universität wien

MASTERARBEIT / MASTER'S THESIS

Titel der Masterarbeit / Title of the Master's Thesis

„Forecasting the Development of XML-based Intrusions Using Models of Attack Patterns "

verfasst von / submitted by

Luka Plepel Markovic, univ. bacc. inf.

angestrebter akademischer Grad / in partial fulfilment of the requirements for the degree of

Master of Science (MSc)

Wien, 2019 / Vienna 2019

Studienkennzahl lt. Studienblatt / degree programme code as it appears on the student record sheet:	UA 066 926
Studienrichtung lt. Studienblatt / degree programme as it appears on the student record sheet:	Masterstudium Wirtschaftsinformatik
Betreut von / Supervisor:	Univ.-Prof. Dipl.-Ing. Dr. Dr. Gerald Quirchmayr

Zusammenfassung

Viele Unternehmen verwenden täglich Extensible Markup Language (XML) und diese Abhängigkeit von XML führt zu unterschiedlichen Angriffsvektoren. Das Auffinden dieser potenziellen Angriffsvektoren ist ein entscheidender Schritt bei der Entwicklung eines sicheren Systems, da ein erfolgreicher Angriff erheblichen Schaden verursachen kann. Die Ziele dieser Angriffe können Denial of Service (DoS), das Erhöhen von Berechtigungen, das Zugreifen auf, Lesen und Schreiben von Daten und das Ausführen nicht autorisierter Befehle sein. Sozio-Technische Systeme machen die Angriffsanalyse zu einer besonderen Herausforderung, da sich komplexe Systeme aus Personen, Software und physischen Infrastrukturen zusammensetzen. Daher muss bei einer gründlichen Angriffsanalyse jeder Aspekt des Sozio-Technischen Systems berücksichtigt werden. Umfangreiche Sicherheitskenntnisse sind erforderlich um das gesamte System zu verstehen. Diese Arbeit befasst sich mit allen oben genannten Herausforderungen. Nach einer umfassenden Literaturanalyse haben wir das gesammelte Wissen genutzt, um elf realistische und detaillierte XML Angriffe zu identifizieren, die dann modelliert und weiter erläutert wurden. Jedes der elf Angriffsmuster wurde aus Sicht des Angreifers erstellt. Hierzu wurde ein dreischichtiges Model Framework zusammen mit einem umfassenden Repository für Angriffswissen verwendet. In dieser Arbeit werden die einzelnen Angriffsmustermodelle detailliert modelliert und erläutert. Das Ergebnis unserer Arbeit ist ein in Python integrierter Prognoseprototyp in einem Jupyter Notebook-Dokument, in dem alle relevanten Informationen zusammen mit den Modellen der Angriffsmuster gespeichert sind. Die Funktionen des Prototyps lauten wie folgt: Manuelle Prognosefunktion basierend auf Aufgaben, automatische Prognosefunktion basierend auf einer Windows Log Datei, Anzeige einer Angriffsmustermodellfigur und Zugriff auf relevante Informationen zu einem Angriffsmuster. Ziel dieser Arbeit ist es, Sicherheitsexperten bei der forensischen Analyse nach einem Angriff zu unterstützen und dabei zu helfen, den Schaden während eines Angriffs zu erkennen und zu mindern.

Abstract

Many businesses use Extensible Markup Language (XML) daily and, this reliance on XML creates distinct attack vectors. Discovering these potential attack vectors is a crucial step in engineering a secure system since a successful attack could cause significant damage. The goals of these attacks can be a Denial of Service (DoS), elevating privileges, accessing, reading and writing data, and executing unauthorized commands. Socio-Technical Systems make attack analysis particularly challenging since any complex systems are composed of people, software, as well as physical infrastructures. As such, a thorough attack analysis needs to consider every aspect of Socio-Technical Systems. To take the whole system into account, a large amount of security knowledge is required. This thesis tackles all the challenges mentioned earlier. After a comprehensive literature analysis was conducted, we used the available knowledge to identify eleven realistic and detailed XML attacks, which were then modeled and explained further. Each of the eleven attack patterns was created from the attacker's perspective. The models were created by using a three-layer modeling framework, along with a comprehensive attack knowledge repository. In this thesis, we model and explain each attack pattern model in detail. The result of our work is a forecasting prototype programmed in Python in a Jupyter Notebook document, which stores all the relevant information along with the models of the attack patterns. The prototype's functions are as follows: manual forecasting function based on tasks, automatic forecasting function based on a windows log file, displaying an attack pattern model figure, and access to relevant information about an attack pattern. This thesis aims to aid security professionals in their forensic analysis after an attack. Additionally, the goal is to help detect and mitigate the damage while an attack is happening.

Acknowledgment

This thesis was written and carried out during the summer of 2019 at the University of Vienna as part of the Master's degree program Business Informatics. The year of research and hard work spent have been a fascinating period that helped me understand much of today's security field. This thesis presents an implementation of a three-layer modeling framework for modeling attack patterns with the aim of assisting system professionals and administrators.

Furthermore, this thesis could not have succeeded without the helpful advice of others, and therefore, I wish to express my sincere gratitude to every single person who supported me in the completion of this thesis. First and foremost, I would like to thank my supervisor, Univ.-Prof. Dipl.-Ing. Dr. Dr. Gerald Quirchmayr at the University of Vienna, Faculty of Computer Science, Research Group of Multimedia Information Systems. He has provided me with inspiration, guidance, perspective, and insight in the security field throughout the writing of this thesis.

At this point, I would also like to sincerely thank my family and friends for their encouragement, helpfulness, and for believing in me.

November 2019
Luka Plepel Markovic

Contents

List of Figures

1 Introduction

The goal of this thesis is to develop a tool-supported framework to forecast the development of intrusions based on attack patterns and to create a library of said attack patterns. The first primary source of inspiration for this framework is the Common Attack Pattern Enumeration and Classification repository [31]. The second source of inspiration are the collective papers written by Li et al. [10, 11, 12, 13, 14]. Based on these sources, a proof of concept was developed in Python. The current state of an intrusion can be guessed by matching the gathered information against the attack pattern repository. The underline assumption of this thesis is that a series of snapshots will allow the forecasting of an intrusion's development. Furthermore, in the early stages of pattern matching, an attack may match a more significant number of possible attacks. A more advanced stage of an attack reduces the number of alterations to a small enough number that a most likely candidate can be forecast. The framework will, therefore, be more efficient after each stage of an attack. In the first step, a literature analysis was conducted to get an overview of the research which this thesis builds on, mainly in the field of attack pattern matching, information gathering, and information fusion. The literature analysis, which forms the basis of the research contribution made by this thesis, covers the following areas:

1. Attack pattern libraries in use
2. Intrusion detection
3. Modeling of security mechanisms
4. Variation test of security solutions

The literature analysis concluded by identifying the primary building blocks needed to develop and implement a prototype.

The framework built in this thesis consists of a set of core intrusion analysis models, as well as modeling guidelines and the accompanying prototype tool. The modeling approach used in this thesis is based on standard Unified Modeling Language (UML) diagrams. The models were created with the help of an attack pattern repository, and they have been validated against the existing literature. Finally, a proof of concept prototype was written in Python in a Jupyter Notebook container and tested. The achieved results are discussed in a later chapter. A conclusion and future work can be found at the end of this thesis.

An attack pattern is a type of pattern that is created from an attacker's viewpoint. It consists of what the attacker wants to achieve called the "problem", what functions an attacker needs to use to accomplish the goal called a "context" and lastly, the steps an attacker performs to attack our system called the "solution". Based on these three parts of an attack pattern, we have identified relevant elements that specify the problem, context, and solution, and we have included them in our model [23, 24]. We predict this will assist in stopping the attack or hindering it. A Socio-Technical System (STS) is a complex system that consists of people, processes, software, and hardware. These systems are diverse and constantly changing due to many goals and restrictions that have to be met for a business to operate smoothly and successfully. One of the main reasons for successful breaches is the complexity of the organization itself. These components raise several security concerns and present a larger attack surface compared to just software systems.

For this reason, the works of Li et al. [10, 11, 12, 13, 14] propose a three-layer requirements framework, which models and analyzes requirements of STS at the business layer, software application layer, and physical infrastructure layer. This approach is considered a holistic approach since it should encapsulate the business and the security requirements in their entirety. The model aims to set goals from the perspective of the attacker, which then helps identify attack vectors. The goals

from an attacker's perspective are also called anti-goals.
Sun Tzu [20] said:

> "If you know the enemy and know yourself, you need not fear the result of a hundred battles. If you know yourself but not the enemy, for every victory gained you will also suffer a defeat. If you know neither the enemy nor yourself, you will succumb in every battle."

After all, if we know what the aim of the attacker is, then we can compensate for the attack or even prevent it outright.

The following chapter will discuss relevant related work. Chapter 3 will be a modeling approach that will involve the explanation of the proposed modeling framework. Chapter 4 will consist of detailed explanations of each of the eleven XML attack pattern models. Following this, chapter 5 describes the usage, workings, and results of the prototype. This thesis concludes with our findings.

2 Literature Analysis

The modeling framework used in this thesis consists of three layers: the business layer, the application layer, and the physical layer. These three interconnected layers are sufficient to capture the complexity of an STS. They are used to model and analyze the requirements for each layer. The end model [12] resembles a UML model with domain assumptions, tasks, and goals from the perspective of the attacker. The reason why this method was chosen is described in detail below.

2.1 Attack Pattern for Voice over Internet Protocol (VoIP)

In the paper by Fernandez et al. [6], they use UML class and sequence diagrams to explain attack patterns, and as an example, they use a Denial of Service attack on an instance of the Voice over Internet Protocol. They do this by going into very fine detail. Each pattern is described using ten attributes. These attributes are: name of the attack, disruption intent, context, problem, solution to the problem, known uses, consequences of the attack, countermeasures and forensics, evidence locations and related patterns.

The name of the attack describes what the attack is about. The intent is a short description of the purpose of the intended attacks. Context is a description of the conditions in which the attack takes place and encompasses the environment, parameters, conditions, defenses, and in some cases, known vulnerabilities. Furthermore, the paper describes the "problem" which is the goal of the attack pattern, also referred to as the anti-goal. The reason why this is defined as the "problem" is that the attacker has the problem of wanting to attack a target system, or rather how to get into the system. The problem can then be further divided up into smaller sub-problems. The attacker might need to create sub-problems to address different aspects of the problem, such as hard to breach defenses of the system, exploits to target, how to navigate the system, and so on. All of these obstruct or delay the

success of the attacker and his anti-goals. The "solution" segment describes the solutions an attacker may use to solve his problem of getting in, how the attack is performed, and the results of these actions. UML class and sequence diagrams are used to explain the system before and during the attack and the messages exchanged during the attack, respectively. Additionally, Fernandez et al. [6] describe known uses for attacks, where to find evidence, consequences, and the benefits and drawbacks of the attack for the attacker, with detailed attention given to the effort, cost, and possible sources of failure. In the segment on countermeasures and forensics where the description of measures taken to stop, mitigate, and trace the attacker is described. This chapter also includes where to obtain information on each stage when determining the attacker's steps. This information can be used to help identify the specific action taken by the attacker. In the segment titled, evidence locations, UML class diagrams, and associations of the relevant parts are used for forensic investigation purposes.

Since UML class diagrams abstract well, they are used for showing only parts of the system relevant to the investigation. The related patterns chapter of the modeling method includes patterns that have different anti-goals but are performed similarly and attacks that have a similar objective but differ in the approach of the attack. Fernandez et al. [6] additionally use this modeling method to describe an example of a VoIP Denial of Service attack with every part of the mentioned modeling method used to describe and examine the attack pattern in detail. The paper concludes with a discussion where they propose that forensic investigators could use this method to search for evidence. Fernandez et al. [6] state that investigators often find it challenging to determine which data should be collected and that collecting this data is arduous. The reason for this is that it involves identifying all the components involved in the attack, deciding which are most likely of interest, finding the location of the components, and collecting the data from each component [6, 9]. The value of this proposed method is in the fact that combining the class and sequence diagrams

guides forensic investigators on what evidence can be found after an attack and where to look for the evidence. The two diagrams need to be combined due to an attack being described dynamically in a sequence diagram but making direct reference to the system components, which are described in the class diagram of the modeling method.

2.2 Three-Layer Model

Security patterns are similar to software engineering design patterns but incorporate security knowledge that helps analysts tackle security problems. Li et al. [15] state that these security patterns are quite thoroughly researched and that this research has produced an impressive collection of patterns. The issue is that in practice, these security patterns are not widely applied. To solve this problem, they propose to integrate security patterns with goal-oriented security requirements analysis [15]. Since there is a lot of existing research in this field, we will instead focus on attack patterns from the perspective of attackers and the attackers' anti-goals.

In comparison to the modeling framework by Fernandez et al., the framework developed by Li et al. [12] is more streamlined. Their framework consists of a context, a problem, and a solution that can be applied to solve that problem in the given context. This thesis builds upon the modeling framework described in papers by Li et al. [10, 11, 12, 13, 14] because it is more straightforward and easy to explain, which makes is ideal for beginners in the security field and for highlighting the importance of including security measures in every part of software development. Analyzing attack vectors from an attacker's perspective has been accepted as a practical approach for dealing with security requirements for complex systems. The problem is that there is no systematic approach to constructing attack scenarios. This results in the attack scenarios being subject to the knowledge and experience of experts to be complete. Li et al. [14] propose a systematic process for identifying attack scenarios to support security analysis, founded on anti-goal

refinement. This comprehensive anti-goal refinement framework would consist of five anti-goal refinement patterns and an analysis process for using the patterns as part of an encompassing security design [14].

In another paper, Li et al. [16] outline a toolkit that would enable semi-automatic anti-goal refinement requiring only the attack vectors as input. In the security requirements analysis in each layer, related threat information is necessary to identify critical security requirements. For this purpose, a toolkit was created, and they used it to model several attack patterns as a proof of concept. The models created with the toolkit can provide information about the system under attack and help to identify realistic attacks on the system across all three layers. The identified attacks can then be transferred back to the three-layer security requirements framework, aiding the analysis of critical security requirements and enabling the creation/generation of security solutions. The prototype tool used by Li et al. is a Java application that incorporates a powerful diagramming application called OmniGraffle. The prototype is called the Multi-layer Security Requirements analysis tool (MUSER). For this thesis, the MUSER tool is not used. Instead, a new Python-based prototype was implemented in a Jupyter Notebook document that works for XML attack patterns.

Attack patterns are similar to security patterns but differ in that they show the event from the viewpoint of the attacker, i.e., what an attacker wants to attack and how the attacker performs the attack. Based on the semantics of attack pattern attributes, we have identified relevant attack pattern attributes that specify the problem, the context, and the solution, and indirectly map them to the contextual goal model elements. This modeling method looks similar to the model framework proposed by Ali et al. [2]. A detailed explanation of the model layers and objects used in the models in this thesis will be provided in chapters 3 and 4, followed by an overview of ten more XML attack patterns also used in the thesis. These models were created as part of this thesis and are based on information found in

the Common Attack Pattern Enumeration and Classification repository. All eleven models are stored in an XML database file. This database acts as a repository for our prototype and can be added to if more models are created. This is done to increase the modularity and extensibility of the prototype for future work purposes.

2.3 Computer Aided Threat Identification

In their paper, Asnar et al. [3] state that a need for automation of global threat identification exists. The paper states that there has been an increase in reported security threats hitting organizations. Logically, the larger the organization, the more attacks it experiences. Some originated from giving inappropriate permissions on sensitive organizational data to users that should not possess such permission in the first place. Thus, organizations must recognize as early as possible the risks deriving by inappropriate access right management and identify the solutions that they need to prevent such risks.

Asnar et al. [3] propose a framework to identify threats during the requirements analysis of an organization's IT systems. They state that their framework does not rely on the level of expertise of the security analyst to detect threats, but allows to automatically identify threats that derive from improper access management, in contrast to other existing frameworks. To capture the organization's setting and the system stakeholders' requirements, they adopt a framework called SI*. This is a requirement engineering framework focused on the concepts of actors, goals, tasks, and resources. This framework is similar to the one used in our thesis, but it focuses more on attacks themselves and how to solve them from the perspective of the system holders. Due to this, the framework is not suitable to model our intended attacker's point of view. Asnar et al. extend the SI* with a technique that identifies potential security threats that might impact resources management and other relevant goals. The reasoning is based on Answer Set Programming (ASP) logic rules that take into account the relationships between resources and the delegation of permission

relations between actors. Generally, they focus more on the intricacies of controls and permission assignments. These intricacies typically become more complex as the size of the company increases, the number of systems that are implemented expands, and the granularity of access rights within the enterprise spanning systems grows.

They continue that in organizations, the managing of access rights while protecting sensitive organizational information, is a daunting process for security administrators, where we tend to agree with them. Many users may not be aware of their access rights or conflicts and issues that their rights could create. In most cases, users are granted access to sensitive information, but the owner of that information does not trust them. It is relatively simple for the users to misuse their permissions, which can then cause security compromises in the organization's assets. Examples of threats that can hit an organization because of an inappropriate access management rule are unintended data editing, unintentional or intentional disclosure of secret information, or internal misuse for personal gain. It is crucial to identify pitfalls in permissions assigned to users in the early phases of an information system engineering process and manage the level of risk by adopting suitable security solutions so that these types of threats are avoided. As stated above, Asnar et al. [3] propose the use of a framework to identify threats that are caused by inappropriate permission assignments. They identify the relevant threats that occur during the requirement analysis of the organization's settings and its IT systems [3].

2.4 Matching Attack Patterns

Gegick et al. [7] state that in general, the cost of finding and fixing software bugs in completed products can be 100 times more expensive than solving the problems while in the design and implementation stages of a project [7]. The path usually taken to fix such issues is with patches that come out at a later date, after the initial release. This can incur problems since patches could introduce new, unforeseen

vulnerabilities. Another problem is that someone could find and exploit the original weakness that needed patching in the first place. Another technique is to rely on external devices or software systems such as intrusion detection systems and firewalls. Patching is the preferred method today, but there are glaring issues when using this method. For example, the biggest issue is that such protections only work when the end software project is complete. A question then arises, what happens when those solutions also have weaknesses or can not detect a particular attack.

Usually, attacks are based on a few known variations, which in turn gives the impression that there should also be few weaknesses, or at least that they should share many similarities. Alexander's [1] idea:

> "A pattern describes a situation where a problem recurs and provides a solution that can be applied regardless of the environment in which the problem occurs." [7]

This general pattern definition Alexander coined can be stated even today. Alexander used this definition in the context of building architecture in 1977, but the general concept can be used in any field, and thus, computer science and security adopted this approach as well. As stated before, many authors use security patterns and define them as a particular recurring security problem that arises in specific contexts and presents a well-proven solution.

In the paper Gegick et al. [7] propose a method for early identification of system vulnerabilities that they call regular expressions based on the descriptions of attack patterns and using those regular expressions as general representations of those attacks. Attack pattern expressions are events that are symbolized by components in the system triggering the attack event [7]. Gegick et al. reference a paper from 1975 by Carlstedt et al. [4] that proposes abstracting system calls, data stores, and other entities in operating system source code into generalized patterns, so that

they can be applied to any operating system. The aim of this abstraction is that each operating system would fundamentally work the same, but could then have different names for those parts. The findings proved that if done correctly, experts could impart needed knowledge of security flaws to those who had little to no prior understanding or experience with them. This was done to prove that the same abstracting to regular expressions was not a new idea.

Furthermore, they mention attack trees and attack nets could be used as graphical representations of attack patterns, but ultimately decide to steer clear. Gegick et al. mention that attack trees do not mimic software system design closely enough. Attack trees are tree diagrams where the nodes show the goals of an attacker. For our thesis, this could be useful and thus was looked more into. The conclusion is that our models have a similarity with attack trees, but no further use was found. Attack trees show the point of view of an attacker completing steps toward the final goal. Software engineers can use attack trees to determine if such a sequence of objectives is possible in their software systems [19]. The second mentioned graphical structure are Attack nets, which are an abstract representation of exposing vulnerabilities in software systems. Attack nets are different from attack trees because attack trees are more interested in the attacker's anti-goal, while attack nets are focused on specific places in the system where penetration testing should be carried out to find weaknesses. Another difference is that Attack nets show more detail of a particular software system part that is of interest but also incorporate the knowledge of the system architects as well as the attacker's point of view, thus being narrower in scope, but deeper in detail [17]. The paper continues to explain how to create the needed regular expressions and how Gegick et al. use them. For this thesis, the regular expressions and attack nets are not used.

2.5 Assessing and Exploiting XML Schema

Both Righetto [26] and Arnaboldi [27] state that the specifications for both XML and XML schema, like XML Schema Definition (XSD) and RelaxNG (RNG) [30], include multiple security flaws, but also at the same time, these specifications provide the tools necessary to protect XML applications. Even though we use the XML schema to define the security of XML documents, they can be used to perform a variety of attacks: file retrieval, server-side request forgery, port scanning, or brute-forcing to name just a few. Righetto wrote up a beautiful and concise "cheat sheet" to expose how to exploit the different possibilities in libraries. Arnaboldi wrote a much more extensive paper that takes an in-depth look at the same problems. Both divide the issues into two sections:

- Malformed XML Documents: vulnerabilities using documents that do not adhere to proper formatting,
- Invalid XML Documents: vulnerabilities using documents that do not have the expected structure.

2.5.1 Malformed XML Documents

The World Wide Web Consortium (W3C) XML specification defines a set of principles that XML documents must follow to be considered well-formed. When a document violates any of these principles, it must be considered a fatal error, and the data it contains is deemed to be malformed [29]. Multiple tactics will cause a malformed document, some of which are: removing an ending tag, rearranging the order of elements into a nonsensical structure, introducing forbidden characters. The XML parser should stop the execution of the XML document once it detects a fatal error. The parser should stop any ongoing processing of the document, and the application should display an error message. The recommendations to avoid these vulnerabilities are to use an XML processor that follows W3C specifications and does not take significant additional time to process malformed documents. Also, use only

well-formed documents and validate the contents of each element and attribute to process only valid values within predefined boundaries [26, 27, 29].

A malformed document may affect the consumption of CPU resources since it would take significantly more time to process that document. In specific scenarios, the amount of time required to process malformed documents may be higher than that required for well-formed documents. In our experience, the larger the document is, the longer the parser takes. This increases even further with malformed documents. This process is highly dependent on the speed of the CPU, as well as the number of threads the application can use to run in parallel. An attacker may exploit this in what is knows as an asymmetric resource consumption attack to take advantage of the more significant processing time, which is a type of Denial of Service (DoS). To analyze the likelihood of this attack, one can analyze the time taken to parse a regular XML document vs. the time taken by a malformed version of that same document, then consider how an attacker could use this vulnerability in conjunction with an XML flood attack using multiple documents to amplify the effect. Individual XML parsers can recover malformed documents. They can be instructed to try their best to return a valid tree with all the content that they can manage to parse, regardless of the document's noncompliance with the specifications. Since there are no predefined rules for the recovery process, the approach and results may not always be the same. Using malformed documents might lead to unexpected issues related to data integrity and information leakage [26, 27].

2.5.2 Invalid XML Documents

Attackers may introduce unexpected values in documents to take advantage of an application that does not verify whether the document contains a valid set of values. XML schemas specify restrictions that help identify whether documents are valid and comply with the formative rules. A valid document is well-formed and complies with the restrictions of a schema, and more than one schema can be used to validate

a document. These restrictions may appear in multiple files, either using a single schema language or relying on the strengths of the different schema languages. The recommendation to prevent these vulnerabilities is that each XML document must have a precisely defined XML schema with every piece of information adequately restricted to avoid problems of improper data validation. It is recommended to use a local copy or a known excellent repository instead of the schema reference supplied in the XML document. Performing an integrity check of the XML schema file being referenced is advised, bearing in mind the possibility that the repository could be compromised. In cases where the XML documents are using remote schemas, configuring servers to use only secure, encrypted communications like Hypertext Transfer Protocol Secure (HTTPS) to prevent attackers from eavesdropping on network traffic [26, 27] is strongly advised.

In this thesis, we will heavily lean on the XML standard and specification for both XML schema and the general XML standard. Righetto's [26] and Arnaboldi's [27] papers were of use in the identification of problems and attack vectors and thus were used in the completion of attack patterns where needed.

2.6 Detecting Anomalous Patterns in XML Documents

XML transactions could become victim to cyber-attacks, or even benign mistakes could happen where the integrity of the XML becomes questionable. This alteration in the structure and content of XML documents is the aim of this thesis. The goal is to prevent or at least reduce damage, the impact of errors, and attacks. Regardless of whether the origin of these attacks is malicious or benign, the altered XMLs are, in fact, attack vectors that could cause real harm. Especially XML documents that adhere to an XSD schema since they can potentially be used to exploit vulnerabilities of the interacting information systems. Menahem et al. [18] state that the state of the art end-to-end security protocols for XML transactions, such as XML encryption [8], XML signature [5], and XML canonization [25] provide little protection against

such threats. They state that the reason for this is that, for the most part, the actions which result in deformation of the XML documents take place before such protective measures are applied at the endpoint's system. The XML documents that are bound to an XSD schema can vary in great detail. Two XML documents that adhere to the same XSD schema rules can have very different attributes regarding both their content and structure [18]. Thus, the importance of well-designed XSD schemas is highlighted.

Anomalous patterns are related to both the structure and content of an XML document. Such patterns can be generated by either malicious cyber-attacks, benign user mistake, or an error. Menahem et al. [18] name the most prominent XML attacks, which are generally subdivided into two parts: real XML attacks and benign anomalies. These attacks are weaknesses in the XML processing mechanism, such as the vulnerability of XML parsers or the weak points of input verification in the target server application. The attacks named are: input validation attacks, probing, malware infiltration, buffer overflow, XML parameter poisoning, Character Data (CDATA) field attacks, Structured Query Language (SQL) injection, cross-site scripting, schema poisoning, Denial of Service, Distributed Denial of Service (DDoS) XML bombardment, Document Object Model (DOM) parser DoS attacks, XML bomb and repetition attack [18]. These XML attacks are the main culprits that result in XML anomalies, since the attacks are expressed through string expressions that, concerning the standard XML transaction collections, are inherently very unlikely to be obtained. Another threat to modern information systems comes from data leakage. Among the causes of data leakage are Trojan attacks, SQL injection attacks, or simply human error. Not all XML anomalies are a product of a cyber-attack or a malicious action [18]. There are many ways in which XML documents might become anomalous. User mistakes, application errors, and communication errors are typical examples of how benign anomalies might be manifested in XML documents. It is necessary to keep in mind that these are still XML documents that are bound to an

XSD schema from the receiver.

Menahem et al. [18] state that it is vital to detect both types of anomalies. Mainly because XML anomalies have the potential to result in unwanted effects in the information processing system, regardless if they are benign or malicious. Anomaly detection is a process aimed at discovering patterns in data sets that deviate from the general behavior or the expected behavior of the majority of the data. Today, anomaly detection can commonly be found in various fields since it has a broad spectrum of applications. Some of the fields are general cyber-security, intrusion detection, fraud detection, financial systems, and military surveillance. Anomaly detection methods employ a wide range of techniques that are based on statistics, classification, clustering, the nearest neighbor search, spectral analysis, information theory, and many more.

The conclusion of their paper Menahem et al. [18] state that despite the risk XML documents anomalies impose, very little relevant research has been done in this area to date. They go on to name a few of the papers their research was based on and write about the contribution they propose the paper brings to the security field. The stated contribution of the paper is three-fold. First, it presents a general and automatic method for extracting structural and content features from XML transactions. Second, it provides a practical method for transforming XML features into vectors of fixed dimensions and hence enables the use of non-proprietary machine-learning algorithms for the XML anomaly detection task. Lastly, they present an anomaly detection algorithm, Attribute Density Function Approximation (ADIFA) [18].

2.7 Common Attack Pattern Enumeration and Classification

The Common Attack Pattern Enumeration and Classification (CAPEC) dictionary and classification repository is another significant building block of this thesis. It features over 500 different attack patterns described in detail, including how they work. Each attack patter also has additional references to external sources on a dedicated page for that attack. The CAPEC effort provides a publicly available catalog of common attack patterns that helps users understand how attackers exploit weaknesses in applications and other cyber-enabled capabilities [31]. Attack patterns are descriptions of the common attributes and approaches employed by adversaries to exploit known vulnerabilities in cyber-enabled capabilities [28].

Attack patterns define the problems laid out in the works of both Fernandez et al. [6] and Li et al. [12]. An attacker may face issues such as how to breach the security of organizations and how they go about creating solutions to those problems. Attack patterns are derived from the concept of design patterns applied in a disruptive fashion rather than a constructive one. An in-depth analysis of specific real-world exploits on specific examples needs to be done to create a formal attack pattern. Each attack pattern captures knowledge about how particular parts of an attack are designed and executed. Additionally, the attack pattern generally gives guidance on ways to mitigate the attack's effectiveness.

Attack patterns are created to help the development of applications and tools for administrating cyber-enabled capabilities to better understand the specific elements of an attack and how to stop them from succeeding, thus preventing damage [31]. Attack patterns captured in such a formalized way can bring considerable value to the development and maintenance of cyber-enabled capabilities, including [31]:

1. Training - Educate software developers, testers, buyers, and managers,
2. Requirements - Define potential threats,
3. Design - Provide context for architectural risk analysis,
4. Implementation - Prioritize review activities,
5. Verification - Guide appropriate penetration testing,
6. Release - Understand trends and attacks to monitor,
7. Response - Leverage lessons learned into preventative guidance.

Of course, attack patterns are not the only useful tool for building secure cyber enabled capabilities. Many other tools, such as misuse/abuse cases, security requirements, threat models, knowledge of common weaknesses and vulnerabilities, and attack trees, can help. Attack patterns play a unique role in this broader architecture of security knowledge and techniques [31].

3 Modeling Approach

In this chapter of the thesis, the modeling approach is explained. One of the models used in the thesis is provided as an example. This thesis will be focused on XML attack vectors. CAPEC lists several attack patterns, each of which was modeled, explained, and implemented in the Python forecasting prototype developed as part of this thesis. Every model will be shown and described in detail in the following chapter. The first model explained in this segment will also serve as an example to demonstrate the layers of the modeling framework. This model is an XML schema poisoning [32] attack. This model will be explained layer by layer and can be seen in Figure 1.

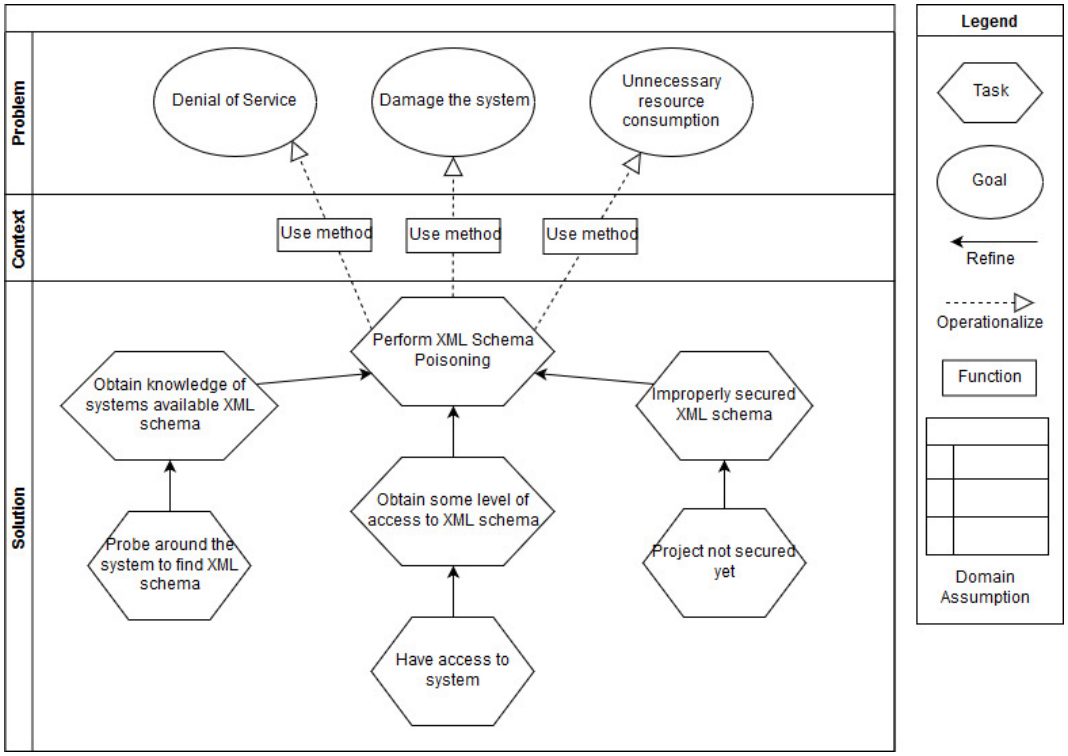

Figure 1: Attack pattern model for XML schema poisoning [32]

The main goal of this attack is to corrupt or modify the content of XML schema information passed between a client and server to undermine the security of the targeted system. XML schemas are essential since they provide the structure and content rules for XML documents. XML Schema poisoning [32] is an attack that aims to manipulate a schema either by replacing or modifying it to compromise the programs that process documents that use this schema.

In the first layer of the modeling framework, we describe the possible anti-goals of the attacker. The reasonable goals for this type of attack are Denial of Service, general damage to the system, and causing unnecessary recourse consumption. Damage to the system is a blanket term for possible damages that were not foreseen by the CAPEC attack pattern repository. These three anti-goals are modeled in the problem layer. They can be achieved by modifying the schema so that it does not contain the required information, which will then cause the XML document to be rejected by the XML schema.

The second layer in our framework is named the context layer. This layer consists of all the methods that can be used by an attacker in order to achieve his anti-goals. A method can also lead to more than one anti-goal, but for our purposes, it is sufficient to model a function. Going into details for each function in each attack pattern is outside of the scope of this thesis.

The third and final layer of our framework is named the solution layer. This layer is the most interesting to us since it describes the steps an attacker needs to take for him to achieve the stated anti-goals. These steps are called tasks in the model. Every solution layer should have at least one task called a general task. This general task is the task of performing the said attack, such as "Perform XML Schema Poisoning" in our example, and can be seen in every attack pattern in the following chapters. Furthermore, to be able to complete this general task, more prerequisite supporting

tasks need to be completed first. In our example, these are three supporting tasks. To complete these three tasks, further supporting tasks need to be completed. This creates a hierarchy of tasks that resembles a tree structure.

For an example of what damage this specific attack pattern in Figure 1 can cause, the original schema may require a @name attribute in all submitted documents. If the attacker removes this attribute from the schema, then documents created using the new modified schema could lack this attribute field. This small and simple change could cause the processing application to enter an unexpected state and stop working since there is data missing in the submitted XML. Another form of manipulation is with the data types described in the schema. If the data type were changed from float to string, calculations might fail to execute correctly, leading to errors in the system. An attacker may also change the encoding of individual fields, defined in the schema, allowing the contents to bypass filters used to filter out dangerous strings. For example, an attacker might change the schema encoding from American Standard Code for Information Interchange (ASCII) to Uniform Resource Locator (URL), which would result in a filter meant to catch a semicolon (;) failing to detect its URL encoding (%3B) [31]. The likelihood of this kind of attack is estimated to be low by CAPEC, but the severity of the attack is very high. Due to the minute changes required for this attack, it is safe to assume it would be challenging to identify in an extensive system.

4 Modeling Attack Patterns

The model in Figure 1 depicts a proposed three-layer model of the XML schema poisoning [32] attack pattern. As explained in chapter 3, the goal of this attack is to undermine the security of the targeted system by corrupting of modifying the XML schema information passed between a client and server. XML schemas are essential since they provide the structure and content rules for XML documents. XML Schema poisoning [32] is an attack that aims to manipulate a schema either by replacing or modifying it to compromise the programs that process documents that use this schema. It is possible that the model could change in the future due to new requirements and information. Currently, this model is based on up-to-date information. The model in Figure 1 was modeled with the visual elements proposed by Li et al. [12] and with the help of information from the CAPEC repository [31].

4.1 Modeling the Problem

The problem segment of the model in figure 1 is the malicious anti-goal that an attacker wants to achieve. For our model, this is modeled as a goal. With that goal in mind, we depict the motivation and consequences that the attacker wants to achieve. We focus on the threat and target that are summarized in the attack pattern repository, and which will be used in our prototype for automatic pattern matching. In our model, we have three problems that the attacker wants to achieve. Denial of Service is a well-known goal, the aim of which is to prevent the system from responding to legitimate requests. Damage to the system is a more general blanket term in use since it is hard to foresee all the damage this attack pattern could do. The last goal is to create highly resource-intensive changes that would cause the target damage over time due to higher operating costs and lost processing times.

4.2 Modeling the Context

The context of an attack pattern specifies under which situation the attack can be used to achieve an attacker's anti-goal. As can be seen in the context section of the model in Figure 1, our attacker uses functions to edit XML schemas he has access to, intending to make them unusable. We model such context and associate it with the operationalization link between Goals and tasks. We obtain the context information of an attack pattern by looking up the attack pattern attributes, attack prerequisites, and technical context from the CAPEC repository [31]. If some of the data is missing, it might be defined by some other reference. Thus we search the available references for the needed data and incorporate it into our model. As the context information is specified in natural language, analysts have to check such context during the application of attack patterns manually. This thesis aims to fill in any missing data needed to complete the eleven attack patterns from the repository and model them accordingly.

4.3 Modeling the Solution

The solution of attack patterns are specific attack actions that are performed by attackers using concrete attack techniques. We obtain the needed information from the CAPEC attack pattern repository [31] and model each attack action as a task in the goal model. The focus is placed on modeling every action needed to execute an attack successfully. When modeling the solution of the pattern, we first create a main general task to summarize the overall attack that has the attacker's anti-goals in mind. In our example, this is "Perform XML Schema Poisoning" as is shown in Figure 1 Next is to add tasks to the model that help or support the main general task. In our example, those are "Obtain knowledge of systems available XML schema", "Obtain some level of access to XML schema" and finally "Improperly secured XML schema" [32]. These tasks are required in order for the main general tasks to be accomplished, since the attacker needs to know what schema exist in the system, the

attacker has to test if the schema is not secured to modification and to be able to change them the attacker needs access to the schema, so he can execute the required methods and functions. These tasks are then supported by other tasks that enable the general execution. To be able to access the XML schema, the attacker has to have access to the system in general. An XML schema might not be adequately secured because the project which uses that XML schema might be ongoing, and thus not adequately secured. It is also possible that an XML schema was not secured when it was done because of a lack of attention from the creator's part. Not all tasks must be executed for the connected task to work. In some other models, such as the "XML Routing Detour Attacks" [33] depicted in Figure 5, some tasks have two or more connections to the parent task. In these cases, this can be understood as a logical "or" relationship, so either of the "Explore" child tasks could cause the parent task to be executable. These cases will be described in detail for each model separately.

4.4 Detailed XML Model Descriptions

Every attack pattern that was created for this thesis is described in this chapter. The figures showing the models were created with the help of the web tool Draw.io [21]. The modeling process is based on the three-layer modeling framework by Li et al. [10, 11, 12, 13, 14]. Each of the models was created by hand based on the descriptions of attack patterns from the CAPEC repository [31]. Additionally, a Python prototype was created in a Jupyter Notebook container to create a forecasting function along with other functionality. Jupyter Notebook is a web browser-based framework that enables the user to have a mix of Python code blocks as well as standard markdown text for easy reading. The information gained from the models is stored in an XML database file. This database is needed for the prototype to function correctly. In addition to this, a RelaxNG validation schema was created to validate the database file if new attack patterns need to be added to it. A template XML file containing all the necessary elements of a well-formed database was created to help users add

new attack patterns. Figure 2 shows this template.

```
<attack_pattern>
  <name></name>
  <likelihood></likelihood>
  <severity></severity>
  <reference></reference>
  <problem>
    <goal></goal>
  </problem>
  <context>
    <function></function>
  </context>
  <solution>
    <task></task>
  </solution>
</attack_pattern>
```

Figure 2: Attack pattern XML database template

Figure 3 shows a snapshot of the XML database file. As stated before, this database is meant to be modular, so adding new attack patterns should be straightforward.

```
<?xml version="1.0" encoding="UTF-8"?>
<attack_pattern_library>
  <attack_pattern>
    <name>XML External Entities Blowup</name>
    <likelihood>Medium</likelihood>
    <severity>High</severity>
    <reference>CAPEC-221</reference>
    <problem>
      <goal>Consume a large amount of resources</goal>
      <goal>Force system to freeze or crash</goal>
      <goal>Execute arbitrary malicious code</goal>
    </problem>
    <context>
      <function>Use method one</function>
      <function>Use method two</function>
      <function>Use method three</function>
    </context>
    <solution>
      <task>Perform XML External Entities Blowup</task>
      <task>Take advantage of the entity replacement property of XML</task>
      <task>Have malicious URI for replacement</task>
      <task>Server that has an implementation that accepts entities containing URI values</task>
      <task>Have server</task>
      <task>Application or service must rely on web service protocols</task>
      <task>Able to manipulate the communications to the targeted application or service</task>
    </solution>
  </attack_pattern>
```

Figure 3: Attack pattern XML database

Figure 4 shows the RelaxNG validation schema that is needed to validate the accuracy of the XML database. Validation can be done online with the help of the web validation tool from Liquid Technologies [22].

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <grammar ns=""
3    xmlns="http://relaxng.org/ns/structure/1.0" datatypeLibrary="
     http://www.w3.org/2001/XMLSchema-datatypes">
4    <start>
5      <element name="attack_pattern_library">
6        <oneOrMore>
7          <element name="attack_pattern">
8            <element name="name">
9              <text/>
10           </element>
11           <element name="likelihood">
12             <text/>
13           </element>
14           <element name="severity">
15             <text/>
16           </element>
17           <element name="reference">
18             <text/>
19           </element>
```

Figure 4: Attack pattern XML database validation schema

In the following ten chapters, every model will be explained in detail. The explanation will contain a short description of the attack pattern, the model and a detailed explanation of the goals, and tasks of the attack pattern.

4.4.1 XML Routing Detour Attacks

A model of the "XML Routing Detour Attacks" attack pattern is depicted in Figure 5. The reference number for this attack pattern in the CAPEC repository is 219 [31, 33]. The likelihood of this attack is high, and severity is medium, so care is needed to ensure that the system is not vulnerable to it. The model shows that the attack pattern has three goals, which are: "Read and modify Data", "Gain Privileges" and "Bypass Protection Mechanism", and that the "Perform XML Routing Detour Attacks" task has to be completed in order for these goals to be reached.

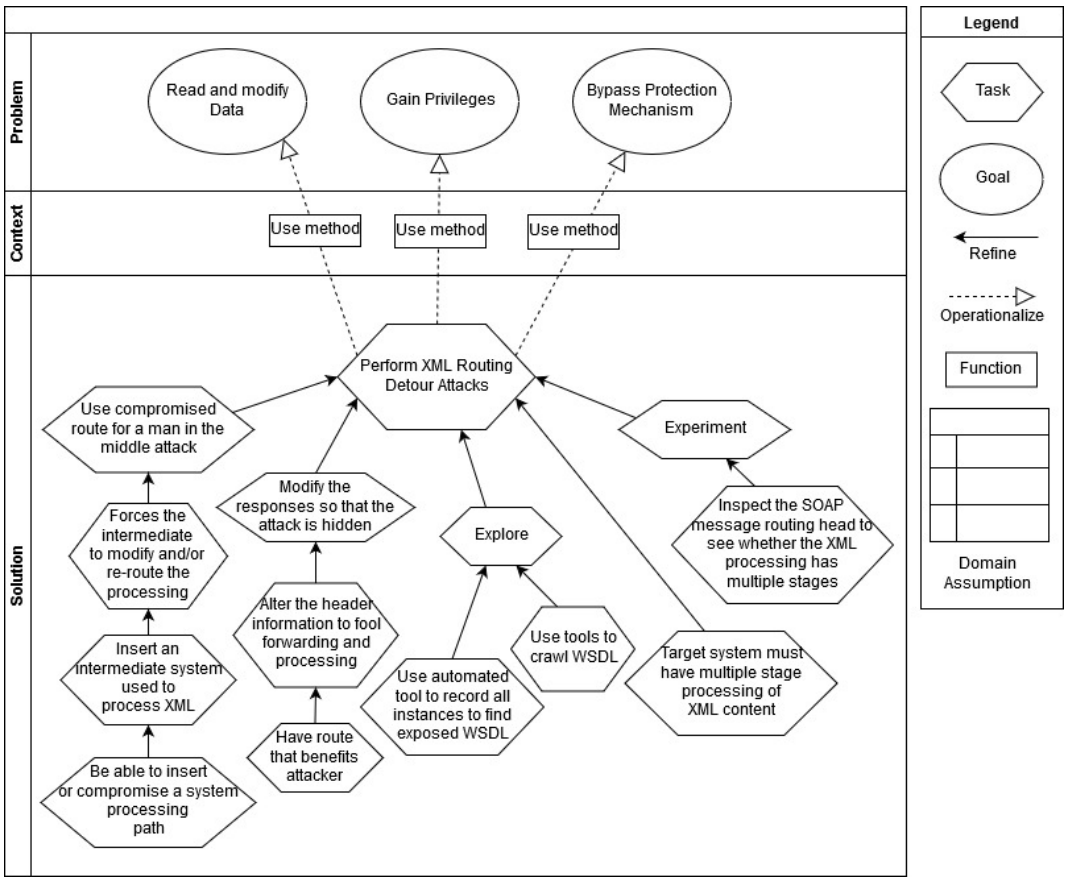

Figure 5: Attack pattern model for XML Routing Detour Attacks [33]

Before the general task can be completed, a series of five tasks need to be completed first. These five tasks are: "Use compromised route for a man in the middle attack", "Modify the responses so that the attack is hidden", "Explore", "Target system must have multiple stage processing of XML content" and "Experiment". For the first task, a series of tasks have to complete, starting at "Be able to insert or compromise a system processing path", "Insert an intermediate system used to process XML", and lastly, "Forces the intermediate to modify and/or re-route the processing". The second task also has a chain of tasks that need to be completed: "Have route that benefits attacker" and "Alter the header information to fool forwarding and processing". The "Explore" task is distinct since either of the child tasks can be completed for the parent task to be achievable. The fourth task has no children tasks, and the fifth task has the child task "Inspect the SOAP message routing head to see whether the XML processing has multiple stages". Going forward, each instance where a logical "or" relationship is present will be explained.

4.4.2 XML External Entities Blowup

The model in Figure 6 is called XML External Entities Blowup. The reference number for this attack pattern in the CAPEC repository is 221 [31, 34]. The likelihood of this attack is medium, and the severity is high. This attack can do significant damage, so care is needed to ensure that the system is not vulnerable to it. The model shows that the attack pattern has three goals: "Consume a large amount of resources", "Force system to freeze or crash", and "Execute arbitrary malicious code". For these goals to be reached, the general task "Perform XML External Entities Blowup" has to be completed. These goals are the reason that this attack can cause significant damage, especially the third goal.

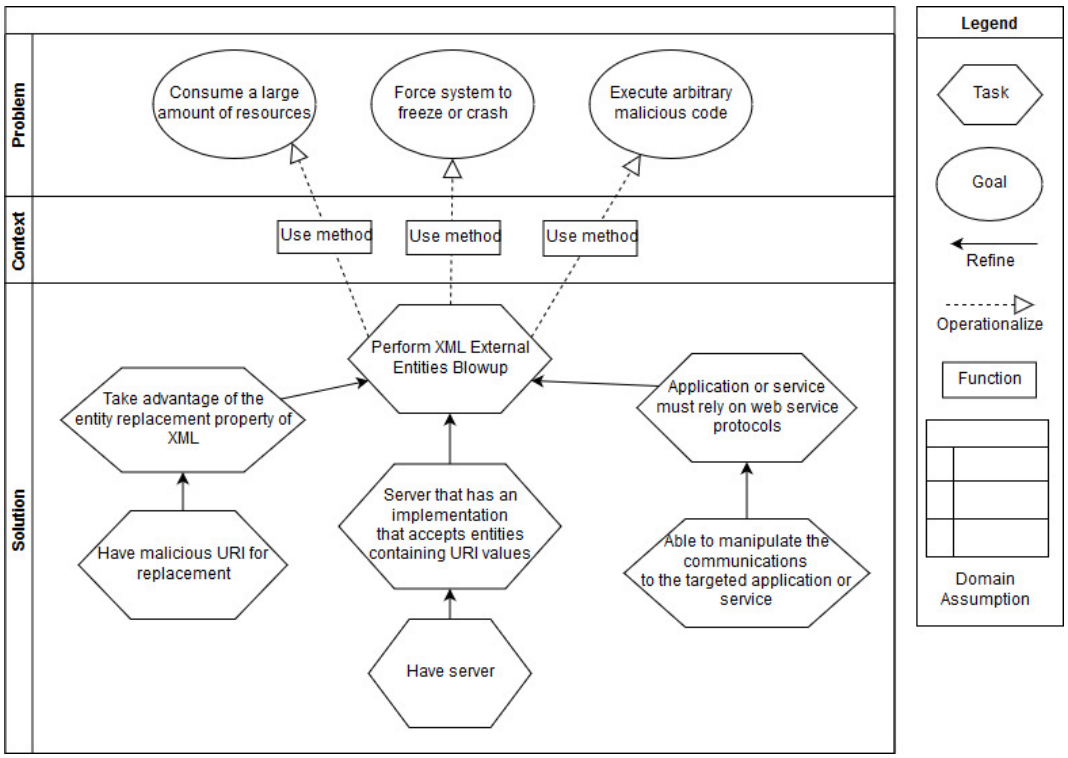

Figure 6: Attack pattern model for XML External Entities Blowup [34]

Before the general task can be completed, much like the model for XML Schema Poisoning, a series of three tasks need to be completed first. The first task, "Take advantage of the entity replacement property of XML", has the child task, "Have malicious URI for replacement", which has to be completed before the parent. The second task, "Server that has an implementation that accepts entities containing URI values", also has the child task, "Have server". Lastly, the third task, "Application or service must rely on web service protocols", has the child task "Able to manipulate the communications to the targeted application or service".

4.4.3 XML Entity Linking

The reference number for this attack pattern in the CAPEC repository is 201 [31, 35]. The likelihood of this attack is high, and the severity is high. This attack can do significant damage and is also very likely to occur, so additional care is needed to ensure that the system is not vulnerable to it. Figure 7 shows that the model has two goals "Reveal sensitive information" and "Allow the opening of arbitrary files or connections". For the attacker to achieve these goals the general task "Perform XML Entity Linking" has to be completed.

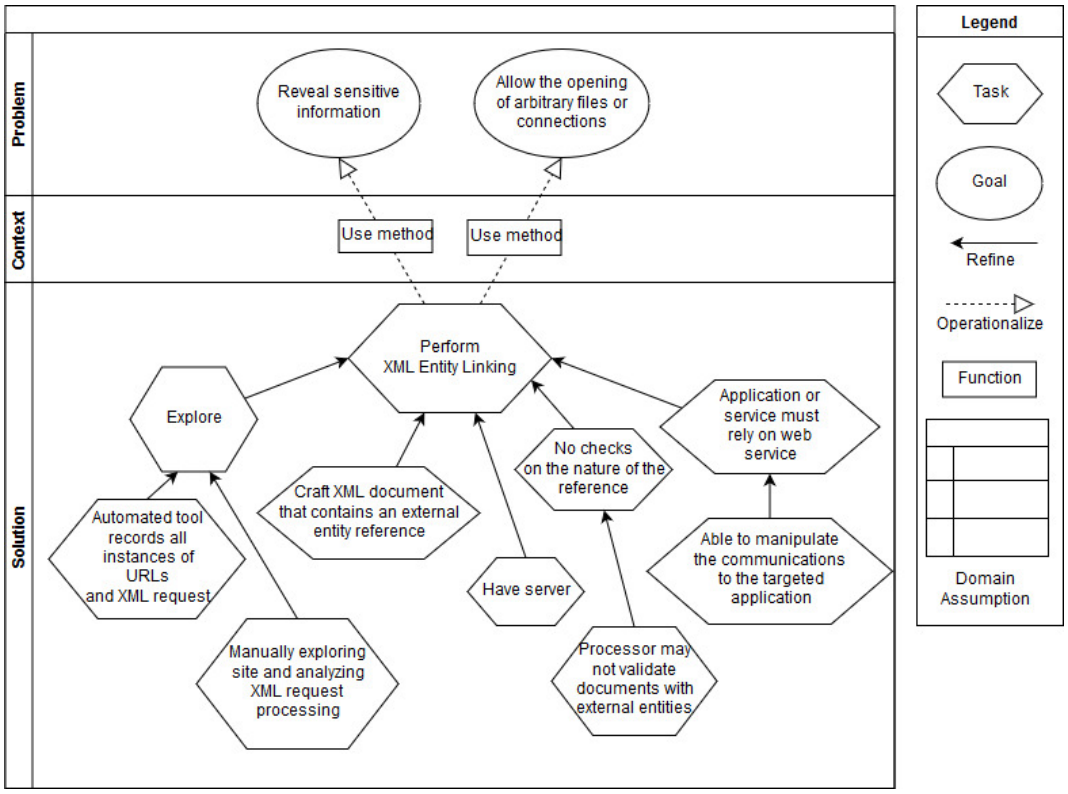

Figure 7: Attack pattern model for XML Entity Linking [35]

Before the general task can be completed, five tasks need to be completed. The first task is to "Explore". This is a task that has two child tasks either of which can be completed. The child tasks are "Automated tool records all instances of URLs and XML request processing" and "Manually exploring site and analyzing XML request processing". "Automated tool records all instances of URLs and XML request processing" is an automatic task meaning a tool does the work for the attacker with little input needed. "Manually exploring site and analyzing XML request processing" is a manual task meaning an attacker has to explore the target by hand. The second and third tasks for the general task are "Craft XML document that contains an external entity reference" and "Have server". The fourth task is "No checks on the nature of the reference". This task has the child task "Processor may not validate documents with external entities" that needs to complete before the parent can complete. The fifth task is "Application or service must rely on web service protocols" with the child task "Able to manipulate the communications to the targeted application or service" were analogous to the fourth task, the child task needs to complete first.

4.4.4 XQuery Injection

The model depicted in Figure 8 is the attack pattern known as "XQuery Injection". The reference number for this attack pattern in the CAPEC repository is 84 [31, 36]. This is the first type of injection-based attack pattern. The likelihood of this attack is high, and the severity is very high. The model shows that the attack pattern has three goals: "Read and modify Data", "Gain Privileges", and "Execute Unauthorized Commands". For these goals to be reached, the general task "Perform XQuery Injection" has to be completed.

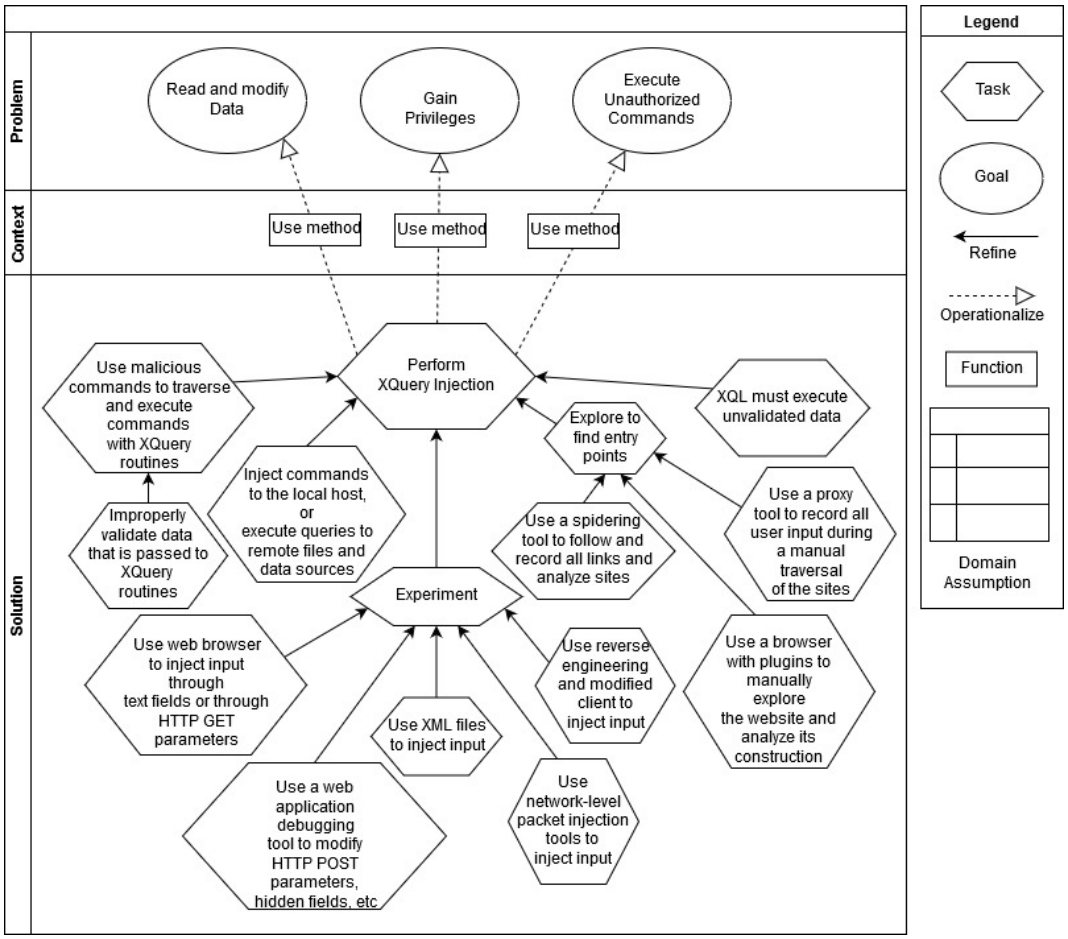

Figure 8: Attack pattern model for XQuery Injection [36]

A set of five tasks needs to be completed before the completion of the general task is possible. The first task is "Use malicious commands to traverse and execute commands with XQuery routines" with a child task "Improperly validate data that is passed to XQuery routines" that has to be completed first. The second task is "Inject commands to the local host, or execute queries to remote files and data sources". The third task is "Experiment". The third task has five child tasks in an "or" relationship, meaning at least one has to complete before the parent task can complete. These child tasks are: "Use web browser to inject input through text fields or through HTTP GET parameters", "Use a web application debugging tool to modify HTTP POST parameters, hidden fields, etc", "Use XML files to inject input", "Use network-level packet injection tools to inject input" and "Use reverse engineering and modified client to inject input". The fourth task is "Explore to find entry points", which also has three child task in an "or" relationship. These tasks are: "Use a spidering tool to follow and record all links and analyze sites", "Use a proxy tool to record all user input during a manual traversal of the sites" and "Use a browser with plugins to manually explore the website and analyze its construction". The final task needed in order to complete the general task is "XQL must execute unvalidated data".

4.4.5 XPath Injection

The model depicted in Figure 9 is a model of the "XPath Injection" attack pattern. The reference number for this attack pattern in the CAPEC repository is 83 [31, 37]. This is the second type of injection-based attack pattern. The likelihood of this attack is high, and the severity is high as well, so special care is needed to ensure that the system is not vulnerable to it. The model shows that the attack pattern has two goals, that being "Read Data" and "Gain Privileges", and for these goals to be reached, the general task "Perform XPath Injection" has to be completed.

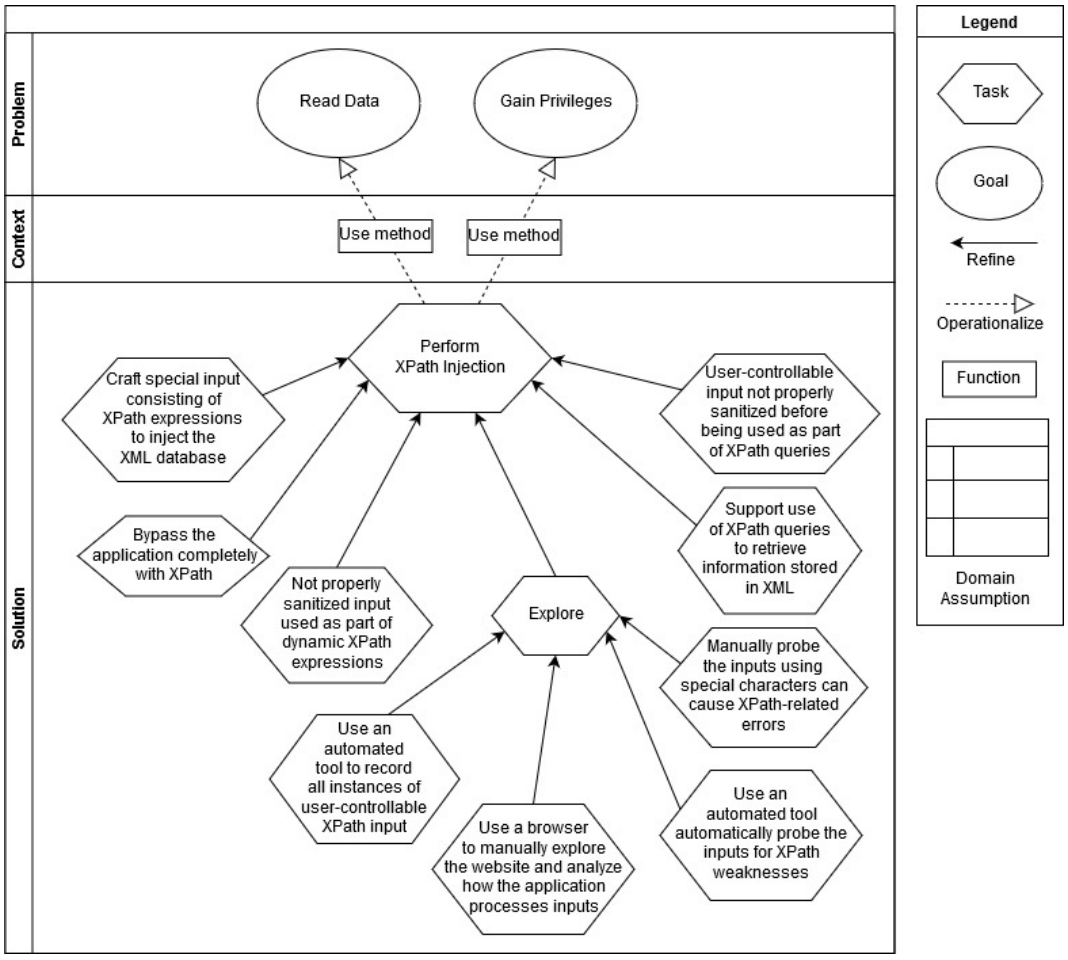

Figure 9: Attack pattern model for XPath Injection [37]

A set of six tasks needs to be completed before the completion of the general task is possible. The first task is "Craft special input consisting of XPath expressions to inject the XML database". The second is "Bypass the application completely with XPath". The third is "Not properly sanitized input used as part of dynamic XPath expressions". The fourth task is "Explore", and this task has four child tasks. At least one of them must be completed because these tasks have an "or" relationship. These tasks are: "Use an automated tool to record all instances of user-controllable XPath input", "Use a browser to manually explore the website and analyze how the application processes inputs", "Use an automated tool automatically probe the inputs for XPath weaknesses" and "Manually probe the inputs using special characters can cause XPath-related errors". The fifth task is "Support use of XPath queries to retrieve information stored in XML". The last task needed for the general task is "User-controllable input not properly sanitized before being used as part of XPath queries".

4.4.6 DTD Injection

The model depicted in Figure 10 is the attack pattern known as Document Type Definition (DTD) Injection. The reference number for this attack pattern in the CAPEC repository is 228 [31, 38]. This is the third type of injection-based attack pattern. The likelihood of this attack is high, and the severity is medium, so care is needed to ensure that the system is not vulnerable to it. The model shows that the attack pattern has two goals: "Negative technical impact" and "Consume a large amount of resources".

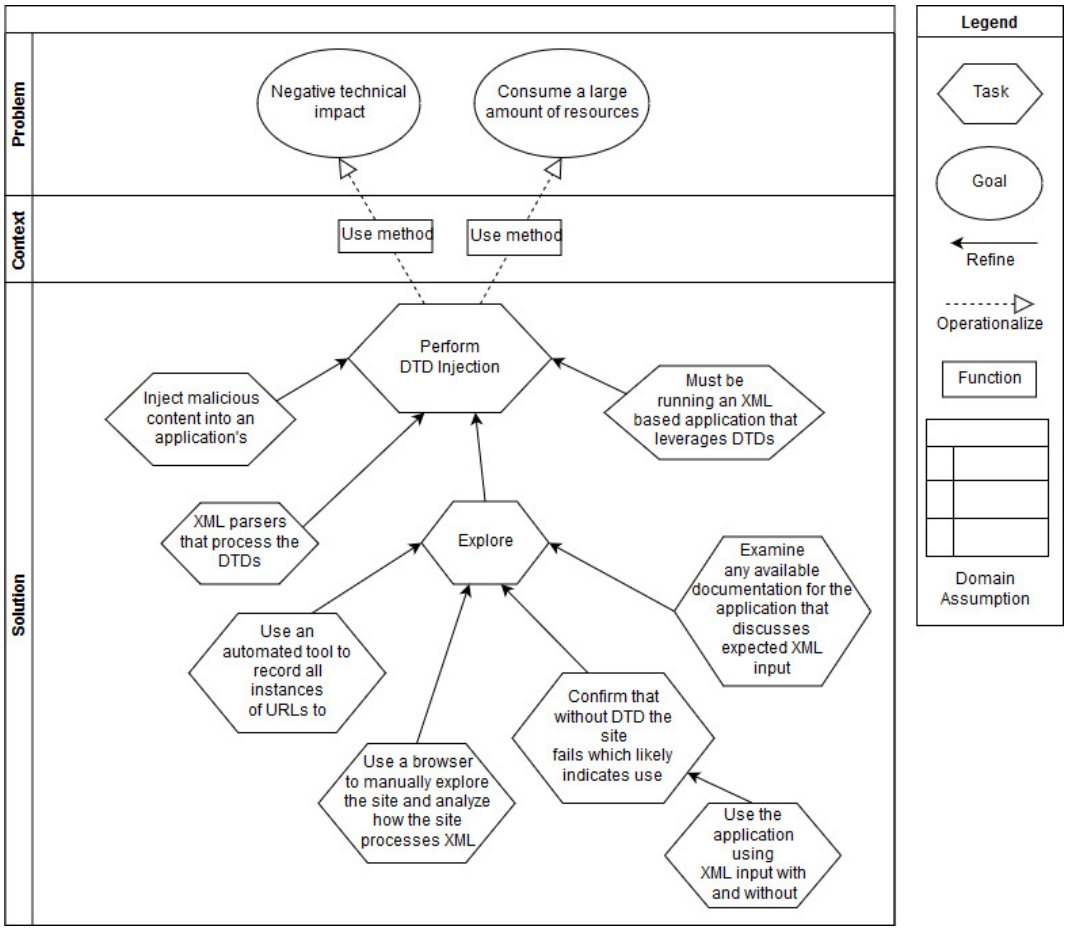

Figure 10: Attack pattern model for DTD Injection [38]

The general task can be seen in Figure 10 as the task "Perform DTD Injection". A set of four tasks needs to be completed before the completion of the general task is possible. The first task is "Inject malicious content into an application's DTD". The second is "XML parsers that process the DTDs". The third task is "Explore", which has four child tasks where either of them must be completed before the parent task can complete. These four child tasks are: "Use an automated tool to record all instances of URLs to process XML", "Use a browser to manually explore the site and analyze how the site processes XML", "Examine any available documentation for the application that discusses expected XML input" and "Confirm that without DTD the site fails which likely indicates use of DTD". The fourth child task has another child task that has to complete first called "Use the application using XML input with and without a DTD". The last task is called "Must be running an XML based application that leverages DTDs". Once these tasks are completed, the general task can proceed.

4.4.7 XML Attribute Blowup

This model is called XML Attribute Blowup. The reference number for this attack pattern in the CAPEC repository is 229 [31, 39]. This attack pattern has the likelihood of occurring high and the severity at high. Since the likelihood of this attack pattern is high and even more critical, the severity is also high, special attention must be given to secure the system against it. This is the only model among the eleven that has four possible goals. The possible goals are: "Consume a large amount of resources", "Read Data", "Execute Unauthorized Commands", and "Gain Privileges".

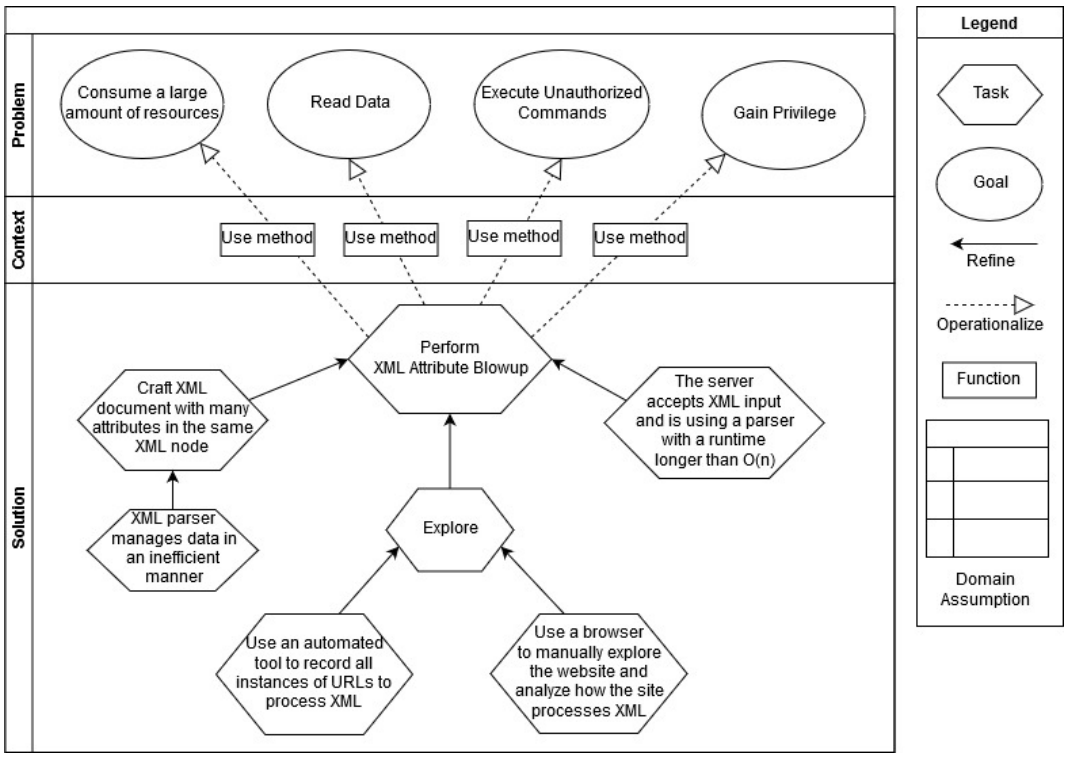

Figure 11: Attack pattern model for XML Attribute Blowup [39]

The model in Figure 11 depicts the attack pattern. The goals can be achieved by performing the general task "Perform XML Attribute Blowup". A set of three tasks needs to be completed before the completion of the general task is possible. The first is "Craft XML document with many attributes in the same XML node". This task also has a child task that must be completed first called "XML parser manages data in an inefficient manner". The second task is "Explore" and this task has two child tasks: "Use an automated tool to record all instances of URLs to process XML" and "Use a browser to manually explore the website and analyze how the site processes XML" where either of them can be completed. The last task is "The server accepts XML input and is using a parser with a runtime longer than O(n)". Each of the goals has its separate function or method which the attacker uses in the context layer.

4.4.8 XML Quadratic Expansion

This attack pattern is not very detailed as opposed to the other models. The reason for this is that this is a very new attack pattern based on the CAPEC reference number. The reference number for this attack pattern in the CAPEC repository is 491 [31, 40]. The CAPEC repository states that this attack pattern has a medium likelihood of occurring, and the severity is high. Based on the reference number, the chance that this model will change in the future is almost assured as new information about the attack pattern becomes known. Figure 12 shows the present state.

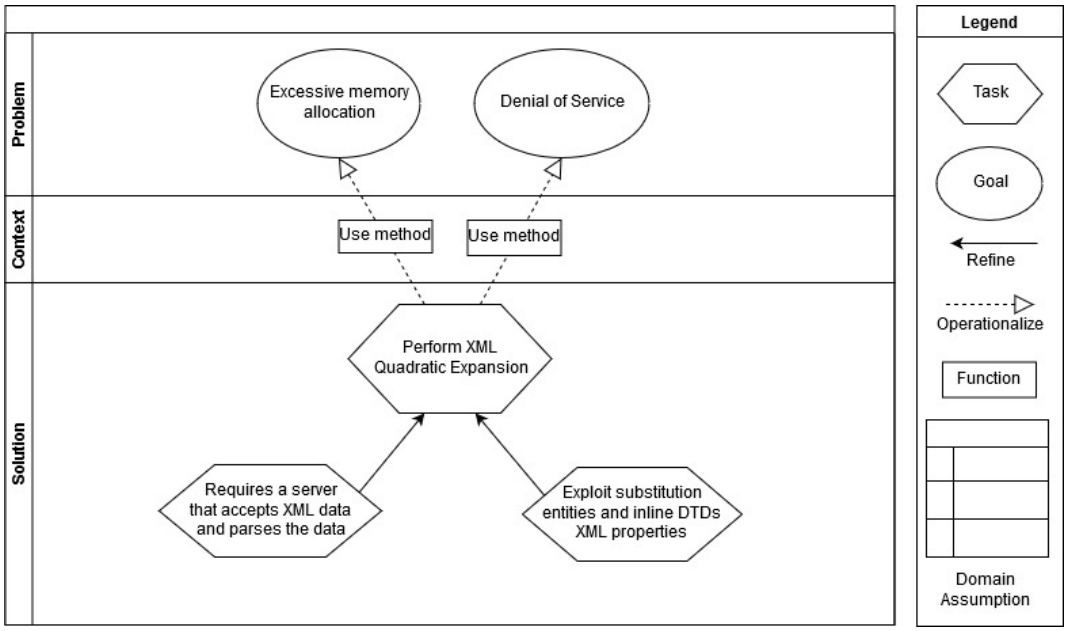

Figure 12: Attack pattern model for XML Quadratic Expansion [40]

Goals for this model are: "Denial of Service" and "Excessive memory allocation". The attacker achieves this trough the task "Perform XML Quadratic Expansion". Before this task can be completed, the tasks "Exploit substitution entities and inline DTDs XML properties" and "Requires a server that accepts XML data and parses the data" need to be completed.

4.4.9 XML Entity Expansion

The reference number for this attack pattern in the CAPEC repository is 197 [31, 41]. This attack pattern has the likelihood at high, and the severity is medium. The likelihood of this attack is high, so attention is needed in securing the system.

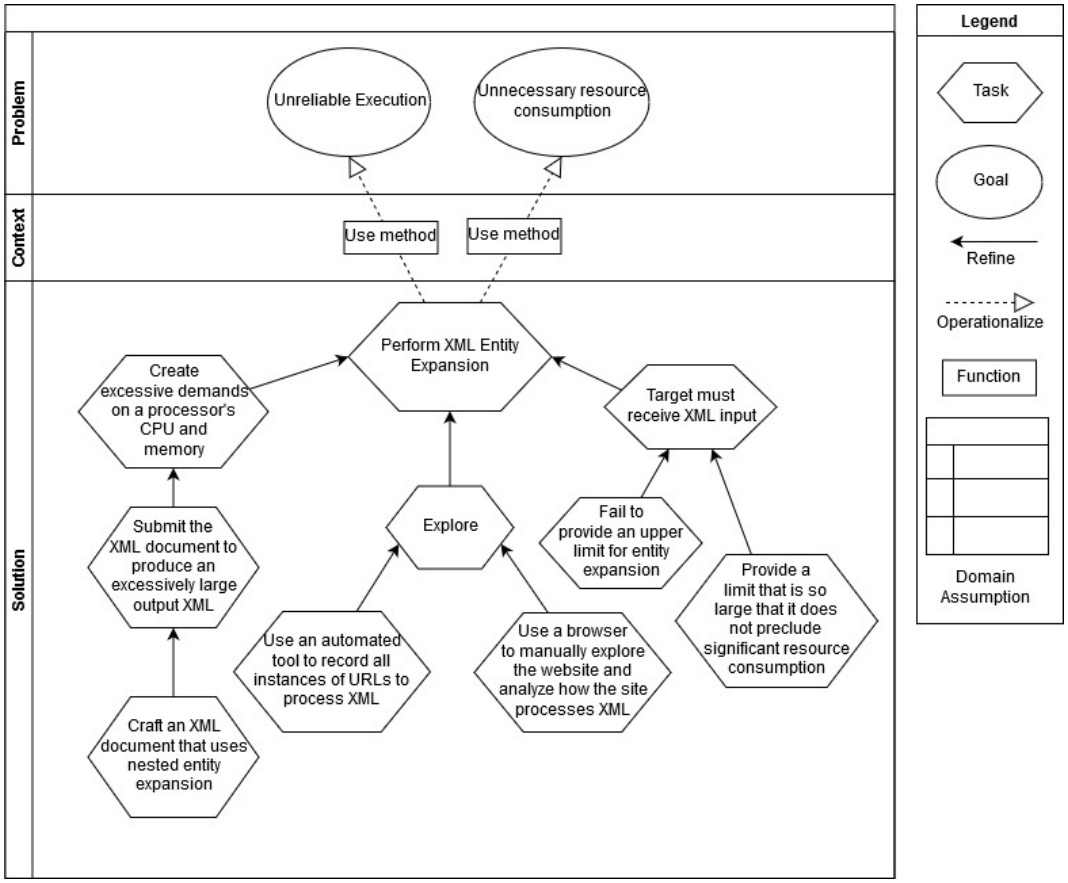

Figure 13: Attack pattern model for XML Entity Expansion [41]

This model features the task "Explore", as well as "Target must receive XML input". These tasks have child tasks that are in a logical "or" relationship therefore only one of the two has to be fulfilled in order for the parent task to be executable. The attack pattern is straight forward and has two goals: "Unreliable Execution" and "Unnecessary resource consumption".

To succeed in the attack and achieve the goals, the attacker must perform the task "Perform XML Entity Expansion". For this task to be executable, the child tasks "Create excessive demands on a processor's CPU and memory", "Explore", and "Target must receive XML input" must first be completed. For "Create excessive demands on a processor's CPU and memory" to complete, the attacker first has to complete "Craft an XML document that uses nested entity expansion", which has the parent task "Submit the XML document to produce an excessively large output XML". The task "Explore" has the child tasks "Use an automated tool to record all instances of URLs to process XML" and "Use a browser to manually explore the website and analyze how the site processes XML" where either of them can be completed. The task "Target must receive XML input" also has two child tasks, "Fail to provide an upper limit for entity expansion" and "Provide a limit that is so large that it does not preclude significant resource consumption", where either of them can also be completed for the parent task to complete.

4.4.10 XML Ping of the Death

The reference number for this attack pattern in the CAPEC repository is 147 [31, 42]. This attack pattern has the likelihood of low and the severity at medium. The model itself is similar to other models since it features the task "Explore". As before, this task has child tasks that are in a logical "or" relationship. Either of the two has to be completed in order for the parent task to be executable. Aside from this, the model is straight forward and has two goals: "Denial of Service" and "Unnecessary resource consumption".

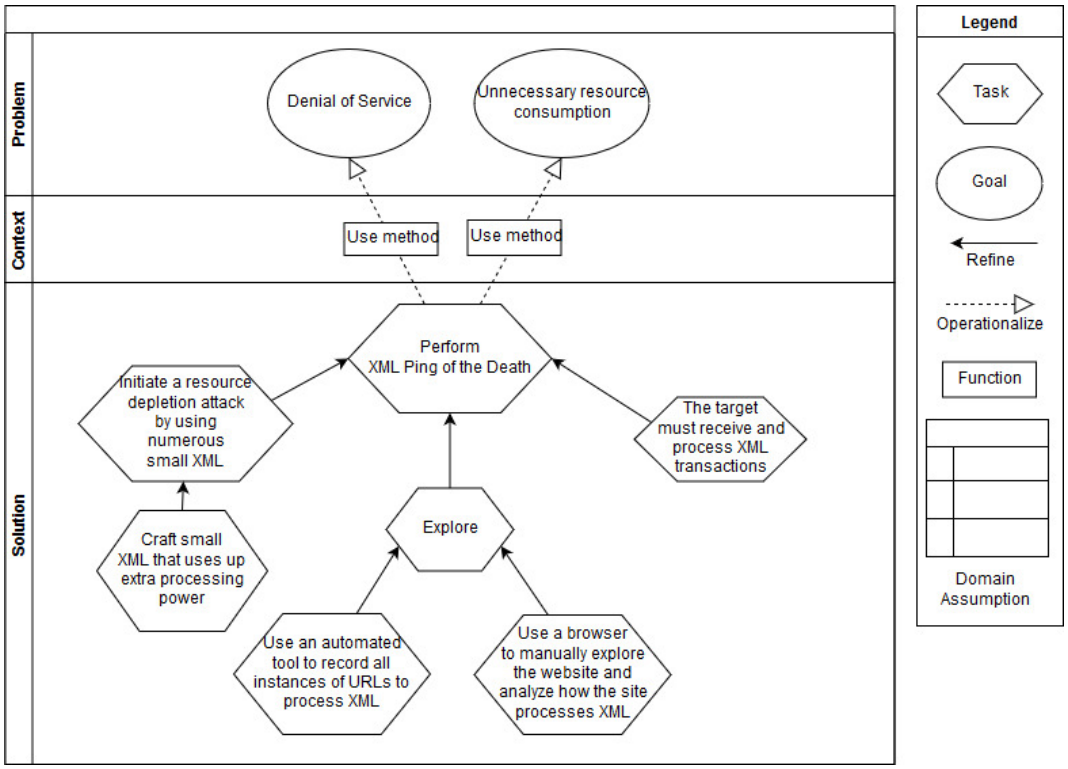

Figure 14: Attack pattern model for XML Ping of the Death [42]

To achieve these goals, the attacker has to perform the general task "Perform XML Ping of the Death". Before this, the attacker has to perform the child tasks: "Initiate a resource depletion attack by using numerous small XML messages", "Explore", and "The target must receive and process XML transactions". To perform the task "Initiate a resource depletion attack by using numerous small XML messages", the attacker first has to have completed the child task "Craft small XML that uses up extra processing power" and verified that the crafted XML does use up extra CPU time. As explained in other models, the task "Explore" has the child tasks "Use an automated tool to record all instances of URLs to process XML" and "Use a browser to manually explore the website and analyze how the site processes XML". Either of these can be completed, and they can be run in tandem as one is a manual task, and the other is an automated one.

4.5 Model Extension

The above attack patterns are specific attacks that are connected to XML attacks. We obtained the needed information from the CAPEC attack pattern repository [31] and created the models described above using that information. Other attack patterns can also be modeled and included in the database used by our prototype application in the same fashion as the eleven we modeled. In order to extend the functionality of the prototype, a model for each attack pattern that is to be added needs to be created so that the tasks, goals, and methods are easily added to the XML database file. When modeling the solution of the attack pattern, we first need to create a main general task to summarize the overall attack that has the attacker's anti-goals in mind. Next, we need to add tasks to the model that help enable or support the main general task. These tasks are required to complete first in order for the main general tasks to be accomplished. These supporting tasks are then supported by other additional tasks that enable the task execution, creating a hierarchy of tasks. As explained in chapter 4, not all tasks must be executed for the connected task to work. In some models such as the "XML Routing Detour Attacks" [33] depict above, some tasks have two or more connections to the parent task. In these cases, this can be understood as a logical "or" relationship, so either of the "Explore" child tasks could cause the parent task to be executable.

In case a model needs to be changed, e.g., to reflect an update in the CAPEC repository [31], the changes should also be made in the XML database. Once the changes or additions have been made, it is advised that the XML database file be validated with the included RelaxNG XML schema to check if it is well-formed and correct.

5 Prototype Implementation and Results

This chapter of the thesis is devoted to the prototype. Here we will showcase relevant excerpts of code, explain the intended usage of the prototype application, and provide our reasoning for the usage of certain functions during development. We will also discuss the problems we encountered during development. Parts of the code will also contain comments explaining their function. Everything written here can also be found in the Jupyter Notebook document that contains the prototype for this thesis. This prototype was created to prove that a forecasting system based on data from the CAPEC repository can be built. The modeling method used for modeling the attacks was devised by Li et al. [10, 11, 12, 13, 14] and is called the three-layer modeling framework. This framework was also used to model the eleven models described in chapter 4. The information gained from those models was then stored in a local database file for attack patterns and is used in this prototype. The information needed for each model is as follows: the name of the attack pattern, the likelihood of the attack happening, the severity of the damage done, the CAPEC or another source, the problem (including the attacker's goals), the context (including the attacker's functions), and lastly the solution (including the attacker's tasks needed for a successful attack). The problem, context, and solution each must at least have one of goal, function, and task element respectively, but usually, have more than one. The Jupyter Notebook document summary is as follows:

1. Variable Initialization and Environment Setup
2. Core Functions
3. Prototype Main Body

5.1 Variable Initialization and Environment Setup

Below are the libraries and the variables needed for the normal functioning of the prototype. Before running this code block, the user has to make sure that an XML file called "AP_library" is located in the same folder as this Jupyter Notebook document. Along with this file, there should be a sub-folder called "img" where every model image will be placed upon creation.

```
# Library imports.
# Library for navigating and parsing XML-s.
import xml.etree.ElementTree as ET

# Library for opening images with OS image viewer.
from PIL import Image

# Library for clearing output window for style.
from IPython.display import clear_output

# Library for counting different things at the same time.
from collections import Counter

# Reading the XML file.
tree = ET.parse('AP_library.xml')
xml_data = tree.getroot()

# List of attack pattern names for visibility and main menu use.
attack_pattern_list = []
for elem in xml_data.findall('.//attack_pattern/name'):
    attack_pattern_list.append(elem.text)
```

Figure 15: Jupyter Notebook variables and environment

Assuming the database and img sub-folder are present, running the next block of code should set up the environment needed to run the other code blocks.

5.2 Core Functions

This chapter features the core functions that make up the prototype. Each code block features a small description of the implemented function, as well as comments within the code itself.

The code block shown in Figure 16 features the function for finding similarly named attack patterns. This is a quality of life function that is used in the "Relevant information for an attack pattern" and "Display image of a model for an attack pattern" functions so that the user does not need to write the exact full name of attack pattern. In the case of more than one attack pattern match to the given input, the attack patterns are listed, and the user is asked for additional input. The user can also choose an attack pattern that isn't in the provided list. In case no attack pattern is found, the function returns False.

```
def find(false_attack_pattern):
    while(True):

    # Need to lower all input text for easy input handling.
        if type(false_attack_pattern) is str:
            false_attack_pattern = false_attack_pattern.lower()
        else:
            print('You have to enter an attack pattern name!')
            break

    # Create a list of attack patterns if there are more than one found in case of similar names.
        true_attack_pattern = []
        for i in range(len(attack_pattern_list)):
            if false_attack_pattern in attack_pattern_list[i].lower():
                true_attack_pattern.append(attack_pattern_list[i])

    # If only one is found then assume that that is the one the user is looking for.
        if (len(true_attack_pattern) == 1):
            return true_attack_pattern[0]

    # For any other number return false and an error message.
        elif (len(true_attack_pattern) == 0):
            return False
        else:
            print('\nThere are more than one similarly named attack patterns!')
            print('These attack patterns are:\n')
            for i in range(len(true_attack_pattern)):
                print(true_attack_pattern[i])
            return False
```

Figure 16: Jupyter Notebook find function

The code block shown in Figure 17 features the function for the forecasting of attack patterns. This function generates a dictionary consisting of key-value pairs where the key is the name of the matched attack pattern, and the value is the number occurrences of that attack pattern in the detection list. This dictionary is then displayed to the user.

```
def forecast():

# List for adding tasks that will be used in the prediction of the attack patterns.
    task_list = []
    xpath = []
    add_tasks = True

    while(True):
    # List for predicted attack patterns.
        forecasted_attack_patterns = []
        print('Please add all of the tasks you know off.')

    # Loop for adding the tasks.
        while(True):
            task = input('\nPlease write the task: ')
            if task not in task_list:
                task_list.append(task)
            else:
                print('Task already in the list!')
            print('\nMore tasks?')
            answer = input('y for yes / n for no: ')

        # Check list for answer just in case the user inputs 'yy' or 'yes' to prevent crash.
            if(answer[0] == 'n' or answer[0] == 'N'):
                add_tasks = False
                break
```

Figure 17: Jupyter Notebook forecast initialisation function

The forecasting is first done by XPath query matching answers to tasks in the XML database (Figure 18). Once a query is made for a task, all of the results are aggregated in a list. Once the list is completed, every attack pattern is counted and the attack pattern with the highest number of counts is the one the user is looking for, or rather the one that has the highest probability of having occurred.

```
    # Relevent information.
        for i in range(len(task_list)):
        # Creating Xpaths and finding all the possible results then adding them to a list for further use
            xpath += xml_data.findall('.//attack_pattern//solution[task="'+ task_list[i] +'"]/..//name')


        for elem in xpath:
        # Adding the names of possible attack patterns to a list from the above mentioned xml object list
            forecasted_attack_patterns.append(elem.text)
```

Figure 18: Jupyter Notebook forecast XPath function

A list is used because there is no native implementation of XPath that supports logical operators ”and” and ”or” and thus a workaround needed to be created.

```
        if (len(Counter(forecasted_attack_patterns)) == 1):
        # If only one attack pattern was found, then give that one as a return
            print('\nThe tool forecasts the following attack pattern:')
            return forecasted_attack_patterns[0]

        elif (len(Counter(forecasted_attack_patterns)) >= 2):
        # If more are found then show them
            clear_output()
            print('\nThere are more then one possible attack patterns.\n')
            print('\nThe possible attack patterns are: \n')
            print(Counter(forecasted_attack_patterns))

            # More refining of the list of possible attack patterns
            print('\nDo you wish to add more tasks to narrow it down more?')
            answer = input('y for yes / n for no: ')

            if(answer[0] == 'y' or answer[0] == 'Y'):
            # Goes back to the input of tasks above in the first while loop
                continue
            elif(answer[0] == 'n' or answer[0] == 'N'):
            # When refining done, return list with attack patterns that have occurred once or more
                return Counter(forecasted_attack_patterns).most_common()

        break
```

Figure 19: Jupyter Notebook forecast function

Figure 19 shows the rest of the function. There are two possible cases here. The first case is that there can only be one attack, so the function returns that single possible attack pattern. The second case is that there is more than one possible attack. A list of possible attacks is converted to a dictionary through the Counter function mentioned above. As mentioned before, this dictionary is a sorted array of key-value pairs where the key is the name of the possible attack pattern, and the value is the number of occurrences in the list. The function will ask the user if he wishes to add more tasks to try to refine the forecasting even more. In case the user wishes to refine the forecasting, the function starts from the beginning with the additional tasks. In case the user does not wish to add more tasks, the function displays the dictionary result.

There is also an automatic version of the forecasting function above. This function, just like the one above, generates a dictionary consisting of key-value pairs where the key is the name of the matched attack pattern, and the value is the number

occurrences of that attack pattern in the detection list. This dictionary is then displayed to the user. The forecasting is done by XPath query matching answers to tasks in the XML database. The tasks are gathered from a windows log file that is exported into an XML format. Once a query is made for a task, all of the results are aggregated in a list. Once the list is completed, every attack pattern is counted, and the attack pattern with the highest number of counts is the one the user is looking for, or rather the one that has the highest probability of having occurred. A list is used because natively, there is no implementation support of XPath that supports logical operators ”and” and ”or” and thus a workaround needed to be created.

The code block shown in Figure 20 features the function that retrieves relevant information needed for a specific attack pattern. This function is meant for use after the above forecasting system predicts which attack pattern is probably the attacker's aim. In order for this function to work, a detailed description of the attack pattern needs to be present in the XML database.

```
def open_info(attack_pattern):
# Infinite loop to let the user decide when done with function.
    while(True):
        attack_pattern_found = find(attack_pattern)
        if attack_pattern_found in attack_pattern_list:
            try:
        # Relevent information.
                clear_output()
                print('\n{} Reference: '.format(attack_pattern_found))
                for elem in xml_data.findall('.//attack_pattern[name="'+ attack_pattern_found +'"]//reference'):
                    print(elem.text)
                print('\n{} Likelihood: '.format(attack_pattern_found))
                for elem in xml_data.findall('.//attack_pattern[name="'+ attack_pattern_found +'"]//likelihood'):
                    print(elem.text)
                print('\n{} Severity: '.format(attack_pattern_found))
                for elem in xml_data.findall('.//attack_pattern[name="'+ attack_pattern_found +'"]//severity'):
                    print(elem.text)
                print('\n{} Goals: '.format(attack_pattern_found))
                for elem in xml_data.findall('.//attack_pattern[name="'+ attack_pattern_found +'"]//goal'):
                    print(elem.text)
                print('\n{} Funktions: '.format(attack_pattern_found))
                for elem in xml_data.findall('.//attack_pattern[name="'+ attack_pattern_found +'"]//function'):
                    print(elem.text)
                print('\n{} Tasks: '.format(attack_pattern_found))
                for elem in xml_data.findall('.//attack_pattern[name="'+ attack_pattern_found +'"]//task'):
                    print(elem.text)
                break
```

Figure 20: Jupyter Notebook function for displaying the relevant information

This function also includes error handling, as shown in Figure 21 This is done so that in case the database is not included, the Jupyter Notebook container does not crashes.

```
            except:
        # Error handling.
                clear_output()
                print('There is no attack pattern with this name or there was an error!')
                print('Would you like to try again for another attack pattern?')
                answer = input('y for yes / n for no: ')
```

Figure 21: Jupyter Notebook function for relevant information error handling

The code block shown in Figure 22 features the function for getting the relevant image of the model for a specific attack pattern. The images are located in a sub-folder that must be at the same location as this Jupyter Notebook container. This function is used as a quality of life feature because the XML database does not include the hierarchy of the exact attack pattern procedure.

```
def open_image(img_name):
# Infinite loop to let the user decide when done with function.
    while (True):
        img_name_found = find(img_name)
        try:
    # From Image library imported function for opening images with OS image viewer.
    # All images are in the img/ subfolder for easy sorting and hierarchy.
            image = Image.open('img/' + img_name_found +'.jpg')
            image.show()
            break

        except:
    # Error handling
            print('\nThere is no attack pattern, or there are many with this name, or there was an error!')
            print('Would you like to try again for another attack pattern?')
            answer = input('\ny for yes / n for no: ')

    # Check list for answer just in case the user inputs 'yy' or 'yes' to prevent crash.
        if(answer[0] == 'n' or answer[0] == 'N'):
            break

        elif(answer[0] == 'y' or answer[0] == 'Y'):
            img_name = input('Please enter the attack pattern name again: ')
```

Figure 22: Jupyter Notebook function for displaying model image

This function also includes error handling, as shown in Figure 21. This function opens the operating systems default image viewer and displays the selected model.

As stated in this chapter, functions for displaying model images and displaying relevant information both have the function "find" integrated as a small quality of life improvement.

5.3 Prototype Main Body

The Code block in Figure 23 depicts the main body of the prototype.

```
while (True):
    try:

        clear_output()
        print('\nWelcome to the prototype!')
        print('\nThis program was written as a master thesis prototype for attack pattern forecasting.')

    # Main menu
        print('\n')
        print('\nOptions implemented are as follows: ')
        print('\n1. Attack pattern forecasting')
        print('\n2. Relevant information for an attack pattern')
        print('\n3. Display image of a model for an attack pattern')
        print('\n4. Log file automatic attack pattern forecasting')
        print('\n5. Exit\n')
        print('\n')

        option = input('Please select an option from the main menu: ')
```

Figure 23: Jupyter Notebook prototype core functions

Once run, the main menu of the prototype will greet the user. It will ask for an input on what options the user would like to use. Once an option is selected, it will describe what it does in detail, as well as what is expected from the user. After the main body is exited, the author, date, and mentor are displayed.

5.4 Prototype Results

After a user has selected a function from the main menu of the prototype, a detailed explanation of that function will be given. The functions require input, and after the user gives said input, the prototype will run the function. In our case, we have chosen the function for automatic attack pattern forecasting from a Windows Log file. After we provide the prototype with the required log file name, we get a dictionary as an output. The dictionary consists of a set of possible attack patterns that may have

occurred and are depicted in Figure 24. The higher the number in the dictionary key-value pair, the higher the likelihood that that specific attack pattern occurred, so for our example, it would most likely be "XML Routing Detour Attacks". After every function, the prototype gives us the option of going back to the main menu. In our example, we would like to know more about said attack pattern, so once we know which attack pattern is responsible, we go back to the main menu and look for the function for relevant information.

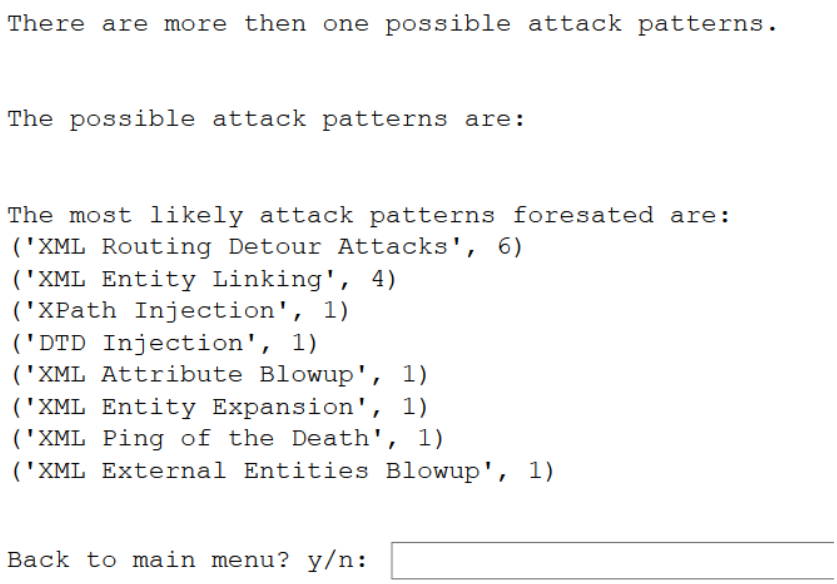

Figure 24: Jupyter Notebook forecasting function results

Back in the main menu, we go to the relevant information function, and as the input, we give the name of the found attack pattern. We do not need to write the full name of the attack pattern since there is a quality of life function to help us find the appropriate attack pattern with only part of the name. For the output, the function gives the included relevant information from the XML database file.

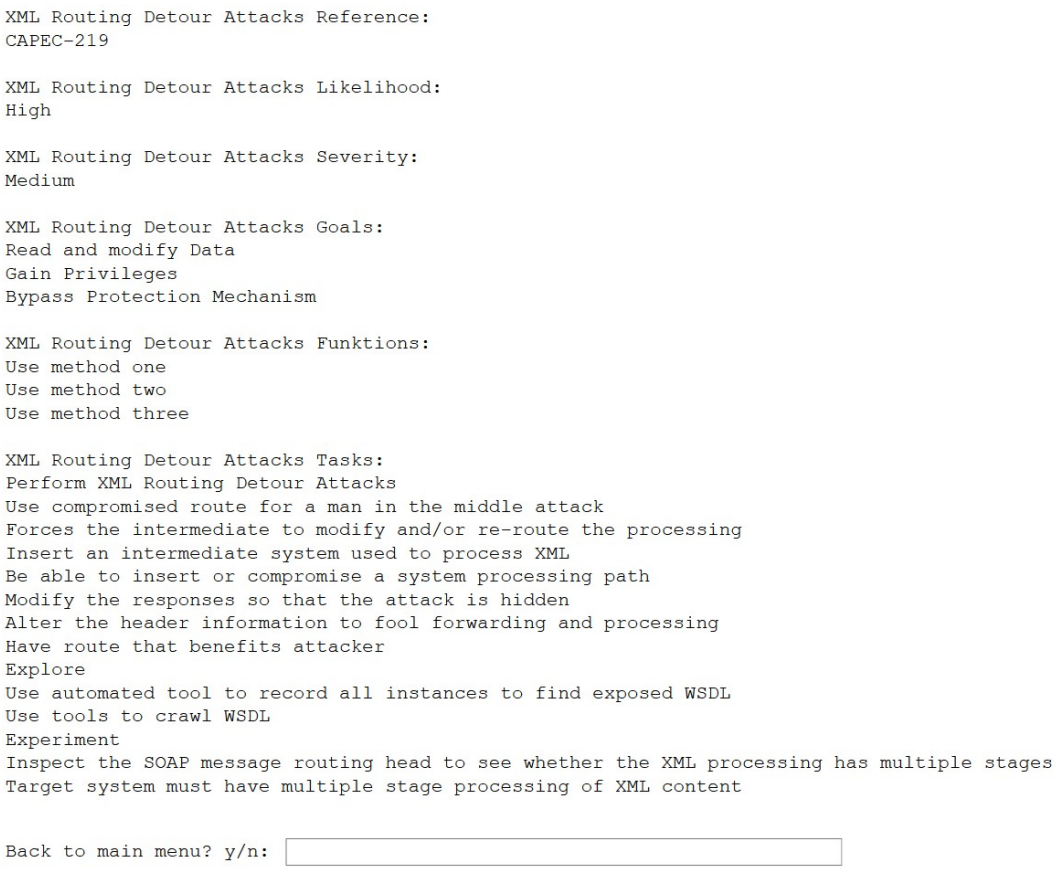

Figure 25: Jupyter Notebook function for relevant information results

In Figure 25 we can see the relevant information about the "XML Routing Detour Attacks" attack pattern: a reference number to the CAPEC repository [31], the likelihood of the attack, the severity of the damage the attack could cause, the goals, Functions and lastly the tasks that the attack pattern model.

The last option we have is to display an image of the model through the appropriate function, and this would display the model seen in Figure 5. These results match the results we expected based on our modifications to the test.xml log file.

6 Conclusion

The end goal of this thesis was to develop a tool-supported framework for forecasting the development of intrusions based on existing libraries of attack patterns. The underlying assumption of this thesis is that in the early stages of an attack, attack pattern matching will lead to a more significant number of potential attacks and that this number of possible attacks will significantly decrease in advanced stages of an attack. This can easily be pictured as a pyramid where the base of the pyramid are early stages, and the higher we go, the more advanced the stages are. The framework will, therefore, be more efficient with each stage of an attack. These stages can be seen as tasks in the appropriate images of the attack pattern models, so as more tasks are completed, the higher the likelihood that an attack pattern is forecast and others are ruled out. The framework consists of the core intrusion analysis model and the accompanying tool created in a Python Jupyter Notebook document. The modeling approach used in the thesis follows slightly adapted UML standards, just like the shown three-layer modeling framework by Li et al. [10, 11, 12, 13, 14]. All eleven models were developed as part of this thesis and integrated into the proof of concept prototype. The prototype has four core functionalities: a manual forecasting function based on tasks and automatic forecasting function based on XML windows logs, a function for displaying the image of an attack pattern model, and a function for showing the relevant information about an attack pattern. This thesis has proven that a prototype based on anti-goals can be created and could be used in the future to further understanding the attacker's point of view.

Additionally, the created prototype was made to be modular, so that extension of the framework and integration of all 516 attack patterns listed in the CAPEC repository is possible and feasible. If more attack patterns are added to the CAPEC repository in the future, they could also be added to our prototype. This would be a task for the future as each model has to be created, and the XML database file

has to be extended. We have provided the tool used in this thesis as well as the instructions on how to use it, which could help in future extensions. This thesis should help analyze and build more secure systems and make the use of XML in daily business more reliable and safe. The aim is to prevent even a slight amount of damage or reduce the resource cost and scope of a forensic analysis after an attack was carried out. This will lead to savings over a long period. As Sun Tzu [20] wrote, we have to know our enemy as well as ourselves so that we can win in every battle.

Glossary

anomaly detection A process aimed at discovering patterns in data sets that deviate from the general behavior or the expected behavior of the majority of the data. 22

anti-goal A goal from the perspective of an attacker. 9–13, 17, 26, 28, 29, 52, 63

attack pattern Is a type of pattern that is created from an attacker's viewpoint, i.e. what his goals are and how he gets to those goals. 2, 3, 6–13, 16, 20, 23–33, 35, 37, 39, 41, 43, 45–48, 50, 52, 53, 55, 57, 58, 60–63

attack vector A possible vulnerability that enables an attacker to compromise the security of a system. 2, 8, 12, 13, 20, 25

class diagram is the UML diagram that describes the structure of a system by showing the system's classes, their attributes, methods, and the relationships among objects. 10–12

dictionary A special Python datatype that consists of a key and value pair in each section. 55, 57, 58, 60

encoding The way in which symbols are mapped onto bytes, e.g. in the rendering of a particular font, or in the mapping from keyboard input into visual text. 27

float Data type for floating-point values, i. e. number values that have potential decimal places. 27

Jupyter Notebook A web browser based framework that enables the user to have a mix of Python code blocks as well as normal markdown text for easy reading. 2, 6, 8, 13, 30, 53–63

Python A programming language commonly used for data analysis. 2, 7, 8, 13, 25, 30, 63

regular expression A set of rules that define what the input should look like, regularly used in passwords. 16, 17

repository A large accumulation of data stored in a single place. 7, 8, 14, 20, 23, 26, 28–30, 33, 35, 37, 39, 41, 43, 45, 47, 48, 50, 52, 53, 62, 63

sequence diagram is the UML diagram that shows object interactions arranged in time sequence. 10–12

Socio-Technical System Is a system created from humanware, hardware, software and orgware of a company. 2

string Data type for a sequence of characters or symbols. 21, 27

three-layer modeling framework Is a framework that is composed by Li et al. and which models and analyzes requirements of STS at the business layer, software application layer, and physical infrastructure layer, respectively. 2, 3, 30, 53, 63

XML schema A set of rules that are used to validate that the input XML is well formed and fits those rules. 6, 18–20, 25, 26, 28–30, 52

XPath Is a query language that is used to retrieve information from an XML. 6, 41, 56, 58

Acronyms

ADIFA Attribute Density Function Approximation. 22

ASCII American Standard Code for Information Interchange. 27

ASP Answer Set Programming. 14

CAPEC Common Attack Pattern Enumeration and Classification. 14, 23, 25–30, 33, 35, 37, 39, 41, 43, 45, 47, 48, 50, 52, 53, 62, 63

CDATA Character Data. 21

CPU Central Processing Unit. 51

DDoS Distributed Denial of Service. 21

DOM Document Object Model. 21

DoS Denial of Service. 2, 11, 19, 21, 26, 28, 47, 50

DTD Document Type Definition. 43, 47

HTTPS Hypertext Transfer Protocol Secure. 20

MUSER Multi-layer Security Requirements analysis tool. 13

RNG RelaxNG. 18, 30, 32, 52

SQL Structured Query Language. 21

STS Socio-Technical System. 2, 8, 10

UML Unified Modeling Language. 8, 10, 11, 63

URL Uniform Resource Locator. 27

VoIP Voice over Internet Protocol. 4, 10, 11

W3C World Wide Web Consortium. 18

XML Extensible Markup Language. 1, 2, 6, 9, 13, 14, 18–22, 25–28, 30–33, 35, 37, 45, 47, 48, 50–52, 54, 58, 61, 63, 64

XSD XML Schema Definition. 18, 20–22

References

[1] Alexander, C. (1977). A pattern language: towns, buildings, construction. Oxford university press.

[2] Ali, R., Dalpiaz, F., & Giorgini, P. (2010). A goal-based framework for contextual requirements modeling and analysis. Requirements Engineering, 15(4), 439-458.

[3] Asnar, Y., Li, T., Massacci, F., & Paci, F. (2011, September). Computer aided threat identification. In 2011 IEEE 13th Conference on Commerce and Enterprise Computing (pp. 145-152). IEEE.

[4] Carlstedt, J., Bisbey, I. I., & Popek, G. (1975). Pattern-Directed Protection Valuation (No. ISI/RR-75-31). UNIVERSITY OF SOUTHERN CALIFORNIA MARINA DEL REY INFORMATION SCIENCES INST.

[5] Eastlake 3rd, D., Reagle, J., & Solo, D. (2001). XML-signature syntax and processing (No. RFC 3075).

[6] Fernandez, E., Pelaez, J., & Larrondo-Petrie, M. (2007, January). Attack patterns: A new forensic and design tool. In IFIP International Conference on Digital Forensics (pp. 345-357). Springer, New York, NY.

[7] Gegick, M., & Williams, L. (2005, May). Matching attack patterns to security vulnerabilities in software-intensive system designs. In ACM SIGSOFT Software Engineering Notes (Vol. 30, No. 4, pp. 1-7). ACM.

[8] Imamura, T., Dillaway, B., Simon, E., Kelvin, Y., Nyström, M., Eastlake, D., ... & Roessler, T. (2013). XML encryption syntax and processing version 1.1. W3C, Recommendation.

[9] Kent, K., Chevalier, S., Grance, T., & Dang, H. (2006). Guide to integrating forensic techniques into incident response. NIST Special Publication, 10(14), 800-86.

[10] Li, T., Paja, E., Mylopoulos, J., Horkoff, J., & Beckers, K. (2015, August). Holistic security requirements analysis: An attacker's perspective. In 2015 IEEE 23rd International Requirements Engineering Conference (RE) (pp. 282-283). IEEE.

[11] Li, T., Horkoff, J., Beckers, K., Paja, E., & Mylopoulos, J. (2015). A holistic approach to security attack modeling and analysis. In Proceedings of the Eighth International i* Workshop (pp. 49-54).

[12] Li, T., Paja, E., Mylopoulos, J., Horkoff, J., & Beckers, K. (2016, June). Security attack analysis using attack patterns. In 2016 IEEE Tenth International Conference on Research Challenges in Information Science (RCIS) (pp. 1-13). IEEE.

[13] Li, T., & Horkoff, J. (2014, June). Dealing with security requirements for socio-technical systems: A holistic approach. In International Conference on Advanced Information Systems Engineering (pp. 285-300). Springer, Cham.

[14] Li, T., Horkoff, J., Paja, E., Beckers, K., & Mylopoulos, J. (2015, November). Analyzing attack strategies through anti-goal refinement. In IFIP Working Conference on The Practice of Enterprise Modeling (pp. 75-90). Springer, Cham.

[15] Li, T., Horkoff, J., & Mylopoulos, J. (2014, November). Integrating security patterns with security requirements analysis using contextual goal models. In IFIP Working Conference on The Practice of Enterprise Modeling (pp. 208-223). Springer, Berlin, Heidelberg.

[16] Li, T., Horkoff, J., & Mylopoulos, J. (2014). A Prototype Tool for Modeling and Analyzing Security Requirements from A Holistic Viewpoint. In CAiSE (Forum/Doctoral Consortium) (pp. 185-192).

[17] McDermott, J. P. (2000, September). Attack net penetration testing. In NSPW (pp. 15-21).

[18] Menahem, E., Schclar, A., Rokach, L., & Elovici, Y. (2012). Securing your transactions: Detecting anomalous patterns in xml documents. arXiv preprint arXiv: 1209.1797.

[19] Saini, V., Duan, Q., & Paruchuri, V. (2008). Threat modeling using attack trees. Journal of Computing Sciences in Colleges, 23(4), 124-131.

[20] Tzu, S. (2008). The art of war. In Strategic Studies (pp. 63-91). Routledge.

Web References

[21] Draw.io, https://about.draw.io/about-us/, last available September 2019.

[22] Liquid Technologies (2019), Online relaxNG Validator, https://www.liquid-technologies.com/online-relaxng-validator, last available September 2019.

[23] Amit Sethi and Sean Barnum, Department of Homeland Security (2013). Introduction to Attack Patterns https://www.us-cert.gov/bsi/articles/knowledge/attack-patterns /introduction-to-attack-patterns, last available September 2019.

[24] Amit Sethi and Sean Barnum, Department of Homeland Security (2013). Attack Patterns, https://www.us-cert.gov/bsi/articles/knowledge/attack-patterns, last available September 2019.

[25] Boyer, J. (2001). Canonical XML version 1.0 (No. RFC 3076), https://tools.ietf.org/html/rfc3076, last available September 2019.

[26] Dominique Righetto (2016). XML Security Cheat Sheet, https://github.com/OWASP/CheatSheetSeries/blob/master/ cheatsheets/XML$security_Cheat_Sheet.md, last available September$2019.

[27] Fernando Arnaboldi IOActive (2016). Assessing and Exploiting XML Schema's Vulnerabilities, https://ioactive.com/wp-content/uploads/2018/ 05/Arnaboldi-XML-Schema-Vulnerabilities-1.pdf, last available September 2019.

[28] John Franco, Dept. Electrical Engineering and Computer Science, Cyber Defense Overview, Attack Patterns, http://gauss.ececs.uc.edu/Courses/c6055/lectures/PDF/ attackpatterns.pdf, last available September 2019.

[29] Quin L. & XML Core Working Group (2016) Extensible Markup Language (XML) Standard, https://www.w3.org/TR/2009/REC-xmlbase-20090128/, last available September 2019.

[30] RELAX NG Technical Committee, OASIS (2014) https://relaxng.org/, last available September 2019.

[31] The MITRE Corporation (2018). Common Attack Pattern Enumeration and Classification dictionary and classification, https://capec.mitre.org/data/definitions/3000.html, last available September 2019.

[32] The MITRE Corporation (2018). Common Attack Pattern Enumeration and Classification dictionary and classification, XML Schema Poisoning, https://capec.mitre.org/data/definitions/146.html, last available September 2019.

[33] The MITRE Corporation (2018). Common Attack Pattern Enumeration and Classification dictionary and classification, XML Routing Detour Attacks, https://capec.mitre.org/data/definitions/219.html, last available September 2019.

[34] The MITRE Corporation (2018). Common Attack Pattern Enumeration and Classification dictionary and classification, XML External Entities Blowup, https://capec.mitre.org/data/definitions/221.html, last available September 2019.

[35] The MITRE Corporation (2018). Common Attack Pattern Enumeration and Classification dictionary and classification, XML Entity Linking, https://capec.mitre.org/data/definitions/201.html, last available September 2019.

[36] The MITRE Corporation (2018). Common Attack Pattern Enumeration and Classification dictionary and classification, XQuery Injection, https://capec.mitre.org/data/definitions/84.html, last available September 2019.

[37] The MITRE Corporation (2018). Common Attack Pattern Enumeration and Classification dictionary and classification, XPath Injection, https://capec.mitre.org/data/definitions/83.html, last available September 2019.

[38] The MITRE Corporation (2018). Common Attack Pattern Enumeration and Classification dictionary and classification, DTD Injection, https://capec.mitre.org/data/definitions/228.html, last available September 2019.

[39] The MITRE Corporation (2018). Common Attack Pattern Enumeration and Classification dictionary and classification, XML Attribute Blowup, https://capec.mitre.org/data/definitions/491.html, last available September 2019.

[40] The MITRE Corporation (2018). Common Attack Pattern Enumeration and Classification dictionary and classification, XML Quadratic Expansion, https://capec.mitre.org/data/definitions/229.html, last available September 2019.

[41] The MITRE Corporation (2018). Common Attack Pattern Enumeration and Classification dictionary and classification, XML Entity Expansion, https://capec.mitre.org/data/definitions/197.html, last available September 2019.

[42] The MITRE Corporation (2018). Common Attack Pattern Enumeration and Classification dictionary and classification, XML Ping of the Death,

https://capec.mitre.org/data/definitions/147.html, last available September 2019.