Supporting Cyber Threat Analysis With Service-Oriented Enterprise Modeling

Anonymous Authors

Keywords: Conceptual Modeling, Threat Modeling, Service-Oriented Architecture, Service-Oriented Computing, Conceptbase, Threat Analysis, Indicators of Compromise, IOC

Abstract: Today's enterprise environment is rapidly changing with organizations adopting cloud services at record rates. This deperimeterization of enterprise computing architectures depends on software as a service (SaaS) and makes traditional perimeter-based defense controls less effective. We propose a service-oriented threat modeling approach that focuses on the perspective of a service consumer. We supplement our approach by providing an implementation view that includes technical details of service implementations that can be queried to identify potential vulnerabilities in the system. Our approach differs from existing threat modeling methods in that we seek to capture interactions between services in a technologically agnostic manner. This extends the applicability of our model into the realm of security operations. A case study and proof-of-concept are presented to validate our approach and demonstrate how such a model can be used to provide meaningful support for operations engineers.

1 INTRODUCTION

Threat modeling is commonly used to denote a process that systematically enumerates attack vectors and potential weaknesses at design time. As such, it is a tool widely used by cybersecurity builders. However, the term threat modeling, as used in cyber defense, is somewhat of a misnomer since it is generally not used to model actual threats. Rather, cyber threat modeling methods are used to describe systems, and to support analysis of those systems, to identify and analyze threats.

Many different approaches to threat modeling have emerged in literature in the recent past. Most of those position threat modeling as an activity that should be performed as part of requirements engineering and systems design efforts. For example, (Dhillon, 2011) sees threat modeling as a conceptual exercise to analyze a system's architecture or design to find security flaws and reduce architectural risk. He acknowledges that threat modeling can occur at any time during a system's life cycle, but recommends it to be part of the architecture or design phase. More recently, (Sion et al., 2019) defined threat modeling as entailing the systematic enumeration of misuse and attack vectors and considering their applicability in the system under design.

As early as 2008, (Malik et al., 2008) acknowledged that threat modeling should be an ongoing process and proposed a 7-step threat modeling approach that loops continuously. However, the approach is defined at a high-level, and does not propose any specific methods by which each step should be accomplished. In this paper, we agree with with Dhillon and Malik that threat modeling should be considered as an ongoing process that continues beyond requirements engineering and design into the deployment and operation of a full enterprise computing landscape.

Today's environment is rapidly changing. Organization are adopting cloud services at a rate not seen before, as 92% of organizations today have IT environments that are at least partly in the cloud (IDG, 2020). Increased use of cloud services results in decreased visibility into the operations of those systems. Consequently, traditional (perimeter-based) cybersecurity practices become less effective.

Employees are also increasingly removed from the enterprise network as they complete their daily duties from home. This trend started well before the outbreak of the global COVID-19 pandemic, but the move to telecommuting was significantly amplified and accelerated by lock-down mandates and social distancing concerns.

The third observable trend is the increased adoption of transport-level encryption (TLS) to cryptographically protect network communications. As of the time of writing this article, adoption of the HTTPS-protocol to access websites exceeds 95% (Google, 2020).

The problem caused by these three coalescing

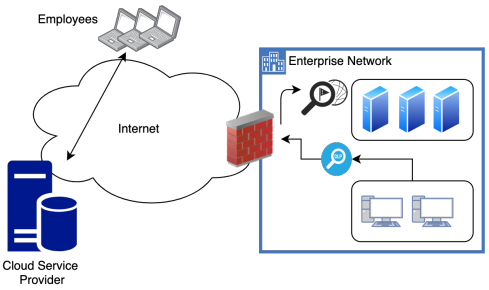

Figure 1: Deperimeterization

trends is illustrated in Figure 1, which shows that most enterprise computing security controls, like firewalls, intrusion detection/prevention systems, and/or data loss prevent systems, are commonly placed at the perimeter of the enterprise network. However, with neither the service provider, nor the service consumer present on that network, the effectiveness of perimeter-based controls has decreased since data no longer flows through them.

The ongoing process of deperimeterization, in which organizations continue to rapidly adopt cloud services, and in which employees are increasingly mobile, makes defending the enterprise environment a much harder process. While some organizations resort to the use of virtual private networks (VPNs) to route network traffic between service providers and their employees through the enterprise network before sending it back out to the Internet, we see this as a solution that does not scale and will not be sustainable in the long term. In addition, requiring employees to establish VPN connections in order to access Internet-based resources introduces additional attack vectors into the protected enterprise network, thus potentially decreasing the overall security posture of the organization. The increased use of end-to-end encryption will continue to erode the effectiveness of perimeter-based controls.

This realization leads us to formulate our research problem as: deperimeterization of enterprise computing architectures will continue, making traditional perimeter-based controls increasingly less effective.

Our long-term research objective is to evolve cyber threat modeling to provide meaningful support to enterprise defenders operating in a deperimeterized computing environment for identifying and analyzing cyber threats. We first need to determine the goal of threat modeling, as well as the additional complexities introduced by operating in a deperimeterized environment. Next, we divide the modeling problem into three separate sub-problems: modeling the enterprise computing environment, modeling threats, and mapping threats to the environment. Each of these steps will be validated through a manual exercise, based on a case study. Lastly, we explore the possibility to automate each of these steps.

This paper will focus on addressing the first of the above objectives: how can we threat model deperimeterized environments to adequately reflect a service-oriented approach to software utilization? We posit that traditional threat modeling approaches, mostly undertaken during requirements engineering and for systems design, can be augmented to provide meaningful support for operations engineers. We propose a model with main components that benefits the service consumer as well as the service provider.

To demonstrate our modeling process, we present a case study and a proof of concept of our modeling proposal using ConceptBase implementing the O-Telos language. We compare our modeling approach vis-à-vis traditional modeling techniques to see how it better describes a deperimeterized environment and how we can query our model to identify potential threats to the described environment. We conclude with directions for future work, including potential approaches to automate this process.

2 RELATED WORK

2.1 Threat Modeling

Modeling is an activity in which an analyst creates an abstract representation of the relevant aspects of a system or process. Modeling approaches are generally specific to their purpose. For example, a data model provides the analyst a vocabulary and a grammar to express data stored in a system and/or moving throughout a system.

Threat modeling is a structured method used to identify possible threats to systems (Hussain et al., 2014). Threat modeling is not new. In an early paper, (Shostack, 2007) describes the experiences of his team at Microsoft with threat modeling going back as early as 1999. The modeling approach described in the paper consists of four main activities: diagramming using data flow diagrams, per-element threat enumeration using the STRIDE methodology, mitigation, and validation. Other well-known threat modeling methods include PASTA (UcedaVelez and Morana, 2015), Attack Trees (Schneier, 1999), VAST (Shevchenko et al., 2018) and OCTAVE (Alberts et al., 2003). All threat modeling approach have elements in common: they define a narrow, well-defined vocabulary to describe conceptual elements, and the relationships that exist between them. Some also propose a modeling process.

To be effective, cyber threat management pro-

cesses require that defenders have an established process in place for mapping knowledge of known cyber threats to the current state of their infrastructure. In the context of product development, threat modeling should be done early in the process. When developing threat models to analyze enterprise networks, modeling must be an ongoing process. As enterprise IT landscapes are highly dynamic in nature, and new services are adopted and brought to end-of-life continuously, models of enterprise architectures must continuously be updated to ensure that they are correct and complete representations of the actual environment.

Approaches to threat modeling can be organized into three categories: asset-centric, attacker-centric, or software-centric (Shostack, 2007). In a later book, Shostack argues for the software-centric approach as the best option as it requires understanding the complexity of the models, which results in substantial improvement in the security of the software components (Shostack, 2014). Focusing on assets requires mapping those assets to software to be useful in identifying threats. Focusing on attackers often does not create reproducible results. Focusing on the software being built and the systems being deployed best incorporates the understanding of threats into the overall operation of the system.

Threat modeling involves understanding the complexity of the system and identifying all possible threats to the system, regardless of whether or not they can be exploited (Myagmar et al., 2005). The authors discuss threat modeling in the context of requirements analysis, which takes place prior to software design and construction.

The focus on applying threat modeling techniques to software development is prevalent throughout the literature. For example, (Torr, 2005) states that "threat modeling an entire product is typically too complex, whereas doing so for individual features is too simplistic."

(Steven, 2010) takes a broader approach to threat modeling. He defines it as "the process of enumerating and risk-rating malicious agents, their attacks, and those attack's possible impacts on a system's assets." However, Steven also argues that the adoption of threat modeling as a common practice is lagging as the result of many security managers considering it as expensive and difficult.

In a cloud-based environment, the natural extension is to shift to focus on services in threat modeling. Focusing on services incorporates the comprehensive model of the system as in software-centric approaches, while considering the assets that are involved in the provided services. It can represent the system from the perspective of the service consumer.

2.2 Service-Oriented Computing

Service-Oriented Computing (SOC) was established as a field of research when it was described by (Papazoglou and Georgakopoulos, 2003) as the computing paradigm that utilizes services as fundamental elements for developing applications with a number of focus areas for further exploration. In 2017, these authors revisited their original article and paired up with others to publish (Bouguettaya et al., 2017), in which service-oriented computing is positioned as having emerged as a cross-disciplinary research field that studies the science and technology underlying the popularity of the IT service industry.

(Papazoglou, 2003) describes the Service Oriented Architecture as a logical way of designing a software system to provide services to either end-user applications or other services distributed in a network through published and discoverable interfaces and on the premise that services are loosely coupled. (Leune, 2007) further identifies services as an aggregate of atomic computational operations and/or of other services.

The service-oriented paradigm promotes describing the services provided by a technical infrastructure abstractly, and separates the description of the service's capabilities and interfaces from their implementation.

This approach closely resembles how modern enterprises are deperimeterizing their IT infrastructures, and, consequently, provides a natural reference framework. The service-oriented paradigm enables an analyst to reason about both the capabilities provided by a service, as well as about the technical properties of the system used to provide the service.

3 SERVICE-ORIENTED MODELING FOR CYBER THREAT ANALYSIS

As mentioned earlier, threat modeling is predominantly embraced during the requirements analysis and design phases of a software engineering project. We propose extending the practice by expanding threat modeling to include separate stages for *enterprise service modeling* and for *cyber threat analysis*.

Furthermore, threat modeling is commonly performed by service providers, but its adoption by services consumers is lagging. Our approach aims to extend the practice of threat modeling to benefit service consumers as well.

In this paper, we adopt a service-oriented perspec-

tive on enterprise modeling, rather than a network-centric approach. We model services, data flows, data storage, and authentication systems conceptually (i.e., technology-agnostic), and then proceed to enrich the model using technical implementation details. The technical details provide an operational perspective, which is mapped to the conceptual overview that provides the design view.

Our approach provides a mechanism for enterprise service modeling in support of cyber threat analysis conducted by service consumers. The deficiencies of traditional modeling techniques in effectively modeling services from the perspective of a service consumer can be attributed, at least in part, to the shift towards cloud-based services. Since modelers typically have fewer insights into the inner workings of the technologies used to provide software-as-a-service, modeling approaches that depend on the availability of such information fall short.

Most of the existing threat modeling techniques lack adequate expressiveness and semantics to enable reasoning about threats. Therefore, the lack of adequate semantics makes the development of automated threat derivation tools and threat model validation difficult (Mirembe and Muyeba, 2008). The most widely used threat modeling approach, STRIDE, is based on an enhanced version of the traditional data flow diagram (DFD) (Hussain et al., 2014). The four primitive terms used in a DFD are the terminal (an entity external to the model), the process, the data flow representing data-in-motion, and the data store representing data-at-rest. The models used in STRIDE extend these four primitives with a fifth: a trust boundary (Shostack, 2007). But in a service-oriented architecture, many of these concepts no longer apply or are inaccessible to the analyst. The priority should be on the capabilities of the services offered and the technical properties of how they are provided (Leune and Kim, 2020).

Before we can establish a modeling approach, we must identify the goals that we set out to achieve. We believe that using threat modeling during requirements engineering and for systems design can be augmented by evolving it to provide meaningful support to enterprise defenders operating in a deperimeterized computing environment.

The complexity of the discipline of secure systems engineering requires automated tool support where the precise definition and specification of a threat becomes a prerequisite (Rouland et al., 2020). Specifically, we intend to develop a method for mapping cyber threat intelligence to enterprise service models in a meaningful, and eventually, automated way. By doing so, enterprise defenders will expand and improve their situational awareness, and be able to better recognize and stop attacks.

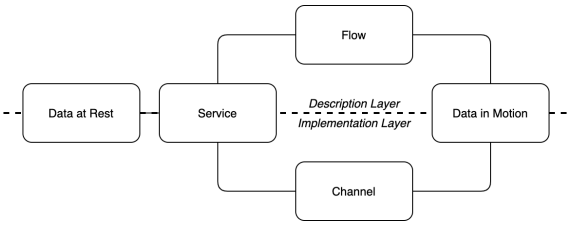

Figure 2: Model Overview

Threat models based on services have been explored by (Kazim and Evans, 2016). They describe (cloud) services as having four generic properties: data storage, data processing, data transfer, and authentication. Their approach uses these four properties as the basis for analyzing a select number threat that apply to cloud services. For each threat, they identify a threat actor and a possible method of attack. While this approach can be useful, it falls short of comprehensively analyzing the full computing environment.

Most modeling techniques incorporate knowledge of potential threats, rather than knowledge of actual threats. In part, that is caused by a relatively broad approach to modeling. For example, consider a composite service consisting of an web service that is accessed from a browser via HTTP/1.1 over a TLS 1.3 channel, and of an administrative service that is accessed via the secure shell protocol. Under traditional modeling techniques, such a service would simply be modeled as a collection of processes and data flows, without the semantic content that would be available otherwise (Shostack, 2007). In particular, traditional modeling approaches use data flows merely to show the existence of and the direction of the flow of data between structural components (Shostack, 2014). Our approach seeks to address this deficiency by representing *how* data flows between services.

We first look at defining the service as the fundamental concept on which this approach is based. We then discuss how services can be combined and extended through orchestration practices. Lastly, we explore service interaction, as a basis for security analysis.

3.1 Model semantics

Prior to defining the syntactic elements of our threat modeling approach, we introduce its main components and explain their semantics.

As shown in Figure 2 The central concept in our approach is the Service. Services contain data at rest, or they receive data flows via flow channels that rep-

resent data in motion. The model is described at two levels of abstractions: a conceptual design view, which describes technology-agnostic properties of the model elements, and a specific implementation view, which provides details about their implementation.

3.1.1 Services

Definition 1 (Service)**.** *A* Service *provides a unit of work.*

From a service consumer's view, a service is often considered as a self-contained black box that logically provides business functions. This conceptual view allows the consumer to reason about the functions provided by a service, without having to also consider its technical implementation details and service-delivery mechanism. This view, which we refer to as the *design view* is most useful when analyzing business processes, assessing overall system designs, and determining data flows into and out of the service.

However, service models used for security analysis cannot ignore implementation details. Specific knowledge concerning application software, operating systems, networking configurations, etc., are needed to conduct in-depth security analysis. Most indicators of compromise that are obtained from sources of cyber threat intelligence will provide such details.

We acknowledge this by also defining a supplemental *implementation view*, which describes the technical details of service implementations. Services can directly provide functionality, but they can also rely on other services to provide their functions. This is known as *service aggregation* or *service composition*.

The design service view extends the work of Kazim and Evans referenced earlier by acknowledging that internal service properties include details about the nature of data stored and processed by the service, and the mechanism by which it is stored, as well as details pertaining to data transfer and authentication.

Distinguishing the design view from the implementation view leads to several items of interest. First, the comparison of actual behavior and intended behavior allow for the automated detection of deviations between the two. Any service that is used in a way that was not expected should be considered for further investigation. Second, the operational behavior of a system can be observed through analysis of event logs, such as authentication logs, access control logs, etc.

3.1.2 Data Flows and Flow Channels

Definition 2 (Data Flow)**.** Data Flows *represent data-driven interactions between services and other services and/or actors playing roles of* Service Providers *or* Service Consumers*. Data flows are described in the* design view.

Services do not exist in isolation and are expected to interact with other services and/or with individual actors in specific roles. Data flows have long been used to represent such data-driven interactions, and we adopt the concept to represent interactions between services as well.

Services are provided by a service provider and consumed by a service consumer. Both are represented in our model as *roles* played by specific *actors*. Knowing what entities interact with a service, and in what capacity they do so, supports investigations and may provide the ability to attribute specific threats to threat actors. For example, the Diamond Model for Intrusion Analysis (Caltagirone et al., 2013) explicitly identifies actors in the roles of victim and/or adversary. By including information regarding these actors, mappings to threat intelligence in a later stage will be facilitated.

Data flows are defined at a conceptual level and, if known, have a source and a destination.

Definition 3 (Flow Channel)**.** *A* Flow Channel *describes implementation-specific properties of data flows. Flow channels are described in the* implementation view.

The channel through which data flows is represented as a *flow channel* and a mapping between the data flow and its flow channel is maintained. Flow channels are described by properties, which are key-value pairs that capture protocols used, protections added, and any other properties that may be deemed relevant for analysis. For example, common flow channels properties include `proto=https-over-tcp`, `encryption=tls1.3`, `port=443`, etc. Flow channel properties can be made as specific or fine-grained as needed.

4 VALIDATION

We validate our approach by manually modeling a case study, and then implement the model. Since we begin with manual validation, the case study is intended to be of relatively low complexity. In future work, we will automate sections of the modeling mechanism and expand testing with additional larger case studies.

4.1 Case Study

The case study revolves around a hypothetical organization providing consultancy services to its clients. To do so, it relies on several cloud-hosted services. The organization's employees all telecommute from various locations around the United States. Employees communicate among themselves via a messaging app, via email, and via video chat. Client contact is generally done via email and video chat only. All three of these services are adopted as software-as-a-service (SaaS) and are accessible as web applications and as mobile apps.

The organization manages its billing and time planning through a unified time management platform, which allows a project manager to define tasks, and for consultants to log hours to those tasks. The system is also used to generate billing information.

There is a small corporate office, which is staffed by the CEO and her executive assistant, as well as a small sales staff. The staffers in the corporate offices store their work on a self-hosted file storage device. Remote employees can access the storage by connecting to a virtual private network server, which is also hosted on-premise.

4.2 Proof of Concept

We have implemented a proof-of-concept using ConceptBase, a multi-user deductive database system implementing the O-Telos language. It is a powerful tool for metamodeling and engineering of customized modeling languages (Jarke et al., 1995). As such, it is an ideal tool for the task at hand. The ConceptBase version used for this proof-of-concept implementation is designed as a client/server application.

The choice to use the Telos object model was primarily motivated by our desire to unambiguously define core modeling concepts, to be able to reason about them, and to be able to query the model. We found this ability in the O-Telos language, which provides facilities for constructing, querying, and updating structured knowledge bases (Mylopoulos et al., 1990).

Further benefits of using the O-Telos implementation provided by ConceptBase include the ability to query the model using a convenient logic-based syntax, a clear frame-based representation format, and the availability of a graphical browser that lets a user easily manipulate constructs. ConceptBase also enforces integrity constraints, ensuring that models are logically consistent at all times.

An O-Telos database has two primary representation mechanisms: a logic representation based on proposition logic, and a frame representation that more closely resembles a typical programming language syntax. All expressions included in this paper are written in frame syntax to ensure greater readability.

```
Service in Class with
attribute
  description: ServiceDescription;
  implementation: ServiceImplementation;
  association: Role;
  extends: Service
end

ServiceDescription with
attribute
  dataflow: DataFlow;
  data: DataElement;
  authentication: Authentication
end

ServiceImplementation with
attribute
  channel: FlowChannel;
  storage: DataStorage;
  authenticationMechanism:
      AuthenticationMechanism
end
```

Listing 1: Model Definition Fragments

4.2.1 Base Model

We begin building out the base model using ConceptBase's O-Telos frame syntax, and define a service as shown in Listing 1.

A `Service` is an instance of a generic `Class` and has four typed attributes: `description` is typed as a `ServiceDescription`, `implementation` as a `ServiceImplementation`, etc.

The `ServiceDescription` contains a description of the service's authentication mechanism, its data storage mechanism, and incoming and outgoing data flows. Since O-Telos attributes can contain multiple values, any descriptive element can describe multiple instances. The elements contained in the `ServiceDescription` represent the design view.

The implementation view is captured by the `ServiceImplementation`, which refers to the implementation details through which data flows, how data is stored, and what authentication mechanisms are used.

Using ConceptBase's GraphEditor, we can visualize the full model as shown in Figure 3. Rectangles represent named objects, black arrows with a solid head represent named attributes, and magenta open-headed arrows represent generalization relationships.

The graph defines the central `Service` concept

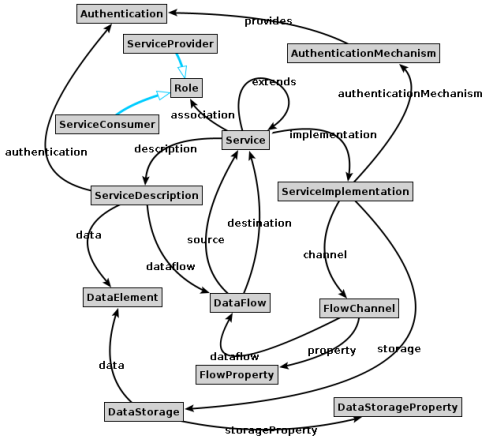

Figure 3: Model

as above, but also contains definitions for the other key modeling primitives. Specifically, it defines that a `ServiceDescription` is associated with a `Service` through its `description` attribute. The Service Description contains `DataElement`s that are stored by the `Service`, and it describes the nature of the `DataFlow`s and `Authentication` requirements. `DataElement`s are described by properties. `DataFlow` can be defined as having a source `Service` and a destination `Service`, as well as a brief description of the flow. Note that at this level, no technical details are included.

Likewise, the `ServiceImplementation` consists of a description of the mechanism used for `DataStorage` and of the mechanism by which the `Service` communicates with other `Service`s. The model also maps the service's descriptions to their implementations. For example, when considering the definition of the `FlowChannel`, which describes the technical properties through which as dataflow travels, the mapping to that `DataFlow` is recorded as well.

4.2.2 Case Study Implementation

To support the proof-of-concept implementation of the model, we once again use ConceptBase. However, rather than defining the model at a class level, we focus on instance levels. In this example, we implement the `MessagingService` and associate it with its `ServiceDescription` and its `ServiceImplementation`.

Due to space limitations, we only include a partial representation of the case study in this paper. The definition of the messaging service, as is used by both employees and clients to interact with each other, is captured in figure 4. A sample of corresponding O-Telos frames in provided in Listing 2.

The `ServiceDescription` describes the `MessagingService` as one that is used by `Client` and `Employee`, which are instances of the

```
MessagingDescription in ServiceDescription with
dataflow
  employeeMessagingFlow: MessagingFlow
data
  messagingData: MessagingData
authentication
  sso: SingleSignOn
end

MessagingWebChannel in FlowChannel with
dataflow
  dataflow: MessagingFlow
property
  protocol: HTTPS;
  tcp: TCP443
end

MessagingImplementation in
    ServiceImplementation with
channel
  webChannel: MessagingWebChannel
authenticationMechanism
  onPremSSO: OnPremSSOServer
end

MessagingService in Service with
description
  description: MessagingDescription
implementation
  implementation: MessagingImplementation
association
  usedByClient: Client;
  usedByEmployee: Employee

end
```

Listing 2: Case study definitions

`ServiceConsumer Role`. The service description identifies one data flow (`MessagingFlow`) and one primary data element (`MessagingData`). It also describes that the service may only be accessed after a service consumer authenticates using `SingleSignOn`.

The technical details describing the service are captured in its `ServiceImplementation`, which describes a `MessagingWebChannel` through which the `MessagingFlow` will travel. The channel uses the HTTPS protocol via its default TCP port, 443. The implementation details also describe the service as using an on-premise single sign-on server (`OnPremSSOServer`).

4.2.3 Using the model for analysis

Capturing a computing environment in the modeling approach outlined in this document helps an analyst visualize the interactions between services and actors. It also provides a mechanism for answering questions. For example, on December 12, 2020, FireEye reported a security problem that was related to its

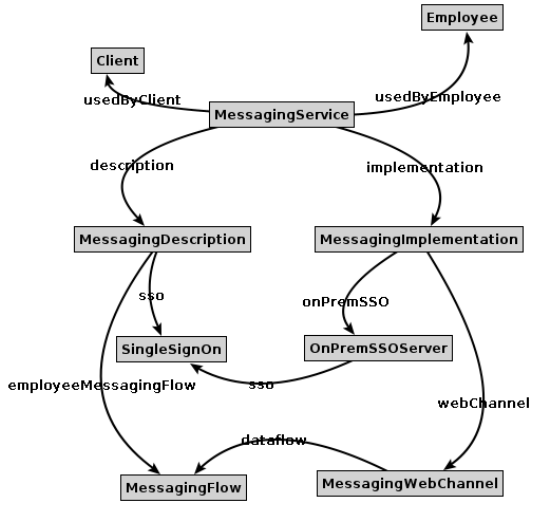

Figure 4: Case study

SolarWinds software (Thomson, 2020; Zetter, 2020). Unit42, the Threat Intelligence Unit of Palo Alto Networks provides several publicly available Indicators of Compromise that are likely associated with the breach (jadefeather, 2020). Specifically, the IOCs contain suspicious Internet domains, as well as file hashes to consider.

Developing an O-Telos query for this situation is relatively easy, as domain names are associated with flow channels.

```
SolarWindsQuery in QueryClass isA Service with
constraint
    r: $   exists si/ServiceImplementation
             (this implementation si)
       and exists fc/FlowChannel
             (si channel fc )
       and exists fp/FlowProperty
             (fc property fp )
       and (
          (fp in FlowChannelDestination) or
          (fp in FlowChannelOrigin)
       ) and (
          (fp domain "databasegalore.com") or
         (fp domain "solartrackingsystem.net")
       ) $
end
```

The query uses the logic to limit the output to only those service that match the constraints specified above. First, the inheritance specification (`isA Service`) ensures only Services are returned. Next, we limit the output set to only those services that have a service implementation that contains a flow channel with an origin or a destination domain that includes at-risk domains.

Using this logic, it becomes trivial to perform threat analysis by querying the model against any property to identify specific services.

In a more conventional threat modeling approach, for example, one based on STRIDE-analysis of data flow diagrams, this interaction would have been difficult to capture. Typically, external systems like Solarwinds network management equipment, are represented as terminals, with limited information represented about them. Interactions between that terminal and the system for which the threat model would be developed would primarily focus on that system's expected functionality and not be readily usable for analysis such as shown above.

5 DISCUSSION

When DFD-based threat models are used during requirements analysis and to design software-based systems, interactions with third-party systems are typically modeled as terminals. Terminals are generally not described in detail, as they are only considered as points-of-contact with the environment in which the newly developed system will function. As such, only data flows to and from the terminal are modeled. In a deperimeterized scenario, such reasoning is no longer sufficient, and it is often necessary to capture additional details about such systems. SolarWinds, as a network management tool, would be modeled as such an external terminal, if it were modeled at all. Our approach, which is able to capture additional relevant aspects of such interactions, would be more suited for analysis.

During the implementation of the case study, it became clear that manual model creation and analysis does not easily scale. To that effect, it will be necessary to explore how modeling efforts can be supported by analyzing electronic data sources. It is easy to imagine a situation in which the implementation view can be generated and/or supported with observations derived from sources like netflows, application logs or authentication logs, while the design view continues to be primarily analyst-driven.

It also rapidly became clear that even simple case studies become too complex to be fully comprehensible by human analysts. While the modeling primitives are intuitive, easy to understand, and can be applied without much effort, resulting models may be too large for the cognitive abilities of most human analysts.

However, since the models are represented in a computer-parseable way, they lend themselves for automated analysis. Several different approaches to such analysis can be adopted. For example, the models provide a powerful mechanism to support decisions made by analysts through their query mechanism. In addition, it is conceivable that supervised

machine learning algorithms can be developed to analyze the models to detect significant anomalies.

One of the more pragmatic methods by which these models can be used is by mapping existing indicators or compromise to the implementation model views. For example, IoCs describing specific IP addresses, host names, port numbers, protocols, software versions, etc. map relatively easily to flow channel properties, as shown in section 4.2.3.

Extending the model with role-based access control analysis techniques should be fairly easy to accomplish as well. The model accounts for the possibility to map actors to roles, and to associate specific service provider roles and service consumer roles with services.

6 CONCLUSIONS AND FUTURE WORK

The confluence of increased adoption of cloud services, paired with the growing prevalence of end-to-end encrypted network communications and the surge in telecommuting activities has resulted in a significant drop in the efficacy of perimeter-based controls. This process of deperimeterization requires threat analysts to rethink how they achieve and maintain situational awareness, analyze threats, and design and build countermeasures.

This observation supports our long-term research objective to evolve threat modeling into providing meaningful support to defenders operating deperimeterized enterprise computing environments.

In this paper, we looked at the first stage of this research: determining how enterprise computing landscapes can be described. In search of an answer, we developed a service-oriented threat modeling approach that can be used to support threat modeling.

The central concept of the approach is to adopt a service-oriented perspective, rather than a network-centric approach. We model services, data flows, data storage, and authentication systems conceptually (i.e., technology-agnostic), and then proceed to enrich the model using technical implementation details. The technical details provide an operational perspective, which is mapped to the conceptual overview that provides the design view.

Specifically, our paper makes the following contributions: We adopt a service-oriented perspective, and specifically that of a service consumer. Most existing threat modeling methods support analysis and design of software-based solutions, while we advocate extending the use of threat modeling into the realm of security operations. Our approach is specifically intended to capture interactions between services, regardless of their ownership, or the platform through which they are provided. Adopting a service-oriented focus will allow us to capture security-relevant properties, other than data flows. In the proof-of-concept implementation, we captured data-at-rest and authentication mechanisms as well. In addition to modeling data flows, we capture the properties of the channels through which these data flows travel. Doing so will allow for more comprehensive analysis and facilitates mapping to threat intelligence.

Automation will play a critical part in supporting model maintenance and in threat analysis. Incorporating automation is the subject of future research.

REFERENCES

Alberts, C. J., Dorofee, A. J., Stevens, J. F., and Woody, C. (2003). Introduction to the OCTAVE Approach. Technical report, Carnegie Mellon University Software Engineering Institute.

Bouguettaya, A., Singh, M., Huhns, M., Sheng, Q. Z., Dong, H., Yu, Q., Neiat, A. G., Mistry, S., Benatallah, B., Medjahed, B., Ouzzani, M., Casati, F., Liu, X., Wang, H., Georgakopoulos, D., Chen, L., Nepal, S., Malik, Z., Erradi, A., Wang, Y., Blake, B., Dustdar, S., Leymann, F., and Papazoglou, M. (2017). A service computing manifesto: The next 10 years. *Commun. ACM*, 60(4):64–72.

Caltagirone, S., Pendergast, A., and Betz, C. (2013). The Diamond Model of Intrusion Analysis. Technical report, Center of Cyber Intelligence Analysis and Threat Research.

Dhillon, D. (2011). Developer-Driven Threat Modeling. *IEEE Security & Privacy*, 9(4):41–47.

Google (2020). HTTPS encryption on the web. Technical report, Google.

Hussain, S., Kamal, A., Ahmad, S., Rasool, G., and Iqbal, S. (2014). Threat Modelling Methodologies: A Survey. *Science International*, 26(4):1607–1609.

IDG (2020). 2020 IDG Cloud Computing Survey. Technical report, idg.

jadefeather (2020). Unit42 solarstorm iocs. Technical report, Palo Alto Networks Unit 42.

Jarke, M., Gallersdörfer, R., Jeusfeld, M. A., Staudt, M., and Eherer, S. (1995). ConceptBase — A deductive object base for meta data management. *Journal of Intelligent Information Systems*, 4:167–192.

Kazim, M. and Evans, D. (2016). Threat Modeling for Services in Cloud. In *Proceedings 2016 IEEE Symposium on Services-Oriented System Engineering (SOSE)*. IEEE.

Leune, K. (2007). *Access Control and Service-Oriented Architectures*. PhD thesis, Tilburg University, CentER.

Leune, K. and Kim, S. (2020). Service-oriented modeling for cyber threat analysis. In *Proceedings of the*

Tenth ACM Conference on Data and Application Security and Privacy, CODASPY '20, page 150–152, New York, NY, USA. Association for Computing Machinery.

Malik, N. A., Javed, M. Y., and Mahmud, U. (2008). Threat modeling in pervasive computing paradigm. In *2008 New Technologies, Mobility and Security*, pages 1–5.

Mirembe, D. P. and Muyeba, M. (2008). Threat Modeling Revisted: Improved Effectiveness of Attack. In *Proceedings of the Second UKSIM European Symposium on Computer Modeling and Simulation*, pages 93–98. IEEE.

Myagmar, S., Lee, A. J., and Yurcik, W. (2005). Threat Modeling as a Basis for Security Requirements. In *Proceedings of the Symposium on Requirements Engineering for Information Security (SREIS 2005)*.

Mylopoulos, J., Borgida, A., Jarke, M., and Koubarakis, M. (1990). Telos: Representing Knowledge About Information Systems. *ACM Transactions on Information Systems*, 8(4):325–362.

Papazoglou, M. (2003). Service-oriented computing: concepts, characteristics and directions. In *Proceedings of the Fourth International Conference on Web Information Systems Engineering, 2003. WISE 2003.* IEEE.

Papazoglou, M. and Georgakopoulos, D. (2003). Service-Oriented Computing. *Communications of the ACM*, 46(10):25–28.

Rouland, Q., Hamid, B., and Jaskolka, J. (2020). Reusable Formal Models for Threat Specification, Detection, and Treatment. In *Reuse in Emerging Software Engineering Practices*, pages 52–68. Springer International Publishing.

Schneier, B. (1999). Attack Trees. *Dr. Dobb's Journal*.

Shevchenko, N., Chick, T. A., O'Riordan, P., Scanlon, T. P., and Woody, C. (2018). Threat Modeling: A Summary of Available Methods. Technical report, Carnegie Mellon University Software Engineering Institute.

Shostack, A. (2007). Experiences Threat Modeling at Microsoft. Technical report, Microsoft.

Shostack, A. (2014). *Threat Modeling: Designing for Security*. Wiley.

Sion, L., Landuyt, D. V., Wuyts, K., and Joosen, W. (2019). Privacy Risk Assessment for Data Subject-Aware Threat Modeling. In *IEEE Security and Privacy Workshows (SPW)*, pages 64–71.

Steven, J. (2010). Threat Modeling—Perhaps It's Time. *IEEE Security & Privacy*, pages 83–86.

Thomson, I. (2020). SolarWinds releases known attack timeline, new data suggests hackers may have done a dummy run last year. *The Register*.

Torr, P. (2005). Demystifying the Threat-Modeling Process. *IEEE Security & Privacy*, 3(5):66–70.

UcedaVelez, T. and Morana, M. (2015). *Risk Centric Threat Modeling: Process for Attack Simulation and Threat Analysis*. Wiley.

Zetter, K. (2020). Hackers last year conducted a 'dry run' of SolarWinds breach. *yahoo!news*.