently does not integrate with code, more work is needed to determine how to smoothly ensure that software developers are able to effectively use MBSE to build their systems following secure-by-design best practices.

There are many avenues for future development work on REAP. Allowing for more choices in what security controls users can apply to their system in simulation runs beyond what REAP currently offers - the controls suggested by RMF - would allow for much greater applicability and flexibility, making REAP more versatile and appealing to a wider audience. Building in more complex attacker and defender models - for example, the probability of successful patching q could vary depending on which device is getting patched, there could be multiple patches deployed simultaneously that each have their own probabilities of success, more security controls could be added to more devices over time throughout simulation runs

instead of staying fixed the entire time, allowing multiple adversaries to attack the system at once (this scenario could model the phenomenon of discovering new zero day vulnerabilities), network topology could change as a function of time (perhaps certain devices could disconnect from the network entirely in response to a successful compromise of a neighboring device), we could allow for devices to try to infect their neighbors *before* we deploy patches. As mentioned earlier, the values for the compromise success rate p and the patching success rate q could also be set using real data.

Lastly, we also aim to improve upon REAP by improving the user experience: in the future, we hope to use tools like Prowler to automatically set security controls for devices in our network. We open-sourced REAP [1] so that application owners and cyber specialists can provide feedback on our work.

References

[1] [n. d.]. A Closer Look: Unveiling the Global Impact of CrowdStrike Event. https://www.guycarp.com/insights/2024/07/global-outage-with-widespread-impact.html. Accessed: 2024-12-15.

[2] [n. d.]. Risk Management Framework for Information Systems and Organizations. https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-37r2.pdf. Accessed: 2025-02-25.

[3] Ludovic Apvrille and Yves Roudier. 2013. SysML-Sec: A SysML Environment for the Design and Development of Secure Embedded Systems. In *APCOSEC 2013*. INCOSE.

[4] Natalia Dakhno, Olga Leshchenko, Yurii Kravchenko, Andriy Dudnik, Olexandr Trush, and Victor Khankishiev. 2021. Dynamic Model of the Spread of Viruses in a Computer Network Using Differential Equations. In *2021 IEEE 3rd International Conference on Advanced Trends in Information Theory (ATIT)*. IEEE, 111–115.

[5] Yuri Gil Dantas, Vivek Nigam, and Ulrich Schopp. 2024. A Model-Based Systems Engineering Plugin for Cloud Security Architecture Design. In *SN Computer Science*. Springer Nature.

[6] Shao-Ting Ge, Gong-You Tang, Xue Yang, Qi-Lei Xu, Hao Yu, and Pei-Dong Wang. 2013. Stability analysis of SEIQR model in computer networks. In *2013 25th Chinese Control and Decision Conference (CCDC)*. IEEE, 2244–2248.

[7] Johannes Geismann, Christopher Gerking, and Eric Bodden. 2018. Towards ensuring security by design in cyber-physical systems engineering processes. In *Proceedings of the 2018 international conference on software and system process*. 123–127.

[8] Myron Hecht and David Baum. 2019. Failure Propagation Modeling in FMEAs for Reliability, Safety, and Cybersecurity using SysML. In *17th Annual Conference on Systems Engineering Research*. Elsevier, 370–377.

[9] Martin Haug Larsen, Satyanarayana Kokkula, and Gerrit Muller. 2024. A Proposal for Model-Based Systems Engineering Method for Creating Secure Cyber-Physical Systems. In *INCOSE International Symposium*, Vol. 34. Wiley Online Library, 37–52.

[10] Azad M Madni and Michael Sievers. 2018. Model-based systems engineering: Motivation, current status, and research opportunities. *INCOSE* 21, 3 (2018), 1725–190.

[11] Swapnita Mohanty, Prasant Kumar Nayak, Arjun Kumar Paul, and Antaryami Basantia. 2023. SIQTRS e-Epidemic Model: A Comprehensive Framework for Analyzing and Managing Computer Virus Propagation in Networks. In *2023 OITS International Conference on Information Technology (OCIT)*. IEEE, 726–731.

[12] Mark Newman. 2018. *Networks*. Oxford university press.

[13] Joe R Reeder and Tommy Hall. 2021. Cybersecurity's pearl harbor moment. *The Cyber Defense Review* 6, 3 (2021), 15–40.

[14] Yves Roudier and Ludovic Apvrille. 2015. SysML-Sec: A model driven approach for designing safe and secure systems. In *2015 3rd International Conference on Model-Driven Engineering and Software Development (MODELSWARD)*. IEEE, 655–664.

[15] Michael Vai, David J Whelihan, Benjamin R Nahill, Daniil M Utin, Sean R O'Melia, and Roger I Khazan. 2016. Secure embedded systems. *Lincoln Laboratory Journal* 22, 1 (2016), 110–122.

Received 07 April 2025

[1] https://anonymous.4open.science/r/Reap-E277/README.md