SECURITY

# THREAT PATTERN LIFE CYCLE DEVELOPMENT

ISACA®

# CONTENTS

4 **Introduction**

5 **The Life Cycle of a Threat Pattern**
**5 /** Analysis
**5 /** Design
**6 /** Development
**6 /** Testing
**6 /** Evolution

7 **Define, Develop and Evolve Modeled Threat Patterns**
**7 /** Phase 1: Analysis
**8 /** Phase 2: Design
**9 /** Phase 3: Development
**10 /** Phase 4: Testing
**11 /** Phase 5: Evolution

13 **Conclusion**

14 **Acknowledgments**

# ABSTRACT

Threat patterns are a set of characteristics that represent suspicious behaviors in security platforms. This white paper describes a structured approach to the development of threat patterns that is based on the software development life cycle.

# Introduction

IT risk comes from various sources that are not always possible to identify in advance, making prevention and mitigation more difficult. With The explosive growth in cloud, social, mobile and bring your own device (BYOD) computing, the attack surface is greater than ever. Many attack scenarios are now possible due in part to the complexity of network topologies and the variety of enterprise applications and technologies that coexist and need to be defended.

Deploying "threat patterns" in security monitoring solutions, whether detective such as a SIEM platform or preventive such as a web gateway platform, is a great starting point for security operations teams to detect suspicious behaviors on networks, systems and applications. A threat pattern is a set of characteristics that represent suspicious behaviors to be detected; much like a "design pattern" in a software development context represents a template for a repeatable, stable, reproducible solution to a known and addressable problem, so too do "threat patterns" reflect a knowable set of facts that might be seen on multiple occasions that reflect a recurring method of attack or compromise. Threat pattern deployment can be driven by specific corporate policies, compliance mandates or out-of-the-box security solutions that provide general threat patterns.

Traditionally, threat patterns are subject to false negatives and false positives—i.e., situations in which a set of facts not reflective of an attack are falsely identified as an attack and, conversely, situations where an attack does exist but is not detected. This ambiguity can require significant effort from limited security operation center (SOC) resources, unless they are accustomed to providing greater context for the environment such as by understanding in detail a given application (for example) and using that understanding to quickly sort through false positives and negatives to provide accuracy to the data. This greater context is made possible with the use of a structured and effective software development life cycle approach.

Furthermore, attack vectors are not static, and advanced attacks are hard to detect. As a result, a one-off effort is usually insufficient and tends to make the pattern itself only useful for a limited time. Ongoing intelligence about attacks (in addition to current understanding of internal business activities and processes) is necessary to ensure that patterns are updated appropriately and that they evolve over time in keeping with developments in the threat environment.

The adoption of a simple and well-defined process that is borrowed from the software development life cycle (SDLC) helps to increase the threat detection rate and limits the effort of the SOC in dealing with false positives.

# The Life Cycle of a Threat Pattern

The SDLC is commonly used in software engineering to describe a set of practices and processes for planning, creating and testing a computer program, application functionality, or other software artifacts such as those developed by the organization (or suppliers) to help support business activities. By following the same approach in the design and implement of threat patterns, and carefully adapting it to a security platform, additional value can be brought to SOC stakeholders (such as analysts, managers, etc.) while minimizing the effort required to maintain these threat indicators.


**FIGURE 1:** Threat Pattern Development Life Cycle

Software development typically evolves through the following phases (see **figure 1**):

- Analysis
- Design
- Development
- Testing
- Evolution

Each phase has its own inputs, outputs and subprocesses, and the overall process is measured and monitored. While the nuances of specific practices will vary from one enterprise to another, the conceptual phases are likely to exist regardless of the specific model employed or the mechanics of SDLC implementation, etc.

## Analysis

Understanding the business and risk context of the enterprise helps to inform the definition of threat patterns and reduce the number of false positives and false negatives. The analysis phase includes review of the high- and low-level design of whatever asset the threat pattern is intended to monitor, along with an assessment of residual risk. An enterprise can do this formally, for example by employing a systematic method such as application threat modeling, or informally by systematically applying internal expertise and understanding to the threat evaluation. Some questions to answer are:

- What data flows, components and access points apply to the asset?
- What is the use (and misuse) case (e.g., the threat to be mitigated)?
- Which techniques and tools are associated with the use case?
- Which security controls are already deployed?

## Design

During the design phase, a conceptual description of the threat pattern should be developed to meet the requirement specifications and the use and misuse cases articulated during analysis. The result can be a formal artifact such as a data flow diagram (DFD) and may directly employ the results of threat modeling done in the prior step. It can leverage design artifacts (e.g., UML) or it can be done less formally.

Then, security platforms that will configure and deploy the threat pattern should be identified. Design objectives are met after all necessary design data are validated. Some useful questions to answer are:

- Which data feeds are required to develop the logic for the threat pattern?
- What is the logical flow of the pattern and its indicators?
- What are the building blocks and how are those linked together?
- What thresholds should be set?

## Development

A good implementation can result in a happy SOC team, instead of angry analysts who are overloaded with false positives. The rule to ensure a good result is not to assume that the design specification will work as expected after the concept is translated into proper machine language (code or a compiled image). Work should be divided into manageable modules and units before coding. Questions to answer include:

- Are the deployed threat feeds and sources delivering the required data?
- Which security platforms will deliver the most effective threat detection capability?
- Can the detection pattern be fully automated?
- Is the pattern indicator being deployed effectively in the security tools and platforms that were configured?

## Testing

Testing a deployed threat pattern is required to ensure that it meets functional requirements and addresses the original objectives. During testing, it is common to find that threat-detection logic and/or the data collected or received by the SOC are not as clean as they should be; both issues can negatively impact productivity and effectiveness if not corrected. Threat patterns must be tested extensively, like any new piece of software. Ideally, full code testing and remediation of all major bugs before moving the code into production will avoid unexpected results.

Some questions to answer during testing are:

- Is the quantity of test use cases adequate to validate the implementation?
- Is the pattern matching (i.e., triggering) when it should?
- Is the output expected?
- What is rate of the false positives and false negatives?

## Evolution

A threat pattern that works effectively today may become obsolete over time. Threat-actor tools, tactics and procedures (TTPs) and tradecraft evolve, as do business requirements and risk tolerance levels. Therefore, it is important to review each pattern periodically, define criteria for periodic evaluation (targeted to a broad spectrum of threats) and ensure that the life-cycle logic continues to align with business and risk objectives. It is generally useful to retain design artifacts during threat pattern development in the same way that they are preserved during software development; key artifacts include records of the pattern's original purpose, documentation of the specific threat/risk scenarios it was designed to detect and documentation of the specific exploitation vectors and targets it was

designed to mitigate. The following questions should be answered:

- Is a new threat pattern required or can the existing one be enhanced?
- Are design modifications properly documented?
- Are changes designed to address the evolving requirements adequately?
- How should measurements of residual risk be updated?

# Define, Develop and Evolve Modeled Threat Patterns

Completing the phases described in this paper will require investment of resources, but the effort is worthwhile. Adversaries and threat actors need only find one exposed weakness to access an enterprise's network. SOCs continually battle inadequate budgets, skill shortages and limited personnel. In this context, an investment in automating and streamlining security processes can have tremendous value. Adopting a structured and focused approach that leverages limited available resources is advantageous to protect the business and broader enterprise.

In the following sections, each phase of the threat pattern life cycle is considered in detail, including additional steps and examples for success.

## Phase 1: Analysis

To protect an asset, security practitioners must identify the threats that pose risk to the asset. For threat pattern identification, it is important to invest effort in the first phase—threat risk analysis—which blends threat modeling with asset risk analysis. Threat modeling allows practitioners to identify and prioritize threats and enumerate appropriate mitigating controls across network, host or application levels. Asset risk analysis involves identifying the assets at risk and the impact of potential loss from a given relevant risk. The objective of threat risk analysis is to understand the ecosystem's characteristics and peculiarities deeply and protect it accordingly.

Combining risk analysis and threat modeling techniques provides the greatest visibility and understanding of the targeted asset and helps to define the realistic business and risk context in which to operate. Adopting this approach helps the security team to evaluate:

- Which asset ecosystem components might be attacked?
- Which attack vectors might be used?
- Who might exploit a given vulnerability (whether an external actor or an internal user)?
- Which threats affect the confidentiality, integrity and/or availability of the asset?
- What are the most likely risk scenarios and associated degrees of damage that attacks can cause?
- Which security solutions can be applied to mitigate identified risk?

After analysis is complete, an understanding of the native security-control capabilities of the asset plays a key role. For example, detection of a brute-force attack use case on the authentication process of an asset may be achieved by implementing the correlation logic *"detect certain number of events X, followed by event Y."* If the approach does not leverage knowledge of the asset's native security control capabilities, then unneeded detection controls might be deployed. For example, if the asset included a CAPTCHA or a comparable security mechanism to mitigate the threat, then resources might

be applied to other use cases for which a security control might be less effective and allow for greater residual risk.

An important possible outcome of the threat risk analysis is that threats cannot always be fully mitigated by applying further security controls. Doing so may be ineffective, costly or not possible, and require a level of asset security monitoring of known vulnerabilities that is higher than necessary (relative to risk) and for which threat patterns have not been implemented. This residual risk, which is risk that remains after security countermeasures have been applied, can be monitored by developing and shaping threat patterns in relation to implemented security controls, data flows and potential attack vectors. Therefore, a threat pattern should always be developed to prioritize the detection of events based on residual risk and the monitoring of the attack scenarios, which can circumvent any existing countermeasures.

In taking this approach, a security team will focus on threats that can exploit identified vulnerabilities, rather than more obvious—but less relevant or harmful—threats. In enterprises with limited security personnel and budgets, this methodology helps to save time and money while also reducing the implementation of unnecessary security controls. The identification and remediation of residual risk results in more efficient resource allocation. Monitoring and identifying the important threats and avoiding elaborate countermeasures that do not enhance the enterprise's security posture are the desired outcomes.

## Phase 2: Design

Sometimes, SOC content engineers want to move immediately from abstract ideas into coding and development, believing that they will save time. This may partially explain why many current detection strategies are ineffective against motivated attackers. Improper design and evaluation can lead to an ineffective threat pattern.

The outcome of proper analysis, design and development provides great insights on the challenges that an enterprise is trying to solve and the residual risk that should be considered. In the design phase, the problems identified in the use and abuse cases begin to be addressed. Good coverage in this phase is a key facilitator to proper risk analysis alignment.

In the design phase, the enterprise should consider a threat pattern in the same way that it considers a software design pattern. The objective is to develop a general and reusable solution to a commonly occurring problem—not to choose the design pattern based on immediately available data or technology. Design should initially consider what data will make an application most relevant, rather than limit the application to given inputs and the capabilities they necessarily entail. Threat patterns are no different.

After considering the data components, the various pieces of the design model need to be linked together. When designing the logical flow and building blocks of the threat pattern, it is critical to not make quick assumptions. For example, assuming there is only one platform for implementing a given pattern dramatically limits the detection efficacy. A very common mistake at this stage is to design threat patterns with only security information and event management (SIEM) technology in mind. Although SIEMs are valuable for gaining an aggregate view and understanding comprehensively what security controls are detecting, they should not be considered in isolation, because the resulting pattern will have inherent limitations.

For detection patterns, the development of logic that identifies a specific threat should be considered in the context of the alerting strategy so that the security team is informed of the relevant threats. Not all detected patterns are bad. It is important to factor the quantification of a given scenario into the design of the threat pattern to ensure that the pattern is flexible enough to adapt to an ever-changing threat landscape. In this phase, the

value of the pattern can be optimized by defining appropriate thresholds for detection, i.e., limits above which an event becomes significant or turns into an incident.

It is important for teams to spend adequate time in the design phase in order to create the most relevant threat patterns. Allocating the necessary time and introducing checkpoints throughout the work can help teams to:

- Identify issues that may not otherwise become visible until implementation
- Provide a reusable paradigm effective for other indicators
- Ensure a platform-independent architecture that is more effective against a specific attack vector

## Phase 3: Development

Concepts articulated in the analysis and design phases come to life when specific controls are deployed in the security ecosystem. However, long gone are the days when a single security solution addresses most of the security challenges that enterprises face. Today, the security ecosystem reflects a multitude of distributed technologies collectively comprising the foundation of the enterprise security strategy.

Security-monitoring capabilities exist in many different platforms and, whether their final objective is to detect or prevent threats associated with specific risk, careful consideration must be given to the choice of target security platforms when new threat-detection controls are being developed and deployed.

In layered detection, one of the primary principles of threat detection, a main element of the threat pattern is configured into the asset (or into the monitoring platform closest to the asset); additional capabilities are implemented into other adjacent threat-detection technologies, integrating and extending detection. This layered configuration enables monitoring for attacks at different levels, thus reducing the dwell time of a threat actor.

An example threat pattern application scenario involves a web server that exposes an authentication system to users. The different elements of the threat pattern are applied to the service architecture tiers. This is followed by the deployment of additional controls in the security platforms that are responsible for monitoring the data being processed, such as web application firewall, database security monitoring and network analytics platforms. In this case, a misuse by a threat actor creates artifacts and traces in these technologies in the form of HTTP server-side error messages, SQL statements by the backend, unexpected business logic behaviors, requests to admin interfaces, call of system functionalities for privilege escalation, upload of file, outbound peaks, etc. These can be carefully monitored to develop the logic of the threat pattern.

Another key aspect to consider in development is the data–analysis techniques that are leveraged in threat pattern implementation. Every threat pattern has unique objectives; a single technique to fit all patterns does not exist. Data collection and analysis phases are heavily based on the inherent data characteristics, the security platform being used and the scenario to be detected, investigated and reported. The data-analysis technique also depends on the type of data collected. For example, in some scenarios, the data must be comprehensive enough to cover not only the logs generated by all the units, but also the network traffic, to augment the visibility and enhance the detection. All this data needs to be married with internal contextual information, such as the residual risk and external threat intelligence.

Data-analysis techniques need to be implemented with in-depth analysis approaches in mind to detect the wide range of threats being monitored for anomalies, unknown threats and suspicious behaviors. Eventually,

the choice of the analysis technique is mainly driven by the data sources, with the inherent value of the collected data only as relevant as the technique applied.

For the development phase as a whole, security engineers should consider the following principles and best practices to maximize their effectiveness and keep up with the rapidly changing landscape:

- Keep implementation of threat patterns simple
- Do not repeat in code writing in multitier architectures
- Heed results of analysis and design phases to avoid the introduction of unneeded functions
- Make threat patterns easy to maintain and extend

The continuous evolution of business and pressure on the IT ecosystem combine to make an ever-changing threat landscape even more daunting. Threat actors need to be right only once, while defenders must be right every time. Even though attacks share some common elements, no single or consistent attack methodology is used by all threat actors. They are elusive and adjust their techniques and workflows to circumnavigate defenses and compromise their targets.

The principles and practices outlined thus far can empower the enterprise to implement effective threat patterns that remain current and better aligned with business requirements.

## Phase 4: Testing

After analyzing requirements, asset and threats, designing a general and reusable model for the threat pattern and implementing the model in the security-monitoring platform, one cannot assume that no mistakes were made. Therefore, a well-structured and accurate testing process is vital to validate pattern implementation and confirm that the enterprise has addressed the attack scenario effectively.

In this phase, as in the others, the typical SDLC process is applicable. After software is developed, it is usually tested with two distinct, but complementary, methodologies—a white-box and a black-box approach.

- **White-box testing** tests the software code from the inside out. It verifies the threat pattern logic and ensures that design and development align with requirements. If the design phase has been approached correctly, the different components making up the pattern should already be well defined and documented: For each component, a list of inputs and expected outputs that trigger each part of a release should be prepared. Testing progress can be quantified by the percentage of code that the test covers.

  White-box testing may be time consuming—and time is a precious resource in a SOC—so automating even part of it is key. At this stage, a good amount of historical data should be available, because information has already begun flowing into the system. Representative sample selection is critical: It must ensure good horizontal coverage (e.g., the number of tested components) and vertical coverage (e.g., the variety of the inputs). The mapping of each selected sample with the component under testing makes logic validation and threat pattern effectiveness more thorough. Furthermore, the same samples can be re-injected into the platform after any change to ensure the output is still consistent with requirements.

- **Black-box testing** takes an outside-in approach, examining the functionality of the threat pattern without dissecting its logic. During black-box testing, the best practice is to start from the attack scenarios originally defined during the analysis phase and determine the possible means an attacker can leverage to exploit the depicted situations. Many enterprises find it beneficial to establish (or otherwise engage) a red team in the role of the attacker, while a blue team defends the enterprise by ensuring the

implemented threat patterns produce the outputs expected. After the exercise, the two teams meet to review results and list all the situations not detected by the current implementation and the threat patterns requiring further review.

This leaves the challenge of how to measure the effectiveness of the patterns based on test results. The simplest way is to evaluate the number of false positives and false negatives. Those two parameters alone may be sufficient to gauge the benefit of an implemented threat pattern versus the potential noise.

It is important for every SOC to understand that the most dangerous enemy may not be an obscure attacker; rather, it may be the number of false positives overwhelming analysts and preventing them from responding to security issues that really matter. It is imperative to reduce the noise-to-signal ratio before it becomes insurmountable on the journey to protect the enterprise.

At the end of the process, a root-cause analysis must be conducted on every identified issue; however, simply changing the implementation is not enough. Valuable lessons can be learned at the end of the life cycle. The requirements and the results of the analysis should be re-examined before touching the code. The following questions should be asked:

- Was the attack scenario missed entirely?
- Is there any data source that was not considered?
- Was it an implementation error?

Approaching this phase with a simple checklist also helps to ensure that a potential change here does not adversely affect other threat patterns.

Finally, it is important to be aware that the work performed thus far may become obsolete in a very short time. New TTPs and attack scenarios may arise, and the enterprise's business strategy may have changed. Evolving the threat pattern life cycle helps to mitigate both situations.

## Phase 5: Evolution

In the era of agile development and digital transformation, any application is subject to ongoing enhancement and improvement. Indeed, software engineering is a complex process with many interdependent tasks where multiple functions share responsibilities to strike a balance between software quality and business objectives, regardless of the specialized nature of the teams within the organizational structure.

After a threat pattern is developed and deployed into a production environment, it must be subject to ongoing review. This life-cycle phase is evolution. Threat patterns need to be monitored to ensure that they remain relevant and continue to deliver value relative to analyst workload. This monitoring involves continuous evaluation and enhancement to keep up with newly discovered attack vectors and changes in the IT ecosystem.

The maintenance of a threat pattern includes:

- Changes in the implementation technique (e.g., the way the model is engineered)
- Changes in the event source or external context data
- Changes in threshold values
- Changes regarding the output of the testing phase

Useful habits can be borrowed from software engineering. Best practices in software maintenance are codified in ISO/IEC 14764, "Software Engineering – Software Life Cycle Processes – Maintenance."[1] This standard identifies four main types of maintenance: corrective, adaptive, perfective and preventive.

1 International Organization for Standardization, "Software Engineering – Software Life Cycle Processes – Maintenance," ISO/IEC 14764, September 2006, *www.iso.org/standard/39064.html*

- **Corrective maintenance** addresses bug repair or failures that should never have occurred, including design, coding or logic errors. The testing phase should, ideally, have prevented most of these issues. However, some errors surface only when patterns are deployed in a production environment with live data. Corrective maintenance may also address threat patterns that exceed defined thresholds and create unintended side-effects, including:
  - Degradation of system performance
  - Excessively high number of alerts
  - Unacceptable levels of false positives/ false negatives
- **Adaptive maintenance** reflects the need for threat patterns to adapt to changes in the IT environment. Even a minimal change can influence the efficiency and efficacy of a threat pattern. Close collaboration with the IT department, in this specific case, is key. For example, the onboarding of a new application/ platform provides the security team with a new event source, which can then participate in the overall threat-detection program.
- **Perfective maintenance** accommodates small functional enhancements, tweaks and minor improvements in the model. Three primary elements should be considered:
  - Operational threat intelligence—Many SOCs have the capability to monitor for targeted attacks proactively. Threat patterns can be enhanced by supporting the ingestion of internal and external context produced by an attacker's activity.
  - Lessons learned during the incident response life cycle—The overall objective is to leverage the knowledge gained across incident responses as input to enhance the current logic of specific threat patterns.
  - Threat hunting activities—Proactive threat hunting enables the security team to understand the IT environment better and, consequently, improve and tune existing threat patterns with more insights about the enterprise network and application infrastructure.
- **Preventive (or proactive) maintenance** reflects periodic efforts to reduce the complexity of threat patterns and maintain relevance by keeping them current.

In this last phase of the life cycle, no single set of actions or inputs can identify the elements to be modified. Ideally, all enhancement requests should be managed in a centralized repository with change-tracking capability to document the benefits and impacts of any proposed changes to the threat patterns.

During the review of a specific threat pattern, the security operation team might consider it no longer useful; for example, it may become clear that a different technology can accomplish the same objective more quickly. In this case, a formalized end-of-life (EOL) process must be activated by assessing the impact and involving all relevant stakeholders.

# Conclusion

A structured approach to threat pattern life cycle development can help the SOC to detect adversaries more rapidly and with greater efficacy, by minimizing the effort required to maintain detection techniques and lower the rate of false positives.

The variety of risk that the SOC encounters and the evolving threat landscape are challenges inherent to the nature of the business. Limited resources require SOC teams to focus on threats that can cause real damage to the business, rather than chase false positives or continue tuning ineffective security countermeasures. A structured approach enables the team to develop meaningful threat patterns and assign the right priorities to specific use (and abuse) cases.

By considering threat pattern development from the perspective of software development and applying best practices and principles borrowed from the SDLC, enterprises can develop threat management capabilities to better protect their assets, address IT security risk, mitigate existing threats and maximize the limited resources available in the SOC.

# Acknowledgments

ISACA would like to recognize:

## Lead Developers

**Demetrio Milea**
CISA, CISM, CISSP,
RSA Security, Italy

**Davide Veneziano**
CISA, CISM, CISSP,
Deloitte Risk Advisory, Italy

## Expert Reviewers

**Joanne De Vito De Palma**
CISM,
The Ardent Group, USA

**Fahima Ahmed Khan**
CISA,
IBM Canada, Canada

**Sean Ennis**
CISSP,
USA

## ISACA Board of Directors

**Theresa Grafenstine**
CISA, CRISC, CGEIT, CGAP, CGMA,
CIA, CISSP, CPA,
Deloitte-Arlington, VA, USA, Chair

**Robert Clyde**
CISM,
Clyde Consulting LLC, USA, Vice-Chair

**Brennan Baybeck**
CISA, CRISC, CISM, CISSP,
Oracle Corporation, USA, Director

**Zubin Chagpar**
CISA, CISM, PMP,
Amazon Web Services, UK, Director

**Peter Christiaans**
CISA, CRISC, CISM, PMP,
Deloitte Consulting LLP, USA, Director

**Hironori Goto**
CISA, CRISC, CISM, CGEIT, ABCP,
Five-I, LLC, Japan, Director

**Mike Hughes**
CISA, CRISC, CGEIT,
Haines Watts, UK, Director

**Leonard Ong**
CISA, CRISC, CISM, CGEIT, CPP, CFE,
PMP, CIPM, CIPT, CISSP ISSMP-ISSAP,
CSSLP, CITBCM, GCIA, GCIH,
GSNA, GCFA,
Merck & Co., Inc., Singapore, Director

**R.V. Raghu**
CISA, CRISC,
Versatilist Consulting India Pvt. Ltd.,
India, Director

**Jo Stewart-Rattray**
CISA, CRISC, CISM, CGEIT, FACS CP,
BRM Holdich, Australia, Director

**Ted Wolff**
CISA,
Vanguard, Inc., USA, Director

**Tichaona Zororo**
CISA, CRISC, CISM, CGEIT, COBIT 5
Certified Assessor, CIA, CRMA,
EGIT | Enterprise Governance of IT
(Pty) Ltd, South Africa, Director

**Christos K. Dimitriadis, Ph.D.**
CISA, CRISC, CISM,
Intralot, S.A., Greece, Past Chair

**Robert E Stroud**
CRISC, CGEIT,
Forrester Research, Inc., USA, Past Chair

**Tony Hayes**
CGEIT, AFCHSE, CHE, FACS, FCPA, FIIA,
Queensland Government, Australia,
Past Chair

**Matt Loeb**
CGEIT, FASAE, CAE, ISACA,
USA, Director

## About ISACA

Nearing its 50th year, ISACA® (*isaca.org*) is a global association helping individuals and enterprises achieve the positive potential of technology. Technology powers today's world and ISACA equips professionals with the knowledge, credentials, education and community to advance their careers and transform their organizations. ISACA leverages the expertise of its half-million engaged professionals in information and cyber security, governance, assurance, risk and innovation, as well as its enterprise performance subsidiary, CMMI® Institute, to help advance innovation through technology. ISACA has a presence in more than 188 countries, including more than 215 chapters and offices in both the United States and China.

### DISCLAIMER

ISACA has designed and created *Threat Pattern Life Cycle Development* (the "Work") primarily as an educational resource for professionals. ISACA makes no claim that use of any of the Work will assure a successful outcome. The Work should not be considered inclusive of all proper information, procedures and tests or exclusive of other information, procedures and tests that are reasonably directed to obtaining the same results. In determining the propriety of any specific information, procedure or test, professionals should apply their own professional judgment to the specific circumstances presented by the particular systems or information technology environment.

### RESERVATION OF RIGHTS



ISACA®

3701 Algonquin Road, Suite 1010
Rolling Meadows, IL 60008 USA

**Phone:** +1.847.660.5505

**Fax:** +1.847.253.1755

**Support:** support.isaca.org

**Web:** www.isaca.org

**Provide feedback:**
www.isaca.org/Threat-Pattern-Life-Cycle-Development

**Participate in the ISACA Knowledge Center:**
www.isaca.org/knowledge-center

**Twitter:**
www.twitter.com/ISACANews

**LinkedIn:**
www.linkd.in/ISACAOfficial

**Facebook:**
www.facebook.com/ISACAHQ

**Instagram:**
www.instagram.com/isacanews/