UNIVERSITÀ DEGLI STUDI DI TRENTO

Department of Information Engineering and Computer Science

Master's Degree in
Computer Science

FINAL DISSERTATION

Drive-by Download Attacks as a Stackelberg Planning Problem

Supervisors

Chari.mo Prof. Fabio Massacci

Dott. Robert Künnemann

Student

Giorgio Di Tizio

Academic year 2017/18

Contents

Executive Summary

This thesis evaluates the security countermeasures implemented on the Internet to assess the impact of Drive-by download attacks in different scenarios. It provides the best combination of possible mitigation techniques for each threat and an estimation of the cost for the defenders.

Using Steinmetz et al.'s Stackelberg planning framework [49], we can identify optimal defender strategies for an optimizing attacker, for example, one that maximizes the number of assets he controls. This relies on a sound and complete formalization of the threat model, of the defender model, and an accurate cost estimation. The formal threat model is developed by analyzing the state-of-the-art techniques used to spread malicious code and the possible mitigations available. Crawling for data on the Internet infrastructure provides a representative environment in which the threat model is evaluated.

This thesis employs a planning algorithm to analyze the Internet defense against specific threats and to evaluate available countermeasures. It illustrates a formal description of the attacker's and defender's actions, as well as an estimation of the cost to implement such countermeasures. A crawling activity is employed to collect data from the Internet. This phase makes use of *target-specific* crawlers to generate a realistic representation of the dependencies on the Internet and to provide a description of the security countermeasures currently enforced.
This thesis presents a set of case studies where we examine the best combination of mitigations to reduce the impact of different kinds of threats. In particular, we focused on two classes of attackers:

- **Malicious Country:** we simulate cyber attacks performed by two European countries, the Netherlands and Great Britain, and by China.
- **Malicious Company:** we access the impact of Amazon and Cloudflare; two companies that provide many Internet services all over the world.

Due to the huge amount of information, for each case study, we preprocess the data to generate an optimized planning problem for the algorithm.
Limitations: in the first place the costs associated to each mitigation, if no other information were available, are the result of an estimation; the formal model developed is assumed to be sound and complete. Finally, due to memory and time constraints related to the planning algorithm, the results could not describe a global and complete picture of the entire Internet infrastructure.

The results show that the Internet infrastructure is based on strong dependencies between domains, name servers and other Internet elements that belong to different countries and organizations. These dependencies entail trust relationships that can be exploited by malicious entities. Companies and countries in a predominant position can affect political, economic and social behaviors through a mass-level infection of millions of users. In particular, we highlight a poor implementation of the security mechanisms in the Internet infrastructure; the results of the planning showed that the economic effort to significantly reduce the profit of attackers is really small. We believe that this situation is due to a lack of security experts in the development of websites and networking aspects, and primarily due to economic driven factors where the security is overcome by the convenience.

Introduction

The Internet infrastructure relies on many network services (for example DNS, CDN, email) that interact with each other generating a multitude of dependencies, often hidden and not clearly defined. The distributed nature of the Internet provides many benefits in terms of performance and accessibility but poses new threats to its security [48, 55]. Interactions on the Internet are often exploited by malicious users to achieve their personal goals. This is the case of the Drive-by download attacks [36, 31, 37], where the complex interaction between clients and web servers is exploited to infect the final users. Currently, the distribution of malware makes an extensive use of Drive-by download mechanisms to spread malicious code and compromise unaware visitors. An attacker can generate Drive-by download attacks in many different ways: relying on cross-site scripting attacks, malicious dynamic content, DNS poisoning and even on routing-level attacks. The rise of Advanced Persistent Threat (APT) groups sponsored by nations [11], driven by economic and political reasons, knows no limit to the assets and ability of the attackers and poses threats to the security of each user on the Internet.

Over the years, different protocols and mitigation techniques have been proposed to protect the Internet infrastructure in terms of routing (IPsec [24, 18]), name resolution level (DNSsec [3]), website delivery (HTTPS [41], DANE [17], HSTS [16], CSP [6]), public-key infrastructure (Certificate Transparency [26]), and third-party JS inclusions (SRI [8]).

No one has evaluated how these countermeasures can be used to mitigate such threats, or even to which extent this is possible, and at what cost. Which combination of proposals is the most cost-effective considering infrastructure? Are some of them too costly to deploy or simply less efficient than existing proposals? Are there countermeasures that are effective for any kind of attacker? Are these countermeasures enough to prevent APT attacks?

Scope of work: The goal of this thesis is to provide a formal description of the Drive-by download attacks and evaluate the impact of these mechanisms on real data of the Internet infrastructure. This thesis *does not* aim at providing a full enumeration of *all* possible techniques used to implement Drive-by download attacks; this enumeration, that typically depends on the level of description chosen, is out of the scope of this thesis.

To achieve this goal, it is necessary to formulate a threat model that describes the attacker's capabilities and the options for the defender. The evaluation of the effects on the Internet requires to collect real data through a crawling activity. The definition of different kinds of attackers, with different characteristics, allows to examine in deep the possible mitigations. Finally, the huge amount of data and the complicated dependencies between the Internet elements require to implement optimizations to reduce the complexity of the problems.

This thesis provides the following contributions:

- a formal description of the Internet infrastructure as well as the definition of available attacks and possible mitigations.
- the creation of a realistic representation of the Internet infrastructure through the collection of data via crawlers.

- the generation of optimized planning problems through a preprocessing of the data collected and the model developed.
- the evaluation of the security mechanisms on the Internet in different case studies. We assess the impact of country attackers like China, Great Britain, and the Netherlands. As well as companies like Amazon and Cloudflare. For each scenario, we analyzed the best countermeasures and their related cost in problems with more than 350k attacker's actions in combination with more than 7k defender's actions.
- an evaluation on the impact of errors in the crawling phase and in the formal threat model based on tainting.

The project is composed by different phases; in Fig. 1 is presented the complete process:

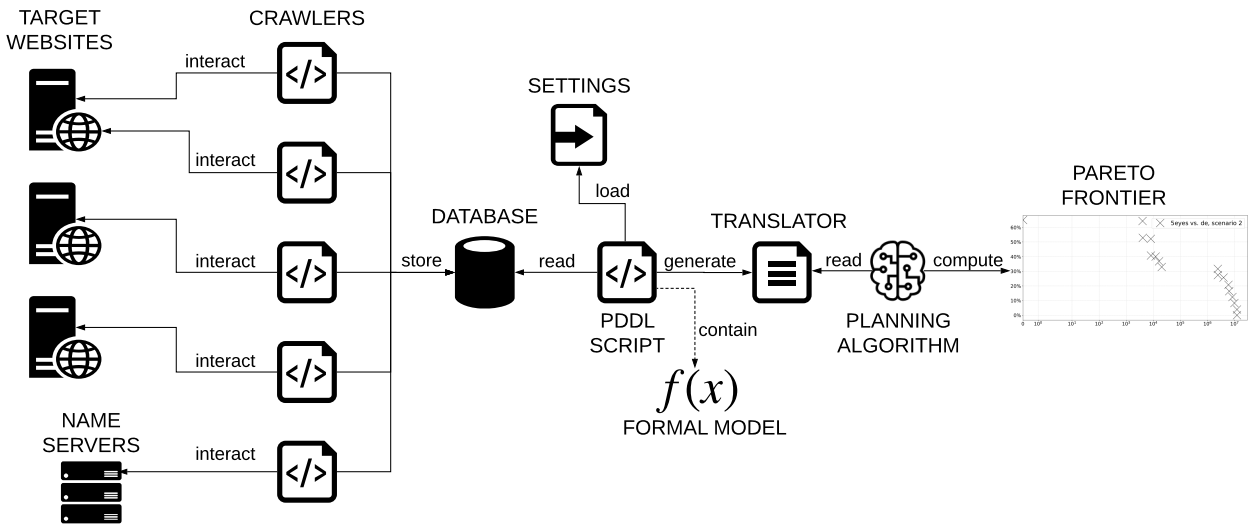

Figure 1: Project workflow

- **Phase 1:** a formal description of the model in terms of predicates and rules is implemented. The model contains the definition of the attacker's actions and the mitigations, that describe the information required for the problem.
- **Phase 2:** the information is retrieved from the Internet through a set of scripts; the websites are obtained from the top 5k Alexa domains list. For each target website, a set of crawlers interact with the web server and other websites that contain statistic information of the target; for example the Google Transparency Report[1] and the Alexa websites[2].
- **Phase 3:** the information collected by the crawlers is preprocessed and stored in a PostgreSQL database hosted in a server.
- **Phase 4:** the planning problems are generated through a script that receives as input the settings for the problems[3] and retrieves the necessary information from the DB. This script is implemented following the formal model defined in **Phase 1**. The final output of this script is not described in the PDDL language but it is an intermediate preprocessed file, called *Translator*, where the variables are instantiated with all their possible values.[4]

[1] *https://transparencyreport.google.com*
[2] *https://www.alexa.com/*
[3] For example, the dimension of the problem in term of data that must be retrieved from the DB and the definition of the attacker's asset.
[4] This procedure is an optimization that reduces the amount of time needed by the planning algorithm.

- **Phase 5:** the planning algorithm receives as input the *Translator* file and the Fast Downward planner computes the Pareto Frontier of the problem. The result is the set of mitigation strategies that are not dominated[5] by any other mitigation strategy.

The results showed that most of the countermeasures are effective to mitigate any kind of attacker. The cost of the mitigations is relatively small with respect to the economic damage of the infection; unfortunately, most of the Internet infrastructure does not enforce proper mechanisms. This situation leaves users vulnerable to Drive-by download attacks. The mitigations that can be implemented vary depending on the specific threat and the choice of certain defenses can be influenced by their cost. In the scenarios analyzed, the implementation of end-point level defense (for example HTTPS, HSTS, CSP, and SRI) represents one of the most cost-effective mitigation.

The thesis proceeds as follows:

- **Chapter 1:** discusses the formal threat model and the defender model;
- **Chapter 2:** presents the data collected through the crawling phase and examines the case studies of China, Great Britain, The Netherlands, Amazon, and Cloudflare.
- **Chapter 3:** reports related work on this topic.
- **Appendix A:** discusses Drive-by download mechanisms and the tools used by cybercriminals;
- **Appendix B:** contains the *complete* formal threat model, that describes the attacker actions and the mitigations;
- **Appendix C:** contains the explanation of the different crawlers used to retrieve information from the Internet;
- **Appendix D:** contains the description of the Python script used to generate the *Translator* file;
- **Appendix E:** discusses taint analysis over the data and the threat model;

This thesis is the result of my internship at the Center for IT-Security, Privacy and Accountability (CISPA) - Helmholtz-Zentrum (i.G.). The content of these chapters and appendixes is the product of *my* work at CISPA. The planning framework used has been developed in other CISPA projects [49, 50].

[5] A mitigation strategy dominates another if its cost is smaller than the other mitigation, while the attacker reward is not larger.

Chapter 1

Drive-by Download Attacks as a Stackelberg Planning Problem

In this chapter we will present the threat model developed for the mitigation analysis and we will provide an estimation of the cost for the defender's actions.

1.1 Planning

Our mitigation analysis is based on *Automated Planning*; a problem is described in terms of state predicates, initial state, a goal specification, and a set of actions that can be used to change the state. The purpose of automated planning is to find a sequence of actions that achieve the goal condition when applied in the initial state. In particular, we implemented *classical planning*, where it is assumed that all actions have deterministic effects and that the initial state is completely known.

Mitigation analysis through planning has been proposed recently in the literature with the goal of analyzing network penetration testing [49, 50]. They are described with a defender-attacker model, where the defender tries to limit the actions that an attacker can execute.
The states are described by a set of propositions; a state is an assignment to these propositions that makes them True. An action is given by a precondition *precond*, a boolean formula over proposition literals and a postcondition *postcond*, a conjunction over proposition literals, and is commonly written as:

$$\frac{precond}{postcond} \tag{1.1}$$

An action is executed only in those states in which the precondition *precond* is True. The application of an action changes the state accordingly to the postcondition expressed in *postcond*, making the boolean formula True.[1] The actions describe specific attacks implemented by an attacker; each attack has associated a *reward*, that estimates the damage generated, while the defender actions have associated a cost, that describes the effort required to enforce such mitigation. The planning analysis is implemented as a Stackelberg game where a defender gets to move first, and the subsequent move of the attacker is executed without additional actions of the defender.
The final result of the algorithm is a Pareto frontier P of mitigation strategies, i.e., the set of mitigation strategies which are not dominated[2] by any other mitigation strategy.

1.2 Formal Threat Model

Our threat model presents different types of attacks, that can be implemented at different levels of the Internet infrastructure. Certain types of attacks are not feasible for any kind of attacker, because they

[1] In our model we implemented *monotonic* attacker actions, which implies some algorithmic advantages. In other words, the preconditions and postconditions are composed only by positive literals; this implementation describes the case in which an attacker continuously gains new assets but it never loses assets previously acquired.

[2] A mitigation strategy dominates another mitigation strategy if its cost is strictly smaller while the attacker reward is not larger, or if the maximal attacker reward is strictly smaller while the cost is not larger.

require to have access to sensitive Internet resources. Our rules are able to describe attacks that can be mount by different actors: from State-sponsored APT to small hacker groups. Our formal threat model takes into account the nature of the attacker and traduces this information in the set of actions that are feasible for the considered actor. These scenarios allow measuring the impact of *Drive-by download* attacks on the Internet and the efficacy of the countermeasures in different scenarios with a different level of danger.
One of the advantages of a drive-by download attack is that presents a higher stealthiness with respect to other mechanisms of malware propagation because the compromise is carried through legitimate network requests. In our model we ignore attacks that can be easily detected and that can bring to a global exposure, for example BGP hijacking attacks, assuming a "sneaky" attacker.

1.2.1 Attacker reward

The goal of the attacker is to deploy malware in as many clients as possible through a malware distribution network; this approach requires to compromise web servers. We evaluate the impact of an attacker in terms of the ratio between the number of malicious web pages that will be accessed by the users and the total number of web pages visited. For each website, an estimation of the number of visitors in a month[3], for a certain group of countries *Countries*[4], is used as reward for the attacker.
In case the attacker is able to compromise the web server for a certain set of countries, its reward is given by the sum of the number of visitors of the considered countries as shown in 1.2

$$Reward = \sum_{i \in Affected\_Countries} Visitors_{i,d} \tag{1.2}$$

where:

- $Visitors_{i,d}$ is the estimated number of visitors for the Website d from the Country i;
- $Affected\_Countries$ is the set of countries that are affected by the attacker. $Affected\_Country$ is a subset of the $Countries$ set ($Affected\_Country \subseteq Countries$);

If an attacker compromises two different domains A and B, it is possible that some visitors of the domain A are also visitors of the domain B; therefore the sum of the rewards could present a shared group of visitors; some domains share more users with each other (*thehackernews.com* and *wired.com*) than others (*nytimes.com* and *lefigaro.fr*), due to lack of data we cannot infer how much this intersection weights in the total number of compromised users for the different cases considered.

1.2.2 Attacker actions

The threat model describes the attacker actions as a set of rules; each rule identifies a specific attack, underlining its requirements and its consequences on the Internet infrastructure. A set of predicates represents the elements on the Internet and their relations like DNS servers, CAs, routing connections, etc.; Table 1.1 contains the predicates defined in the model. The complete model and the list of predicates is presented in the Appendix B. The attacker has access to an initial asset composed of IPs, autonomous systems, domains, name servers, and certificate authorities; this initial asset depends on the type of attacker considered in the planning problem.[5]

In the following, we present some of our rules that compose the threat model, the rules vary from web attacks to routing and protocols attacks. The representation of each attack follows the structure presented in Section 1.1, where *precond* represents the set of conditions that are required to implement the attack and *postcond* represents its consequence.
For example, the rule 1.3 can be read as follow: given any domain d ($d \in D$) vulnerable to XSS attacks ($XSS(d)$) that does not implement a *secure* content security policy ($\neg CSP(d)$), it can be compromised ($C(d)$) through a cross-site scripting attack.

[3]This information is retrieved from Alexa. We assume that the attacks are stealthy and can remain active for this interval of time.

[4]This set contains the first 10 countries that present the highest number of visitors for the first 10 Alexa domains: CN, US, JP, IN, GB, DE, KR, FR, BR, TW.

[5]For example, State-sponsored APT groups will have a richer asset with respect to small hacker groups due to economic and political support.

Table 1.1: Threat Model Predicates[6]

Predicate	Description
$x \xrightarrow{\text{loc}} cn$	The element $x \in AS \cup IP \cup D \cup NS$ is located in $cn \in Country$
$d \xrightarrow{\text{A}} i$	The element $d \in D$ has address $i \in IP$
$i \xrightarrow{\text{orig}} a$	The element $i \in IP$ belongs to $a \in AS$
$c \xrightarrow{\text{JS}} d$	The element $d \in D$ contains a JS script hosted in the element $c \in D$
$e \xrightarrow{\text{DNS}} d$	The element $e \in NS$ can be queried to resolve the DNS translation of $d \in D$. In other words e is one of the authoritative name servers of d
$a \xrightarrow{\text{RTE(b)}} c$	Given $a, b, c \in AS$, the route from a to c passes through b
$C(x)$	The element $x \in AS \cup IP \cup D \cup Country \cup NS$ is compromised. In case $x \in D \cup NS$, the predicate means that x can be used in a **Drive-By Download** mechanism. This predicate will be called *Globally compromised*
$C(c, d)$	The element $d \in D$ is considered compromised for all the clients that belong to $c \in Country$. This predicate will be called *Country compromised*
$XSS(d)$	The element $d \in D$ is vulnerable to XSS
$CSP(d)$	The element $d \in D$ implements a **secure** Content Security Policy
$UpgradeRequests(d)$	The element $d \in D$ forces to use the HTTPS protocol for all the resource requests. This predicate describes the field `upgrade-insecure-requests` in the CSP
$SRI(d, c)$	The element $d \in D$ implements the Sub-Resource Integrity mitigation for **all** the resources, used by d, stored in $c \in D$. It is assumed $d \neq c$
$IPsec(a, b)$	The packets routed between $a \in AS$ and $b \in AS$ are protected via IPsec
$DNSsec(f)$	The element $f \in NS$ implements DNSsec
$HTTPS(d)$	The element $d \in D$ implements HTTPS
$l\_HTTPS(d, e)$	**All** the JS resources, used by $d \in D$ and hosted in $e \in D$, are retrieved over HTTPS
$HSTS(d)$	The element $d \in D$ implements the header `Strict-Transport-Security`
$Redirect(d)$	The element $d \in D$ redirects HTTP connections to HTTPS. The redirection is either Temporary (Status code 302) or Permanent (Status code 301)
$CT(d)$	The digital certificates, for the element $d \in D$, are signed by CAs that are compliant with the Certificate Transparency
$DANE(d)$	The element $d \in NS$ implements DANE protocol
$I^{\text{DNS}}(d)$	The DNS resolution of the element $d \in D$ is compromised. This predicate will be called *Globally compromised DNS*
$I^{\text{DNS}}(d, e)$	The DNS resolution of the element $d \in D$ is compromised for the clients in $c \in Country$. This predicate will be called *Country compromised DNS*
$I^{\text{R}}(a, c)$	The route between $a, c \in AS$ is compromised
$I\_CA(d)$	The element $d \in D$ is vulnerable to certificate authority attacks

Content Compromise

The attacker can exploit Web application vulnerabilities, for example cross-site scripting (XSS) vulnerabilities, to compromise the content of the website. In this case, the web server is considered compromised and can be used as a *landing page* in the malware distribution network.

$$\frac{d \in D \quad XSS(d) \quad \neg CSP(d)}{C(d)} \tag{1.3}$$

We considered as mitigation the implementation of a *secure* Content Security Policy (CSP), whose main goal is to protect websites against XSS attacks.

[6]Some predicates are represented with an arrow notation. The set of predicates vary over the set of $AS, IP, D, Country, NS, CA$.

Third-party JS Injection

The advent of the dynamic Web and, later on, the Web 2.0 increased the complexity of websites. Each content in the Internet requires to access a set of HTML,CSS and JS files; often some of these resources are stored in third-party servers.
Once a client connects to a website, it automatically requests the external resources to a content delivery network (CDN); this approach improves the performance but presents some security problems. The CDN has full control over the content of a website; if one of these third-party servers is compromised, it can inject malicious code in all of these websites [14]. In this case, the attacker can insert malicious JS code to redirect the victims to an Exploit Kit (EK).

$$\frac{d, c \in D \quad c \xrightarrow{\text{JS}} d \quad \neg SRI(d, c) \quad C(c)}{C(d)} \tag{1.4}$$

We considered as mitigation the implementation of Subresource Integrity (SRI) on the JS resources retrieved from third-party servers; this mitigation allows to provide to the browser a hash of the trusted script that will be compared with the one of the received file.

DNS Compromise

The main purpose of the DNS protocol is the resolution of symbolic hostnames into their related IP addresses. To resolve a domain name, the client sends a query to a recursive DNS resolver; if the information is not present in the cache, the recursive DNS traverses the DNS hierarchy structure, until it reaches an authoritative name server (NS) for the target domain. If the attacker has access to one of the NS queried in the DNS resolution path, then the entire DNS resolution is compromised. The attacker can modify the content of the DNS records and redirect the client to a malware distribution site.[7]
In our model the attacker is not able to compromise root servers; this situation does not fulfill the requirement of a "sneaky" attacker. Furthermore, if the root NSs are compromised, then the entire DNS protocol is not reliable.

$$\frac{d \in D \quad e \in NS \quad e \xrightarrow{DNS} d \quad C(e)}{I^{\text{DNS}}(d)} \tag{1.5}$$

Route Compromise

Our model describes routing attacks at the autonomous systems (AS) level; the Border Gateway Protocol (BGP) [40] is used to compute the AS path for a destination. The routes from an AS depend on the routing policies implemented; these policies are influenced by political, economic and security aspects. The extraction of this routes is described in Section 2.1.
In case the attacker has control over an autonomous system within the AS path, the routing path is considered compromised; if specific conditions are met the attacker can drop and replace packets without being detect[8]. The advantageous position of the attacker is not the result of a BGP hijacking but is given by the Border Gateway Protocol.

$$\frac{a, b, c \in AS \quad a \neq b \neq c \quad a \xrightarrow{RTE(b)} c \quad \neg IPsec(a, c) \quad C(b)}{I^{\text{R}}(a, c)} \tag{1.6}$$

We considered as mitigation the implementation of IPsec to encrypt the communication between the sender and the destination AS.

[7] This attack is described in the rule 1.9 and 1.10.
[8] This type of attack is described in the rule 1.11

Route to Web Server Compromise

If the route from a client to a web server is compromised, the attacker can implement a MITM attack and alter the communication; such attack is possible if the client does not authenticate the web server's communication via cryptographic protocols. The implementation, by the web server, of HTTPS is not enough to prevent a MITM attack; by default, if not explicitly defined[9], browsers access resources via the insecure HTTP protocol [30][10]. In this case, the client cannot authenticate the web server, therefore the attacker can eavesdrop the unencrypted TCP connection, drop the packets directed to the web server, and send malicious responses. Even if an HTTP server is implemented to redirect the connection to an encrypted one, the first access of each connection is insecure and can be prone to MITM attack.
To force the browsers to access the website via a secure protocol, the HSTS policy must be implemented. This policy defines a header that must be sent over a secure communication[11], therefore it is necessary to implement a redirection to HTTPS. Furthermore, if a web server with HTTPS enable is vulnerable to certificate authority (CA) attacks,[12] then a malicious CA can forge digital certificates for the domain and use them to authenticate connections to malicious web servers. In this case HTTPS is not effective to protect the domain.

$$\frac{\begin{array}{c} e{\in}Country \quad d{\in}D \quad a,c{\in}AS \quad d \xrightarrow{orig} c \quad a \xrightarrow{loc} e \quad I^{\mathrm{R}}(a,c) \\ (\neg HTTPS(d) \vee (HTTPS(d) \wedge I\_CA(d)) \vee \\ (HTTPS(d) \wedge (\neg Redirect(d) \vee (Redirect(d) \wedge \neg HSTS(d))))) \end{array}}{C(e,d)} \tag{1.7}$$

In our model we do not consider the possibility to implement a bootstrap MITM attack: this vulnerability happen when a HSTS secured website is initially accessed through HTTP, because no previous HSTS information were available.[13] To mitigate this type of attack, browser vendors include a HSTS pre-loaded list[14], which ensures that websites in the list are directly accessed through HTTPS.

Route to Name Server Compromise

If the route from a client to a name server passes through an AS under the control of the attacker, the attacker can perform a DNS cache poisoning. Due to the fact that the DNS protocol does not require any authentication, the attacker can send to the victim a malicious DNS resolution; the MITM position of the attacker does not require to guess the *Query ID* and the port used by the resolver. The malicious DNS resolution will redirect the traffic to the malware distribution network of the attacker.

$$\frac{\begin{array}{c} a,c{\in}AS \quad a \xrightarrow{loc} e \quad e{\in}Country \quad f \xrightarrow{orig} c \\ f \xrightarrow{DNS} d \quad I^{\mathrm{R}}(a,c) \quad \neg DNSsec(f) \end{array}}{I^{\mathrm{DNS}}(d,e)} \tag{1.8}$$

The DNSsec protocol is used to mitigate this kind of attack; the DNS response is authenticated via cryptographic signatures.

From DNS to Domain Compromise

If the attacker compromises an authoritative NS of a web server ($I^{\mathrm{DNS}}(d)$[15]), it can insert malicious DNS records to redirect the victim to an Exploit Kit. This attack is feasible if the client does not access the Website via a secure protocol that requires authentication[16] or the HTTPS is make ineffective due

[9] I.e., the *https* protocol is not present in the URL.
[10] This behavior is due to the fact that the HTTPS protocol has been proposed after the HTTP protocol and it is not globally deployed.
[11] Otherwise the source cannot be trusted.
[12] See rules 1.12 and 1.13
[13] Even if the web server redirects to HTTPS.
[14] *https://hstspreload.org*
[15] See *DNS Compromise* rule in Section 1.2.2
[16] As previously explained in the *Route to Web Server Compromise* rule.

to certificate authority vulnerabilities ($I\_CA(d)$).[17]

$$\frac{I^{\mathrm{DNS}}(d) \quad (\neg HTTPS(d) \vee (HTTPS(d) \wedge I\_CA(d)) \vee (HTTPS(d) \wedge (\neg Redirect(d) \vee (Redirect(d) \wedge \neg HSTS(d)))))}{C(d)} \tag{1.9}$$

The same attack is implemented if the authoritative NS of one of the CDN is compromised. In this case the attacker can redirect, via malicious DNS response, a client to another CDN which provides malicious code. This code can either infect the victim or redirect the client to a malware distribution site. This attack is possible if the resource is addressed through an insecure protocol ($\neg l\_HTTPS(d, c)$) and the browser does not upgrade insecure connections to HTTPS ($\neg UpgradeRequests(d)$).

$$\frac{I^{\mathrm{DNS}}(c) \quad c \xrightarrow{JS} d}{\neg SRI(d,c) \quad \neg l\_HTTPS(d,c) \quad \neg UpgradeRequests(d)} \over C(d) \tag{1.10}$$

We consider as mitigation the connection via HTTPS and the implementation of Subresource Integrity for the JS resources retrieved from third-party servers ($SRI(d, c)$).

Inline JS Injection

If a resource is retrieved through an insecure connection ($\neg l\_HTTPS(d, d\_2) \wedge \neg UpgradeRequests(d)$) and the website does not implement the Subresource Integrity check ($\neg SRI(d, d\_2)$), then a MITM attacker ($I^{\mathrm{R}}(a, b)$) can eavesdrop and modify the content of the resource, injecting malicious code. New version of browsers implement a *Mixed content* [7] block, that does not allow to retrieve *active* resources[18] using an insecure protocol, if the Website is loaded over HTTPS. Therefore, the inline JS injection requires that the Website is accessed over HTTP ($\neg HTTPS(d) \vee (HTTPS(d) \wedge (\neg Redirect(d) \vee (Redirect(d) \wedge \neg HSTS(d)))))$.

$$\frac{\begin{array}{c} c \in Country \quad a,b \in AS \quad d,d\_2 \in D \quad d\_2 \xrightarrow{orig} b \\ a \xrightarrow{loc} c \quad I^{\mathrm{R}}(a,b) \quad d\_2 \xrightarrow{JS} d \\ \neg l\_HTTPS(d,d\_2) \quad \neg UpgradeRequests(d) \quad \neg SRI(d,d\_2) \\ (\neg HTTPS(d) \vee (HTTPS(d) \wedge (\neg Redirect(d) \vee (Redirect(d) \wedge \neg HSTS(d))))) \end{array}}{C(c,d)} \tag{1.11}$$

To mitigate this attack, the website must force the client to retrieve external resources through HTTPS or implement a Subresource integrity check.

Certificate Compromise

The attacker can compromise a Certification authority to generate digital certificates for the target domain. Our model assumes that, if the target domain does not have digital certificates signed by CAs compliant with the Certificate Transparency project ($\neg CT(d)$), it is unlikely that the domain owner will verify the CT logs to detect misissued digital certificate.[19]
In case the Certificate Transparency (CT) logs are not controlled, the web server can secure its certificates using DANE ($DANE(e)$); this protocol allows to define TLSA records in the DNS response of a NS, with the information about the list of certificates or CAs that can be trusted by the client. To successfully issue malicious digital certificates, the attacker requires that the authoritative NSes of the target website do not implement the DANE protocol. Once a certificate is generated by the compromised CA, the attacker can redirect clients to malicious web server, with a valid certificate for

[17] As described previously in the rule *Route to Web Server Compromise* in Section 1.2.2, we assume that the first connection to the website is not tainted.

[18] Scripts, stylesheet, flash resources.

[19] This simplification is due to the fact that current browsers are not forced to verify the SCT.

the target domain.

$$\frac{\begin{array}{c} a{\in}CA \quad d{\in}D \quad C(a) \\ e \xrightarrow{DNS} d \quad \neg CT(d) \quad \neg DANE(e) \end{array}}{I\_CA(d)} \tag{1.12}$$

The attacker can generate new certificate for a web server, even if the DANE protocol is implemented, if it has control of the authoritative NS for the domain; in this case the attacker can modify the TLSA records and insert malicious certificates generated by the compromised CA.

$$\frac{\begin{array}{c} a{\in}CA \quad d{\in}D \quad C(a) \quad e \xrightarrow{DNS} d \\ \neg CT(d) \quad DANE(e) \quad C(e) \end{array}}{I\_CA(d)} \tag{1.13}$$

1.3 Formal Defender Model

The defender model is composed by a set of actions that aims to minimize the attacker reward presented in Section 1.2. Each action has associated a cost and it is proposed as mitigation for a specific attack. The cost is based on publicly available data and can be seen as a general description of the effort that the defender must put to mitigate a specific attack.

1.3.1 Content mitigations

To secure the content of a website against Cross-site scripting attacks, we propose the implementation of a *secure* Content Security Policy [51]. The core of the CSP is the definition of a list of trusted domains as permitted sources of external resources and the block of inline scripts in the web pages. The implementation of a secure policy will reduce the probability that a malicious user will be able to exploit XSS vulnerabilities in the Web application.
In the different versions of the specification, the CSP added many functionalities, that allow to protect against different types of attacks. In our mitigation proposal, we focus on the implementation of a CSP that protect Web application against Cross-site scripting attacks[20].
CSP, as well as many other security-based HTTP headers, is not widely deployed in the Internet [13][21]; furthermore, almost all of the implemented policies are bypassable [54]. For this reason, we assume that the CSP mitigation follows the guidelines proposed by Weichselbaum et al. [54].

$$\frac{d \in D}{CSP(d)} \; c = \$(\#inline + \#ext + \#event) * 1\,\text{h} * 34.9\,\$/\text{h} \tag{1.14}$$

`Cost estimation:` The implementation of a secure CSP is strictly related to the specific website; the number of inline scripts, external JS resources, inline event handlers and the presence of advertisements are some factors that influence its complexity. We assumed that the CSP implemented is based on the level 3 specification[22], where a *nonce-based* protection can work in parallel with a *whitelist-based* approach, and dynamically generated scripts can be handled with the `strict-dynamic` keyword. We modeled the cost for a website as a consultant cost[23] that depends on the web page structure. A *nonce-based* approach does not require to refactor the application in case inline scripts are present but, vulnerabilities within the scripts, allow to bypass the mitigation; we estimate a cost of 1 h of consultant work to analyze each inline script. The same considerations hold for the external scripts: injections into the `src` attribute or, in case of a whitelist approach, script execution bypasses [54] can evade the restrictions of the CSP. In case the web page presents inline event handlers incompatible with the CSP, it is necessary to refactor the website; we associated a 1 h of consultant work for each inline event handler that must be refactored. The implementation of the policy requires a *Trial and error* approach and a period of testing[24]; we estimate at least a week to analyze the behavior of the

[20] We will analyze in a different section a specific field of the CSP to mitigate inline JS injection.
[21] This scenario is confirmed by the results of our crawling activity.
[22] *https://www.w3.org/TR/CSP3*
[23] We assumed a per hour cost for a Security Engineer. Source: *https://www.payscale.com/research/US/*
[24] CSP allows the implementation of a `report-only` mode.

CSP[25], with an additional cost for the testing equal to 56 h. This cost estimation allows to have a coarse idea of the amount of effort that a domain must employ to define a *secure* CSP.

Attackers can target CDN to compromise hundreds of thousands of sites that depend on it; all the external resources retrieved from these servers can be modified to infect website's users. We can secure websites against malicious modification of JS resources via Subresource integrity: the website provides a hash of the resource[26] hosted in a third-party server, the file is then retrieved from the CDN and its hash is compared with the integrity value. If the value does not match, the browser will not execute or load the resource.
This type of attacks are widespread and can be implemented in large scale as shown by the *Great Cannon* attack [27, 1]; the implementation of SRI for the resources provided by the Baidu servers, could have reduced the impact of this attack. [53]
The SRI mitigation can be implemented for all the resources that do not change over the time, for example, versioned JS libraries. In case a resource is modified its integrity value must be updated, otherwise, the hash comparison will fail;[27] furthermore, the SRI cannot protect dynamically generated scripts.

$$\frac{d, c \in D \quad c \xrightarrow{\text{JS}} d}{SRI(d, c)} \; c = \$280 \tag{1.15}$$

`Cost estimation:` We assigned a consultant cost of 1 day for implementing SRI on a website; exist several tools [23] that can be used to help developers in the inclusion of the integrity value. We did not consider any backup cost to handle mismatches of hashes.

1.3.2 Secure protocol mitigations

The HTTP connection between a client and a website can be secured through TLS to achieve authentication, integrity, and confidentiality. HTTP is the default protocol used by browsers; a connection to a website is carried over HTTP if a different protocol is not specified. A website, that correctly implements the HTTPS protocol, can be vulnerable to MITM attacks if a client accesses the server through an insecure connection. To prevent this risk the client must be redirected to an HTTPS connection.
To force clients to access a web server directly through HTTPS, web servers can implement an HSTS policy. The HSTS header is sent to inform the browser that the specific domain and, if explicitly declared, all its subdomains must be accessed via HTTPS for a certain period of time. All major browsers present a *HSTS preload* list that contains a set of domains for which the browser automatically creates an HTTPS connection.

$$\frac{d \in D}{HTTPS(d)} \; c = \$280 \tag{1.16}$$

$$\frac{d \in D \quad HTTPS(d)}{Redirect(d)} \; c = \$280 \tag{1.17}$$

$$\frac{d \in D \quad HTTPS(d)}{HSTS(d)} \; c = \$280 \tag{1.18}$$

`Cost estimation:` We calculated the cost for implementing a secure connection as a consultant cost of 1 day. The implementation of HTTPS on a website requires, in the worst case, to manually modify the configurations of the web server. We did not include the cost of the digital certificate, due to the

[25] This period of time is strictly related to the number of visitors; higher is the number of visitors, faster is the generation of a complete report for the CSP.

[26] Typically a script or a stylesheet.

[27] To temporarily handle this mismatch the external resource can be retrieved from a local repository. This approach reduces the advantages provided by the CDN.

presence of free CA like *Let's Encrypt*[28]. The same considerations hold for the redirection mechanism and the implementation of an HSTS policy.

Even if a website is accessed via HTTPS, an attacker can exploit subresources retrieved through HTTP to implement a MITM attack. This situation is called Mixed content; resources retrieved through an insecure connection are displayed in a web page along with content loaded over HTTPS. A MITM attacker can intercept the insecure connection and alter the external resource to infect visitors. Modern browsers block Mixed content for active resources that can compromise the entire website; passive mixed content is typically notified through a warning message. Even if client-side countermeasures are present, web servers must not use insecure URL for subresources. Different browser vendors can handle in a different way Mixed content, and outdated browsers are not automatically protected from this threat.
Subresources can be loaded through a secure connection either explicitly defining the HTTPS protocol or using a Content security policy with a `upgrade-insecure-requests` directive. The latter technique allows to inform the browser that all the site's insecure URL must be replaced with HTTPS.
This mitigation requires that the external resources are available over HTTPS.

$$\frac{d, c \in D \quad c \xrightarrow{\text{JS}} d \quad HTTPS(c)}{UpgradeRequests(d)} \; c = \$280 \tag{1.19}$$

$$\frac{d, c \in D \quad c \xrightarrow{\text{JS}} d \quad HTTPS(c)}{l\_HTTPS(d, c)} \; c = \$280 \tag{1.20}$$

`Cost estimation:` We assigned a consultant cost of 1 day for implementing HTTPS for all the external resources of a website. The deployment of this mitigation requires to check that the subresources are available through a secure connection and to force the use of HTTPS.

1.3.3 Routing mitigations

To prevent routing attack from a malicious AS in the AS path, packets routed between two autonomous systems are encrypted and authenticated through IPsec [24, 18]. We assumed that the implementation of an IPsec connection is not influenced by the geolocation of the endpoints, and it is the result of a private agreement between AS owners.

$$\frac{a, b \in AS}{IPsec(a, c)} \; c = \$56000 \tag{1.21}$$

`Cost estimation:` The cost of deploying an IPsec connection between two AS with a link speed of $10\,\text{Gbit}\,\text{s}^{-1}$ is roughly of 56 000$; this estimation includes the cost of two dedicated routers for 24 000$ each [39] and the consultant cost for configuration and maintenance per year (about 80 consulting hours) [5].

1.3.4 DNS-level mitigations

To prevent DNS cache poisoning attacks, the DNS data are authenticated and protected through DNSsec [3]. The adoption of DNSsec by end users is still very low [2] therefore we assumed that the DNSsec verification is implemented in the recursive resolver of the ISP [32][29] and the route from the user to the recursive DNS resolver is secure. DNSsec is deployed in the root servers and in most of the TLDs [21]; we estimated the overall cost to activate DNSsec in all the authoritative NS managed by a company. This mitigation is implemented only for those domains, for which the DNSsec is deployed

[28] This additional cost depends on the company strategy.
[29] The DNSsec mitigation is useless in case the clients do not verify the signature.

in all the elements of the chain of trust up to the root NSs.

$$\frac{\forall f \in NS : f \xrightarrow{DNS} d}{DNSsec(f)} \; c = \$366342 \tag{1.22}$$

DNS-Based Authentication of Named Entities (DANE) [17] is a DNS-level mitigation against vulnerabilities in the CA model [34]. This protocol allows to retrieve, through DNS queries, both DNS resolution and certificate information for a certain domain name. The `TLSA` record presents different usage field values that allow working along with or without the CA model. To authenticate and protect the certificate information, DANE requires to work over DNSsec.

$$\frac{\forall f \in NS : f \xrightarrow{DNS} d \quad DNSsec(f)}{DANE(f)} \; c = \$4000 \tag{1.23}$$

`Cost estimation:` We estimated an overall cost of 366 342$ to deploy DNSsec; this value is the maximum CAPEX extracted from a survey [32] among 19 companies. We assumed that this cost is representative for the set of companies that manage the authoritative NSs in the database.[30]
The cost of implementing DANE should not require a big effort in case DNSsec and TLS protocols are already deployed; the major operation is the creation of the `TLSA` record for the certificate.[31] We fix the cost for this process to be no more than 4000$, as in [50], corresponding to 40 working hours.
Currently, major browsers do not automatically validate DNSsec and DANE; this procedure can be achieved through plugins. We assume that an extensive adoption of these protocols will encourage browser vendors to implement these functionalities.

1.3.5 CA mitigations

The authentication of a web server on the Internet relies on digital certificates issued by certificate authorities. In the last years, this model showed many flaws that brought to mistakenly issued certificates and CA compromise. Google presented the Certificate Transparency [26] project that aims to detect misissued certificates; this is done through a set of publicly available append-only certificate logs, that contains all the certificates present on the Internet. Domain owners can verify the list of digital certificates issued for their domains and detect the presence of unauthorized ones.
Chrome browser requires that all the certificates issued after April 2018, must be compliant with the CT policy; other browser vendors, for example Mozilla [29], are planning to include support for the CT project.

$$\frac{d \in D \quad HTTPS(d)}{CT(d)} \; c = \$280 \tag{1.24}$$

`Cost estimation:` This mitigation does not require any effort for domain owners; we assigned a consultant cost of 1 day to choose CAs compliant with the CT policy, assuming that in the short-term future, all the browsers will implement this functionality. Chrome covers more than 65% of the desktop browser market share [32], therefore this mitigation concerns many Internet users. In our cost estimation we did not include the implementation of monitors to detect unauthorized certificate; exist public Websites to access CT logs[33].

[30] The maximum value allows to represent the worst-case scenario.
[31] Exist tools to automatically generate this records, for example *https://ssl-tools.net/tlsa-generator*.
[32] *http://gs.statcounter.com/browser-market-share/desktop/worldwide*, last access: 20/09/2018
[33] For example *https://transparencyreport.google.com/https/certificates*.

Chapter 2

Experimental Validation over the Internet

In this chapter we will describe the data collected from the Internet and we will present an automated mitigation analysis for two case studies.

2.1 Data acquisition

Starting from the top 5k Alexa domains, we extracted the required information via DNS queries and Web crawlers.
We had access to a list of websites with reflected XSS vulnerabilities on date 29 April 2018 from a project of the Secure Web Applications Group of CISPA[1], and we considered those domains that were present in the 5k Alexa list.

2.1.1 Server

We extract from the Internet the information about web servers, authoritative name servers, and content delivery networks.

Web Server

For each domain in the 5k Alexa list, we obtain the IP address of the website[2] and we analyze the security countermeasures implemented. We extract the security headers `Strict-Transport-Security`, `X-XSS-Protection`, `X-Content-Type-Options`, `X-Frame-Options` and `Content-Security-Policy`. In case a website implements CSP, we collect a set of fields regarding XSS and protocol mitigations.[3]
We access each domain through a secure connection to analyze if HTTPS is implemented; next, we check if the web server redirects insecure connections to HTTPS, and we collect the sequence and type of redirections.
We extract the list of JS resources statically and dynamically loaded and we analyze the presence of subresource integrity. From the source code of the initial page of the website, we acquire the number of inline JS, *eval()*, *Function()*, *window.setInterval()*, *window.setTimeout()* and *onclick* methods.
For each website, we extract the list of CAs that signed a digital certificate for the domain and we check their presence in the Google Certificate Transparency logs.

Content delivery network

Starting from the URI of the JS resources, we identify the list of CDNs from which the external resources are retrieved and we obtain their IPs; we analyze the type of protocol used and if the resources are available via HTTPS.

Authoritative NS

For each website and CDN, we collect the list of authoritative name servers that are contacted during iterative DNS resolution; we obtain the IP for each server and we analyze the DNS records to detect

[1] *https://swag.cispa.saarland/*
[2] We eliminated those domains that do not resolve to a website.
[3] The most relevant fields are: `default-src`, `script-src`, `style-src`, `object-src` and `upgrade-insecure-requests`.

DNSsec and DANE implementations.

2.1.2 Routing and Network Information

To model the internet connectivity, i.e., connectivity between ASes, we collect traceroutes provided by RIPE Atlas [42] for the set of Autonomous systems considered in our dataset. We observed traceroutes that have the domains from our dataset as their destination.

2.1.3 Countries

For each NS, website, and CDN, we include in our dataset their geolocation. We link IPs to ASes using RIPEstat database service [33] and we map ASes to countries using the MaxMind database [28]. In addition, each CA is mapped to a specific country using the information stored in the `issuer` section of the digital certificate.

2.2 Result and Evaluation

The data collected in Section 2.1 are used to implement a set of planning problems from the threat model. We compute different scenarios with different kind of attackers, from malicious countries to companies, to describes various levels of risk for the Internet infrastructure.
The description of Drive-by download problems at the Internet level presents a huge amount of information; due to the fact that a planning problem grows exponentially with the number of mitigation actions, the planning algorithm is unable to manage all the possible combinations. The data set is reduced to a subset of the 5k Alexa domains, depending on the specific attacker scenario[4]. We furthermore limit the maximum amount of memory space to 704 Gbit due to architectural constraints. Even with these simplifications, most of the problems required many days to compute the Pareto frontier; the majority of the time is dedicated to the pre-processing phase of the planning algorithm. All the planning problems ran on an Intel Xeon E5-4650L @ 2.60GHz machine with 32 physical and 32 HT cores.

We weight domains by their number of visitors in a month, using the data provided by Alexa, and we assigned these values as reward for the attacker. These data do not map directly to the number of visitors compromised due to the overlap of users for different domains. Thus, this number should not be interpreted as the number of users potentially affected, but as the total number of web pages visited (out of our subsets) that gets to try to infect a user. We normalize this number for the total number of web pages visited for the set of domains considered[5]. The final result represents the percentage of malicious web pages that users retrieve on the Internet. The results presented in this paper can be affected by different factors: first, the cost associated with each mitigation is retrieved through publicly available information and, in case no data were obtainable, the costs are the result of estimations; furthermore, the crawling activity describes a snapshot of the Internet that could not be representative of the current status, due to the deep changes that can be implemented in a short amount of time.

Table 2.1: Statistics Problems Complexity DB with top 100 Alexa Domains

Attacker	AS asset	Domains asset	NS asset	IP asset	CA asset	Attacker actions	Defender actions
US	79	264	338	733	32	333546	7173
DE	30	255	0	478	0	352512	7171
IT	12	105	0	110	0	333546	7168
Google	19	47	7	30	/	352512	7170
CloudFlare	8	22	8	25	/	350867	7169

[4]More powerful is the attacker, higher is the number of combinations that the planner must check and consequently the amount of time and memory required. A smaller number of domains reduces the complexity of the problem.
[5]In other words, the sum of the rewards for all the domains in the planning problem.

2.2.1 Attackers Identification

Simeonovski et al. [48] presented a set of metrics that allow to define possible attackers[6]. The metrics we extracted are: the number of Alexa domains, the number of domains hosting JS libraries and the number of name servers. The results presented by Simeonovski et al. are based on the top 100k Alexa domains. This dataset provides a good representation of the most powerful entities on the Internet. Their work analyzed the content of a single AS to access the influence of each company. We assumed that this simplification does not have a drastic impact on the results if the analysis is extended to all the ASes of each organization. The decision of the attackers is also influenced by the limitation of our planning algorithm; the Five Eyes countries or powerful states like US and China, represent interesting scenarios but, the corresponding planning problems are too big even with a reduced set of domains, as shown in Section 2.2.2. Table 2.1 shows the complexity for different types of attackers on a DB reduced to the top 100 Alexa domains; the number of attacker and defender actions represent the upper bound for the DB used. The complexity is, in the worst case, exponential in the number of executable defender actions, and this number grows linearly with the attacker asset. An attacker with a huge number of Internet infrastructures or more critical asset[7] will execute most of the available attacker's actions, increasing the number of executable mitigation actions.
We underline that the criteria used to identify the attackers depends on information security considerations and are not influenced by political orientations.

In appendix E, we will analyze the impact of errors in the data collected during the crawling phase and in the threat model. We will compare the results of the mitigation analysis in a certain scenario with tainted versions of the problem.

2.2.2 Case study: Malicious Countries

We evaluate the impact of malicious countries that want to spread malware on the Internet. The goal can vary depending on the country; from national security reasons to sabotage and espionage purposes. Many countries present cybersecurity task forces for national security and exist many state-sponsored APT groups that can have access to nations' Internet infrastructure [11].

Netherlands

The Netherlands has an important role in the Internet infrastructure as showed in [48]; in the top 100k Alexa domains, more than four thousand domains are hosted in the Netherlands and the number of name server located in this country is higher than the number of NS in Canada, Russia, and China.
Fig. 2.1 shows the Pareto frontier assuming as attacker the Netherlands with a restricted database on the top 100 Alexa domains. The Netherlands has a limited impact on the users of these target domains because the number of web pages that can be compromised by the attacker is negligible. The number of malicious web pages visited by any user in a month is one out of one million; the attacker's reward can be reduced to zero with the implementation of *secure* CSP on a small subset of domains.

Great Britain

Great Britain is a more powerful attacker than the Netherlands as shown in Fig. 2.1; even if, in the top 100k Alexa domains, the number of domains located in this country and the number of domains that host JS libraries is smaller than the Netherlands. The results show that Great Britain holds more critical Internet infrastructures, vital for the Internet ecosystem. Indeed, 97 out of 100 web pages visited by a user could contain malicious code generated by the attacker; the reward of the attacker can be set to 0 with a higher number of mitigation actions with respect to the Netherlands scenario but the overall cost is about half. The set of defender actions that can be implemented regard *secure protocol mitigations*. In particular, an adoption of HSTS policies for a set of domains, among which

[6]The attacker is a country or an organization.

[7]With the term *critical asset* we refer to an asset that contains fundamental Internet infrastructure; a resource is classified as fundamental due to the dependencies that other Internet elements in the dataset have with this resource. For example, `ns*.google.com` are a more critical asset than the name servers of a University organization in the top 5k list.

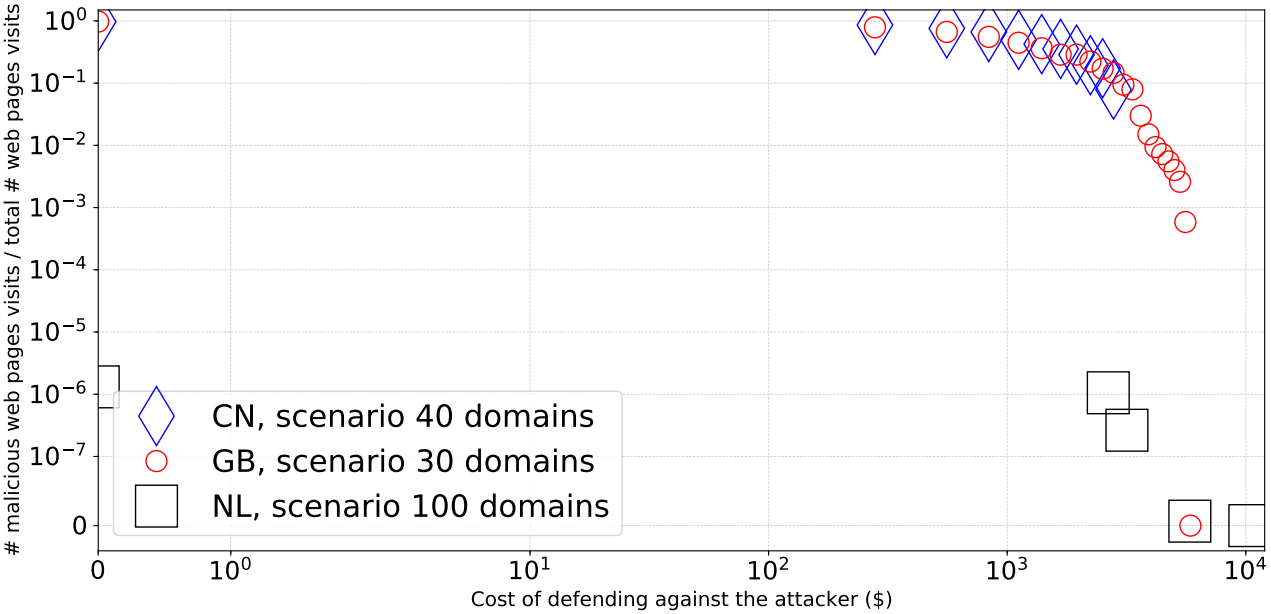

Figure 2.1: Results for malicious countries in log scale linear around zero. Note that the scenarios are w.r.t. a different number of domains. A Pareto frontier for a higher number of domains gives an upper bound for the Pareto frontier of a lower number of domains

reddit.com and *tmall.com*, and the implementation of HTTPS in parallel with redirection and SRI could reduce the impact of Drive-by download attacks to zero.

China

We evaluate the impact, in the Internet infrastructure, of one of the world's largest economies: China. Marczak et al. [27] claimed that the massive DoS attack against GitHub and GreatFire.org was produced by the Chinese government. The mechanism used, called Great Cannon, allows to implement a MITM attack on the unencrypted connections that transit China's network border. While the goal of the Great Cannon was to create a massive DDoS attack, this tool can be easily modified to deploy any kind of malware to the users.
Fig. 2.1 shows the result for a small scenario with the top 40 Alexa domains, the planning algorithm went out of memory, exceeding the limit of 704 Gbit. The mitigations require a higher economical effort than other scenarios like Great Britain and the Netherlands; roughly 3000 dollars allow to reduce significantly the web pages with malicious code but they are still not enough to reduce to zero the reward of the attacker[8]. Most of the defender actions are *secure protocol mitigations* on Chinese websites. Mitigations like HTTPS, redirection, HSTS, and `upgrade-insecure-requests` in the Content Security Policy are enough to make useless the Great Cannon tool.

2.2.3 Case study: Malicious Companies

Most of the Internet resources are controlled by a small set of companies [48]. If a company is compromised or it becomes malicious, the security of the entire Internet could be affected. We assess this impact with case studies about two American companies.

Amazon

According to [48], the colossus of the e-commerce owns more than two thousand domains in the top 100k Alexa and it is the company with the highest number of domains that host JS libraries[9].
Most of the feasible actions are routing attacks against name servers and web servers connections;

[8]The final reward could be reduced to zero with the implementation of additional mitigations, but due to memory limit we have not information regarding the right part of the Pareto.

[9]Roughly double the number of Cloudflare's domains and more than 10 times the number of Google's domains.

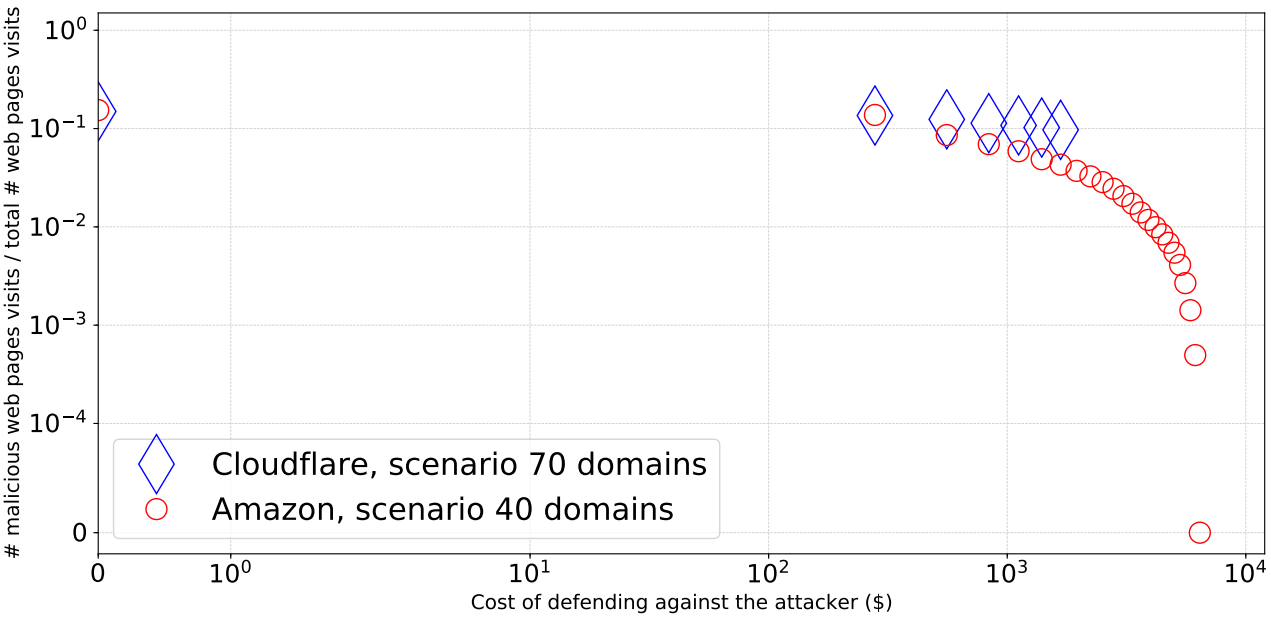

Figure 2.2: Results for malicious companies in log scale linear around zero. Note that the scenarios are w.r.t. a different number of domains

these attacks can be contained with the implementation of server-side mitigations like HSTS, HTTPS, SRI, and `upgrade-insecure-requests` in the Content Security Policy. With an overall cost of roughly four thousand dollars, the number of malicious web pages retrieved by the users can be reduced by approximately a factor 10; the total number of malicious web pages can be reduce to zero with an estimated cost of no more than 7000 dollars.
Some of the compromised entities are shared among different type of attackers; for example Amazon and Great Britain can both affect insecure connections to *baidu.com*; therefore, some of the mitigations are shared among different scenarios.

Cloudflare

Cloudflare provides many utilities among which DDoS mitigations, CDN and DNS services. According to [48], the American company owns more than 7000 domains and host JS libraries in more than 10 000 of the top 100k Alexa domains. Due to the DNS services offered, it also controls a large group of name servers. The results presented in Fig. 2.2 are a partial solution to the planning problem; due to the complexity of the scenario and the asset of the attacker, the planning algorithm exceed the memory limit considering the top 70 Alexa domains. The number of malicious web pages loaded by the users is high, 15 web pages out of 100 could contain malicious code injected by Cloudflare and does not decrease rapidly even with more than 2000 dollars of mitigations. Some of the mitigations proposed in the fragment of the Pareto frontiers regards some big American companies like *Apple* and *Microsoft*, that have a large number of visitors.

Chapter 3

Related work

In this chapter, we will discuss related work on Drive-by downloads, on the effects of malicious infrastructure, and on automated mitigation analysis.

Drive-by download

A number of papers have analyzed *Drive-by downloads*, studying their infrastructures and proposing new techniques to prevent this kind of attacks.
Grier et al. [12] described the evolution of the Drive-by download ecosystem and analyzed the families of Exploit Kits and malware present in the wild; additionally, they studied the lifetime of these infrastructures. Provos et al. [36] analyzed URLs to detect malicious URLs that initiate drive-by download; they constructed the related malware distribution network and identified the geographic locality of web-based malware. Furthermore they extensively studied the exploitation of advertisements to implement drive-by attacks; by contrast, we investigated potential future exploitations. A description of the prevalent mechanisms used to inject malicious content on websites is presented by Provos in [37]. Polychronakis et al. [35] described the life cycle of web-based malware and analyzed the network interaction generated by the infected machines. Cova et al. [9] proposed a modified browser to automatically detect and analyze malicious web pages. Narvaez et al. [31] collected malware samples via honeyclients and analyzed the performance of different AV vendors. Our work tries to prevent the infection in the first stage of the attack, protecting Internet connections; the AVs must be the last line of defense for the final user. Rajab et al. [38] described the most prevalent web-malware detection systems and showed methods that malicious websites employ to evade detection.

Distribution of malicious infrastructures

Shue et al. [47] analyzed the Internet at the AS level and showed that malicious activities are localized in specific ASes and are not uniformly distributed on the Internet. Simeonovski et al. [48] presented a model of the Internet infrastructure to assess attackers impact, without an analysis of the possible mitigations.

Automated mitigation analysis

Our work is closely related to the work of Speicher et al. [50, 49] and shares the base structure of the planning framework. While their work regards email sniffing, our model investigates a completely different scenario, with a formal description of the model at different levels of the Internet infrastructure. Simulated pentesting is based on the concept of attack graphs [52, 44, 45, 43], where are described the combination of actions used to compromise a certain target. In our model, we implemented a *monotonic* formulation of the boolean statements.

Chapter 4

Conclusion

In this thesis, we have presented an automated mitigation analysis of deployment benefits of secure protocols and configurations to mitigate Drive-by download attacks. Further, we estimated costs for implementing countermeasures at different levels of the Internet infrastructure, trying to assess the global effort to protect against the distribution of malicious code.
We demonstrated the effectiveness of this approach through the implementation of different case studies, based on data collected from the Internet through a crawling activity. We assessed the impact of threats posed by malicious countries analyzing the influence of China, The Netherlands, and Great Britain. We examined Amazon and Cloudflare, to access the damage produced by malicious code spread by companies. Additionally, we showed that the mitigations currently implemented on the Internet are not adequate to protect users against Drive-by download attacks and the economic effort required to secure the Internet infrastructure is reasonable for any kind of company.
An interesting future research direction could be to extend the formal threat model developed, with possible attacks through software vulnerabilities. For example, web servers and name servers with outdated and insecure software can be compromised by an attacker and used for a Drive-by download attack. Provos et al. [36] showed that 38% of the Apache servers run an unpatched version.
The algorithm can be extended to implement additional optimizations to reduce the computational complexity of the planning problem. Finally, the costs associated to each mitigation can be improved through information and statistics provided by companies that already implemented such countermeasures; an exchange of information would produce more realistic scenarios and more reliable results.

Appendix A

Drive-by Download and Exploit Kits

The distribution of malware through the Internet is changed from a push-based to a pull-based approach [36]. A *Drive-by download* attack is a technique that allows to deploy malware, exploiting vulnerabilities in the client machine. The procedure is composed by different steps:

- **Website compromise:** the attacker compromises a popular website exploiting vulnerabilities in the web server, in the Web application or using Malvertising[1];
- **Client redirection:** the client visits the compromised website and executes the malicious code present in the server response. This code, typically, forces the client to follow a sequence of redirects that leads to a malware distribution site[2];
- **Vulnerability exploitation:** the malware distribution site sends a malicious payload that compromises the victim;

The first phase of the attack can be implemented through different techniques:

- *Web server compromise:* the attacker exploits vulnerabilities in the server machine; Provos et al. [36] showed that 38.1% of the Apache servers and 39.9% of the server with PHP scripting, for which it was able to extract information, presented vulnerable software.
- *Content compromise:* the attacker compromises the content of the website either exploiting XSS and SQL vulnerabilities[3] or using third-party widgets.[4] [37]
- *Malvertising:* the attacker acquires advertisements in the target website and inserts malicious code. This mechanism allows to implement *Drive-by download* attacks without the need to compromise a server. Furthermore, another advantage is that ads are distributed to specific users that match certain criteria.[5] The use of Ad syndication, which allows selling advertising space to other advertising companies, generates a chain of trust that is typically abused by malicious users. The malicious content inserted into the ads, forces the client to follow a sequence of redirections that lead to the malware distribution site.[6] [36]

A.1 Drive-by Download distribution

Sue et al. [46] analyzed the distribution of malicious activities at the AS level; the results showed that malicious activities are not distributed uniformly over the Internet. Indeed exit ASes with a higher percentage of malicious IPs with respect to other ASes of the same size.

[1] In some cases it is possible that the primary goal of a web server is to deploy malicious software, but this approach is less frequent and presents obvious drawbacks.

[2] The number of redirections depends on the specific case [36].

[3] Common targets are blogs, profiles, and website that allow comments

[4] Embedded links to external JavaScript that provides additional functionalities to the web page.

[5] This information could regard browser type, version and installed plugins. [4]

[6] In some cases, the malvertising campaign directly inserts malware within the ads.

In the case of *Drive-by download* attacks, Provos et al. [36] showed that the 95% of the malware distribution sites map to only 210 ASes; furthermore 67% of the malware distribution sites and 64.6% of the compromised websites are located in China.
This nonuniform distribution of malicious activities allows to identify specific entities and Internet interactions that have a higher probability to be exploited by *Drive-by download* attacks; in other words, exist some scenarios where it is more likely that this type of attacks can have origin.

A.2 Exploit Kits

The implementation of these attacks requires different steps that involve the generation of traffic to malicious servers, the implementation of exploits for browsers and plugins and the creation of malware for the specific target. This complex mechanism is now simplified due to the presence of *Exploit Kits* and *Traffic-PPI* services [12], that moved the *Drive-by download* to an *exploit-as-a-service* model.
An *Exploit Kit* is an HTTP server-side application that delivers malware to the victim: it scans the client, identifies vulnerable components, and deploys an executable malware on the victim machine. Grier et al. [12] showed that 47% of the Drive-by attacks lead to an EK.
This malicious artifact presents many functionalities that allow to compromise clients and to hide its presence from malware scanners [25]:

- **Exploit code obfuscation:** the exploits[7] are obfuscated to avoid signature-based detection mechanisms;
- **User agent identification:** the software identifies the browser version, the O.S. and the list of plugins used by the victim;
- **Exploit selection:** the software selects the specific exploit depending on the client configuration;
- **IP cloaking:** the EK changes its behavior depending on the client that interacts with the server. In case a honeyclient is detected, the software behaves as a normal website and does not send any exploit.
- **Mimic genuine website:** the EK can redirect the victim to an innocent website or can provide some web content in case the client is not vulnerable[8];
- **AV detection:** the EK scans IP Blacklists and AV signature repositories to check for its presence.
- **Exploit statistics:** the EK provides to the end-user statistics about the configuration of the victims, the number of clients compromised and the exploits used to infect;

These software artifacts are purchased in the cybercrime underground markets, where it is also possible to access *Traffic-PPI* services. In this case, the malicious user has only to provide the executable malware, the remaining steps of the drive-by attacks are automatically managed by the service.

[7] Typically JS exploits.
[8] This behavior can also be implemented after a successful exploitation.

Appendix B

Formal model of the Attacker

This appendix contains the complete threat model used to describe Drive-by Download mechanisms. Each predicate in the *precondition* of an action is in conjunction with the other predicates; the presence of disjunctions in a rule is used as shorthand to represent different rules for the same action with a shared part in the *precondition*.
Table B.1 is an extension of Table 1.1 and contains the entire list of predicates used in the model. The predicates in Table 1.1 are repeated here to provide a self-contained appendix.

Table B.1: Threat Model Predicates[1]

Predicate	Description
$x \xrightarrow{\text{loc}} cn$	The element $x \in AS \cup IP \cup D \cup NS$ is located in $cn \in Country$
$d \xrightarrow{\text{A}} i$	The element $d \in D$ has address $i \in IP$
$i \xrightarrow{\text{orig}} a$	The element $i \in IP$ belongs to $a \in AS$
$c \xrightarrow{\text{JS}} d$	The element $d \in D$ contains a JS script hosted in the element $c \in D$
$e \xrightarrow{\text{DNS}} d$	The element $e \in NS$ can be queried to resolve the DNS translation of $d \in D$. In other words e is one of the authoritative name servers of d
$a \xrightarrow{\text{RTE(b)}} c$	Given $a, b, c \in AS$, the route from a to c passes through b
$C(x)$	The element $x \in AS \cup IP \cup D \cup Country \cup NS$ is compromised. In case $x \in D \cup NS$, the predicate means that x can be used in a **Drive-By Download** mechanism. This predicate will be called *Globally compromised*
$C(c, d)$	The element $d \in D$ is considered compromised for all the clients that belong to $c \in Country$. This predicate will be called *Country compromised*
$XSS(d)$	The element $d \in D$ is vulnerable to XSS
$CSP(d)$	The element $d \in D$ implements a **secure** Content Security Policy
$UpgradeRequests(d)$	The element $d \in D$ forces to use the HTTPS protocol for all the resource requests. This predicate describes the field `upgrade-insecure-requests` in the CSP
$SRI(d, c)$	The element $d \in D$ implements the Sub-Resource Integrity mitigation for **all** the resources, used by d, stored in $c \in D$. It is assumed $d \neq c$
$IPsec(a, b)$	The packets routed between $a \in AS$ and $b \in AS$ are protected via IPsec
$DNSsec(f)$	The element $f \in NS$ implements DNSsec
$HTTPS(d)$	The element $d \in D$ implements HTTPS
$l\_HTTPS(d, e)$	**All** the JS resources, used by $d \in D$ and hosted in $e \in D$, are retrieved over HTTPS
$HSTS(d)$	The element $d \in D$ implements the header `Strict-Transport-Security`
$Redirect(d)$	The element $d \in D$ redirects HTTP connections to HTTPS. The redirection is either Temporary (Status code 302) or Permanent (Status code 301)
$CT(d)$	The digital certificates, for the element $d \in D$, are signed by CAs that are compliant with the Certificate Transparency
$DANE(d)$	The element $d \in NS$ implements DANE protocol
$I^{\text{DNS}}(d)$	The DNS resolution of the element $d \in D$ is compromised. This predicate will be called *Globally compromised DNS*
$I^{\text{DNS}}(d, e)$	The DNS resolution of the element $d \in D$ is compromised for the clients in $c \in Country$. This predicate will be called *Country compromised DNS*

[1]Some predicates are represented with an arrow notation. The set of predicates vary over the set of $AS, IP, D, Country, NS, CA$.

$I^{\mathrm{R}}(a, c)$	The route between $a, c \in AS$ is compromised
$I\_CA(d)$	The element $d \in D$ is vulnerable to certificate authority attacks
$TLSA\_0(a)$	The element $a \in CA$ is present in the Certificate chain of the TLSA record with certificate usage field 0
$TLSA\_2(a)$	The element $a \in CA$ is present in the Certificate chain of the TLSA record with certificate usage field 2

Using the notation of the Boolean Logic, the symbol $\neg$ in front of a predicate negates the predicate itself.

B.1 Propagation rules

This section describes the propagation rules used in the threat model. Table B.2 connects the rules presented in Section 1.2 of Chapter 1 to the one presented in this appendix. We provide each rule, followed by the intuition of what kind of attack it represents.

Table B.2: Propagation Rules

Rule	Chapter 1	Appendix B
Initially Compromised Nodes		Rules B.1, B.2, B.3
Content Compromise	Rule 1.3	Rule B.4
Third-party JS Injection	Rule 1.4	Rule B.5
DNS Compromise	Rule 1.5	Rule B.6
Route Compromise	Rule 1.6	Rules B.7, B.8
Route to Web Server Compromise	Rule 1.7	Rule B.9
Route to Name Server Compromise	Rule 1.8	Rule B.10
From DNS to Domain Compromise	Rules 1.9, 1.10	Rules B.11, B.12, B.13, B.14
Inline JS Injection	Rule 1.11	Rule B.15
Certificate Compromise	Rules 1.12, 1.13	Rules B.16, B.17, B.18

B.1.1 Initially Compromised Nodes

$$\frac{x \in AS \cup IP \cup NS \cup CA \quad cn \in Country \quad x \xrightarrow{\text{loc}} cn \quad C(cn)}{C(x)} \tag{B.1}$$

`Intuition:` All the autonomous systems, IPs, name servers and certificate authorities associated to a malicious country are under the control of the attacker.

$$\frac{i \in IP \quad a \in AS \quad i \xrightarrow{\text{orig}} a \quad C(a)}{C(i)} \tag{B.2}$$

`Intuition:` All the IPs, that belong to an sutonomous system compromised by the attacker, are considered under the control of the attacker.

$$\frac{i \in IP \quad d \in D \cup NS \quad d \xrightarrow{\text{A}} i \quad C(i)}{C(d)} \tag{B.3}$$

`Intuition:` If a domain (name server) resolves to an IP address under the control of the attacker, then also the domain (name server) is considered compromised.

B.1.2 Content Compromise

$$\frac{d \in D \quad XSS(d) \quad \neg CSP(d)}{C(d)} \tag{B.4}$$

`Intuition:` If a web server is vulnerable to XSS attacks and it does not implement a **secure** Content Security Policy, then the attacker can gain control of the content of the domain.

B.1.3 Third-party JS Injection

$$\frac{d, c \in D \quad c \xrightarrow{\text{JS}} d \quad \neg SRI(d, c) \quad C(c)}{C(d)} \tag{B.5}$$

`Intuition:` If a web server contains a JS resource that is not protected via Subresource Integrity and it is hosted in a domain under the control of the attacker, then the attacker can modify the content of the JS script with malicious code.

B.1.4 DNS Compromise

$$\frac{d \in D \quad e \in NS \quad e \xrightarrow{DNS} d \quad C(e)}{I^{\text{DNS}}(d)} \tag{B.6}$$

`Intuition:` If one of the authoritative name servers of a domain is under the control of the attacker, then the DNS resolution for this domain is considered compromised[2]. An attacker can modify the DNS resolution and map the domain name to a different IP.

B.1.5 Route Compromise

$$\frac{a, b, c \in AS \quad a \neq b \neq c \quad a \xrightarrow{RTE(b)} c \quad \neg IPsec(a, c) \quad C(b)}{I^{\text{R}}(a, c)} \tag{B.7}$$

`Intuition:` If a route from an autonomous system to another AS is not protected via IPsec and it passes through an autonomous system under the control of the attacker, then the route is insecure and the two end-point of the communication could be target of an attack. This rule does not consider the case in which the sender or the destination are compromised, this because the IPsec mitigation cannot protect against this scenario.

$$\frac{a \in AS \quad C(a)}{\forall c \in AS : I^{\text{R}}(a, c)} \tag{B.8}$$

`Intuition:` If the sender AS is under the control of the attacker, then **all** the routes that have origin in this autonomous system are considered insecure. This scenario describes the situation where a country or a provider implements surveillance over its population.

[2]Due to the fact that there are no information about which authoritative NS is queried by a client, this is a simplification implemented in the model. Furthermore, if the attacker is able to compromise one of the authoritative NS for a domain, it is possible that it is also able to compromise the other NSs.

B.1.6 Route to Web Server Compromise

$$\frac{\begin{array}{c} e{\in}Country \quad d{\in}D \quad a,c{\in}AS \quad d \xrightarrow{orig} c \quad a \xrightarrow{loc} e \\ (\neg HTTPS(d) \quad \vee \quad (HTTPS(d) \wedge I\_CA(d)) \quad \vee \quad (HTTPS(d) \wedge (\neg Redirect(d) \vee (Redirect(d) \wedge \neg HSTS(d))))) \quad I^{\mathrm{R}}(a,c) \end{array}}{C(e,d)} \tag{B.9}$$

`Intuition:` If a route between a client and a web server is insecure, then the attacker can implement a MITM attack in the following cases:
This threat model assumes the worst scenario in which a *non-tech-savvy* user accesses the web server via HTTP. It is important to underline that HTTP is the default protocol used by browsers if a protocol is not explicitly defined;

- *Case 1:* If the web server does not implement HTTPS, then the attacker can eavesdrop and replace the content retrieved from the web server;
- *Case 2:* If the web server implements HTTPS but it does not redirect to HTTPS, then, for the hypothesis previously presented, the attacker can eavesdrop and replace the content retrieved from the web server. The HSTS header does not provide any protection if the redirection is not implemented; indeed the header is ignored in an HTTP connection[16];
- *Case 3:* If the web server implements HTTPS and redirects HTTP traffic to HTTPS but it does not implement HSTS, then the attacker can compromise the connection before the redirection phase;
- *Case 4:* If the web server implements HTTPS but it is vulnerable to certificate authority attacks[3], then a malicious CA can forge digital certificates for the domain and use them to authenticate connections to malicious web servers.

The *Case 3* does not take into account the fact that, using a Permanent redirection, the new URI is cached; therefore all the new requests for the resource will be automatically mapped to the new URI [10]. This model requires, in addition, the presence of the `Strict-Transport-Security` header. This choice is due to the fact that the redirection is not a secure mitigation and the HSTS provides a better security with respect to the Permanent redirection:

- HSTS covers the entire domain;
- HSTS implements a preloaded list[4];

For those domains that are not in the preloaded HSTS list, the first access to a web server is still insecure even if all the previous requirements are met.[5] In this model the attacker is not allowed to exploit this vulnerable window to implement a MITM attack; the attacker can compromise the subsequent connections.
The *post condition* of this rule declares that all the connections, that have origin in the country where the sender AS is located, are compromised. This is an upper bound assumption because it is possible that exist ASs, located in the country, that do not present an insecure route. This simplification is due to the fact that there are no information about the location, within the country, of the client that contacts the web server.

B.1.7 Route to Name Server Compromise

$$\frac{f \xrightarrow{DNS} d \quad a,c \in AS \quad a \xrightarrow{loc} e \quad e \in Country \quad f \xrightarrow{orig} c \quad I^{\mathrm{R}}(a,c) \quad \neg DNSsec(f)}{I^{\mathrm{DNS}}(d,e)} \tag{B.10}$$

[3] See Section B.1.10
[4] It is a list of domains that are automatically configured with HSTS. This list is integrated in the browser.
[5] A possible mitigation for this scenario is to increase the number of domains contained in the preloaded HSTS list.

`Intuition:` If a route between a client and a name server is insecure and the NS does not implement the DNSsec protocol, then the attacker can redirect the client to a malicious NS or can implement a DNS cache poisoning attack. As a result the DNS resolution of the queried domain is compromised for all the connections that have origin in the country where the sender AS is located[6].

B.1.8 From DNS to Domain Compromise

$$\frac{I^{\text{DNS}}(d) \quad (\neg HTTPS(d) \vee (HTTPS(d) \wedge I\_CA(d)) \vee (HTTPS(d) \wedge (\neg Redirect(d) \vee (Redirect(d) \wedge \neg HSTS(d)))))}{C(d)} \tag{B.11}$$

`Intuition:` If a web server has a *Globally compromised DNS*[7] and it does not fulfill all the conditions to establish a secure connection[8], then the attacker can redirect **all** the clients to a malicious web server that can claim to be the legitimate one.

$$\frac{I^{\text{DNS}}(d,e) \quad e \in Country \quad (\neg HTTPS(d) \vee (HTTPS(d) \wedge I\_CA(d)) \vee (HTTPS(d) \wedge (\neg Redirect(d) \vee (Redirect(d) \wedge \neg HSTS(d)))))}{C(e,d)} \tag{B.12}$$

`Intuition:` The same situation applies in case the web server has a *Country compromised DNS*; the only difference lies in the *post condition*, where the attacker can redirect the client of the country to a malicious web server.

$$\frac{I^{\text{DNS}}(c) \quad c \xrightarrow{JS} d \quad \neg SRI(d,c) \quad \neg l\_HTTPS(d,c) \quad \neg UpgradeRequests(d)}{C(d)} \tag{B.13}$$

`Intuition:` If a CDN, that provides JS resources for a certain web server, has a *Globally compromised DNS*, then the attacker can redirect the client to a CDN that provides malicious JS resources. This scenario is possible if **all** these conditions are met:

- *The web server does not implement the Subresource integrity mitigation:* in this case the JS resource can be modified with a malicious one.
- *The protocol used to retrieve the resources of the CDN is not HTTPS:* in this case the endpoint CDN is not authenticated.
- *The web server does not implement the* `upgrade-insecure-requests` *field in the CSP:* in this case all the site's insecure URLs are not replaced with secure one.

$$\frac{I^{\text{DNS}}(c,e) \quad e \in Country \quad c \xrightarrow{JS} d \quad \neg SRI(d,c) \quad (\neg l\_HTTPS(d,c) \wedge \neg UpgradeRequests(d))}{C(e,d)} \tag{B.14}$$

`Intuition:` The same situation applies in case the CDN has a *Country compromised DNS*; the *post condition* presents the same structure of rule B.12.
In rules B.13 and B.14 the *post condition* considers the web server and not the CDN as compromised. This choice is due to the fact that the initial connection, that allows the attacker to exploit the vulnerability, starts from the interaction with the web server.

[6] This is the same simplification presented in rule B.1.6.
[7] This means that the attacker has control over the content provided by one of the authoritative NSs for this domain.
[8] A secure connection via HTTPS allows to authenticate the endpoints.

B.1.9 Inline JS Injection

$$\frac{\begin{array}{c} c{\in}Country \quad d2 \xrightarrow{JS} d1 \quad a,b{\in}AS \quad d1,d2{\in}D \quad d2 \xrightarrow{orig} b \quad I^{\mathrm{R}}(a,b) \quad a \xrightarrow{loc} c \\ (\neg HTTPS(d1) \vee (HTTPS(d1) \wedge (\neg Redirect(d1) \vee (Redirect(d1) \wedge \neg HSTS(d1))))) \\ \neg l\_HTTPS(d1,d2) \quad \neg UpgradeRequests(d1) \quad \neg SRI(d1,d2) \end{array}}{C(c,d1)} \tag{B.15}$$

`Intuition:` If the route from a client to a CDN, that provides JS resources to a web server, is insecure[9] and **all** these conditions are met:

- *The web server does not implement the Subresource integrity mitigation:* in this case a MITM attacker can drop the legitimate JS resource and can replace the content with malicious code.
- *The protocol used to retrieve the resources of the CDN is not HTTPS*
- *The web server does not implement the* `upgrade-insecure-requests` *field in the CSP*
- *The web server is not accessible via HTTPS or does not redirect automatically to the secure protocol:* in this case all the requests are not mixed content and therefore are not blocked by the browser.

Then, the attacker can intercept the JS requests and inject malicious JS code.

B.1.10 Certificate Compromise

$$\frac{a \in CA \quad d \in D \quad C(a) \quad e \xrightarrow{DNS} d \quad \neg CT(d) \quad \neg DANE(e)}{I\_CA(d)} \tag{B.16}$$

`Intuition:` If a certificate authority is under the control of the attacker and these conditions are met:

- *the web server's digital certificate are signed by CAs that are not compliant with the Certificate Transparency project.*
- *the authoritative NSs of the domain do not implement the DANE protocol.*

Then, the attacker can forge malicious digital certificates for the domain and use them to generate authenticated connections to malicious web servers.

$$\frac{a \in CA \quad d \in D \quad C(a) \quad e \xrightarrow{DNS} d \quad \neg CT(d) \quad DANE(e) \quad C(e)}{I\_CA(d)} \tag{B.17}$$

`Intuition:` If, in the same scenario of rule B.16, one of the NS is under the control of the attacker, the DANE protocol cannot be trust. For example the attacker can modify the TLSA records and insert a new hash of a digital certificate signed by the compromised CA.

$$\frac{\begin{array}{c} a{\in}CA \quad d{\in}D \quad C(a) \quad e \xrightarrow{DNS} d \\ \neg CT(d) \quad DANE(e) \quad \neg C(e) \quad (TLSA\_0(a) \vee TLSA\_2(a)) \end{array}}{I\_CA(d)} \tag{B.18}$$

`Intuition:` If, in the same scenario of rule B.16, the authoritative NS are not compromised and implement the DANE protocol, the attacker can forge new digital certificates if one of these two conditions is met:

- *the TLSA certificate usage field is* 0 *and the compromised CA is in the Certificate Chain*[10]

[9] This model assumes that the web server does not implement a Proxy to retrieve the resources from the CDN on behalf of clients.

[10] The model assumes that the TLSA record defines the entire chain; this is the most secure approach.

- *the TLSA certificate usage field is* 2 *and the compromised CA is in the Certificate Chain from the Server certificate to the Trust anchor*

Rule B.18 is not implemented in the planner problem because no NS, of the considered domains, implements TLSA record with these specific usage fields.
Even if CT and DANE are considered different technologies [22], in this scenario they can be considered alternative solutions.

Appendix C

Crawlers and Data acquisition

The threat model developed to describe Drive-by download attacks, requires a set of heterogeneous data that range from networking mechanisms to web security and cryptographic solutions. To retrieve all this information it is necessary to implement different crawlers that are able to interact with diverse elements on the Internet.

Some of the information contained in the final dataset is the result of external and related projects. We had access to a list of websites with reflected cross-site scripting vulnerabilities; this information is used to identify, in the dataset, the websites vulnerable to XSS attacks.[1]
A web crawler, developed in previous CISPA projects, is used to obtain all the JS resources dynamically loaded in a web page; the routes among ASes are collected thanks to an existing Python script.

It is now presented a detailed explanation of the crawlers developed to obtain the dataset. All the scripts are written in Python and the extrapolated data are stored in a PostgreSQL database.

C.0.1 Header Crawler

The main goal of the crawler is to collect the security headers send by a web server; an additional functionality allows to detect, in a website, the presence of SRI in static JS resources. To achieve these objects the script makes extensive use of two libraries: *Requests* and *Beautiful Soup*.
The crawler presents two different modes:

- `Single mode:` allows to crawl a specific target;
- `Multiple mode:` allows to crawl a set of websites from a file;

The mode is selected via a command line argument.
The script is composed by a set of functions, each of which analyzes a specific header. The security headers collected are `X-XSS-Protection`, `X-Content-Type-Options`, `X-Frame-Options`, `Strict-Transport-Security` and `Content-Security-Policy`.

In Listing C.1 is presented the procedure used to collect the Content security policy header. The mechanism works as follow: the response header is examined to detect if the CSP header, or a deprecated version[2], is implemented; if the header is found, the value of the CSP fields are collected and the policy is evaluated to discover if inline scripts and eval functions are allowed.

```
1 #this function analyzes the response header to detect CSP fields
2 #the arguments of the function are: the response headers and a dictionary that will
      contains the CSP fields and their values
3 def check_CSP(headers,dictionary):
4     #flag to determine if inline scripts are allowed
```

[1] This value represents a lower bound; it is possible that other domains in the list are vulnerable to a different type of XSS attack.

[2] `X-Content-Security-Policy` and `X-WebKit-CSP`.

```
    allowed_inline = True
    #flag to determine if eval functions are allowed
    allowed_eval = True
    #this flag is used to determine if the field 'script-src' is present or not
    found_script_src = False

    for csp in label_CSP:
        if csp in headers:
            print('FOUND CSP')
            #save the header in a variable and extract the information about the different fields
            header_CSP = headers[csp]
            #the fields extracted are: default-src, script-src, style-src, connect-src, object-src, upgrade-insecure-requests, require-sri-for
            for field in CSP_fields:
                #for each field create an entry in the dictionary
                dictionary[field]={}
                result = re.split(field,header_CSP)
                #if the result of the split has length > 1 means it matched something
                if len(result) > 1:
                    dictionary[field]['Present']=True
                    #access the second element and eliminate the initial space
                    content = re.split(';',result[1].strip())
                    #the value of the field is in the first element
                    content = content[0]
                    #some value in the field can be surronded by '' (e.g. 'self')
                    content = content.replace("'","")
                    #add the content to the Value Key
                    if field == 'require-sri-for' or field == 'upgrade-insecure-requests':
                        #these two fields do not present any content, therefore their value will be set to True
                        dictionary[field]['Value']=str(True)
                    else:
                        dictionary[field]['Value']=content
                    #analyze if inline-script and eval are allowed
                    #Logic for inline script: (NOT(default-src) AND NOT(script-src)) OR (default-src=='unsafe-inline' AND NOT(script-src)) OR (script-src=='unsafe-inline') -> inline script allowed
                    if field == 'script-src':
                        found_script_src = True
                        #this function checks if (script-src=='unsafe-inline')
                        allowed_inline = check_inline_script(content)
                        #this function checks if (script-src=='unsafe-eval')
                        allowed_eval = check_eval(content)
                    #this statement corresponds to (default-src=='unsafe-inline' AND NOT(script-src))
                    if field == 'default-src' and not(found_script_src):
                        allowed_inline = check_inline_script(content)
                        allowed_eval = check_eval(content)
                else:
                    #the field is not present in the CSP
                    dictionary[field]['Present']=False
                    dictionary[field]['Value']='NULL'

            #if a field is not specified in the CSP, it is value is inherit from the default-src content
            #check that the default-src is present
            if dictionary['default-src']['Present']==True:
                for field in CSP_fields:
                    #ignore the field that does not inherit from default-src
                    if field != 'require-sri-for' and field!= 'upgrade-insecure-requests':
                        #check if the field is not present
                        if dictionary[field]['Present']==False:
                            #inherit from default-src
```

```
                dictionary[field]['Value']=dictionary['default-src']['Value']
            #add entry for inline script
            dictionary['inline-script']={}
            dictionary['inline-script']['Present']=True
            dictionary['inline-script']['Value']=str(allowed_inline)
            #add entry for eval
            dictionary['eval']={}
            dictionary['eval']['Present']=True
            dictionary['eval']['Value']=str(allowed_eval)
```

Listing C.1: CSP function

The header crawling phase presented some difficulties during the entire process. Network problems, temporary unavailability and other kinds of errors required to store in log files failed connections, and re-run the process after a certain amount of time. Many domains do not properly redirect connections to their website; in some cases, it was necessary to change the protocol and the URL to access specific resources.

C.0.2 CT Crawler

The list of digital certificates of a domain, stored in the active Certificate Transparency logs, is accessible through a website managed by Google.[3]
CT crawler receives as input a list of domains and contacts the website to retrieve the CAs that signed a digital certificate; due to the fact that the content is dynamically generated, the script uses *Selenium Web driver*, an object-oriented API for web-app testing. The script generates a headless Firefox browser[4], and interacts with the website as a normal client. The entire content of the page is loaded and the HTML code is analyzed to extract the relevant information. Due to the fact that this technique requires to run a browser, the performances are worst with respect to traditional web crawlers; to improve the efficiency, the python script utilizes threats to generate many instances of the headless browser.

C.0.3 Alexa Crawler

Alexa[5] services allow to access website statistics regarding popularity, engagement, audience geography, and visitors. Alexa crawler generates a headless browser to collect the percentage and the value of unique visitors from all the domains in the dataset. This information is a premium feature accessible through a login in the Alexa website; the script simulates a user, inserting the credentials and verifying that the login occurs. Listing C.2 describes the creation of the headless browser and the authentication procedure.

```
    options = Options()
    options.set_headless(headless=True)
    driver = webdriver.Firefox(executable_path=path_firefox,firefox_options=options)
    #load the page
    driver.get(url_login)

    #identify the login input
    email = driver.find_element_by_name('email')
    email.send_keys(my_email)

    password = driver.find_element_by_name('password')
    password.send_keys(my_pw)

    login_attempt = driver.find_element_by_xpath("//*[@type='submit']")
    login_attempt.submit()
```

Listing C.2: Alexa authentication phase

[3] *https://transparencyreport.google.com/https/certificates*
[4] A web browser without GUI.
[5] https://www.alexa.com/

The extraction of the number of unique visitors is done through a specific function presented in Listing C.3. The HTML source code is passed to the function and all the `span` tags are selected. At this point the country is identified through an acronym in the ID of the tag; while the number of visitors is collected from the text part of the `span` tag.
In some cases the number of visitors from a certain country is not available, therefore the script estimates this value from the percentage of visitors assigned to the country and the number of visitors and percentages available from other countries.

```
def getUniqueVisitors(html_code):
    dictionary={}
    html_source = BeautifulSoup(html_code,'html.parser')

    #the unique visitor information is stored in a <span> tag with id=muvnum-* where *
    is the initial of the country
    #e.g. for the USA: 'muvnum-US'
    result_script = html_source.find_all('span', id=re.compile('muvnum-*'))
    for element in result_script:
        #extract the value of the id attribute
        id_value=element.get('id')
        #extract only the dynamic part
        country=re.split('muvnum-',id_value)
        country = country[1]
        #extract the content of the tag span, this will be the unique visitor value
        unique_visitors = element.getText()
        #convert to integer
        unique_visitors = locale.atoi(unique_visitors)
        dictionary[country]=unique_visitors

    return dictionary
```

Listing C.3: Extraction number of visitors

C.0.4 HTTPS Crawler

HTTPS is a protocol used to establish a secure connection on the Internet; over the years an increasing number of domains supported this protocol[6] but, many websites[7] still serve content over unencrypted connections. [20]
The implementation of HTTPS is not enough to prevent attacks from malicious users. It is necessary to redirect all the HTTP traffic to a secure connection and force the user to access the resource over HTTPS in all the subsequent accesses.
The crawler contacts a list of websites to detect if HTTPS is implemented; it also collects the sequence and type of redirections needed to reach the web server. The procedure works as follow:

- **First phase:** the crawler connects to the website through HTTP; it collects the sequence of redirections and analyzes if the web server switches to a secure connection. If this occurs, the redirection code is analyzed to fetch its type. The script inspects the landing URL to detect if the final domain is over HTTPS. Listing C.4 presents a snippet of the code used in this phase.
- **Second phase:** the crawler directly connects to the website through HTTPS. The script examines the response status code to detect if the content is available through the encrypted connection.

```
def get_redirect(url):
    ...
    try:
        resource = 'http://' + url
        response = requests.get(resource,timeout=timeout,headers=headers)
        if response.history:
```

[6] *https://letsencrypt.org/stats/*
[7] Among which some of the top-ranked Alexa domains.

```
            #append the last response to the response history
            response.history.append(response)
            #flags used to detect HTTPS redirection
            flag = False
            triggered = False
            #analyze each redirection (if any)
            for resp in response.history:
                if flag:
                    #check if it redirects to HTTPS
                    if 'https://' in resp.url:
                        if code==301:
                            print('Permanent Redirection to HTTPS')
                            redirection = 301
                        else:
                            print('Temporary Redirection to HTTPS')
                            redirection = 302
                        flag=False
                        triggered=True
                    else:
                        flag=False
                if not triggered:
                    #this is executed only if a HTTPS redirection is not found yet
                    if resp.status_code == 301 or resp.status_code == 302:
                        flag = True
                        code = resp.status_code
                print("{}:{}".format(resp.status_code, resp.url))
                #add to the db
                db.add_record_redirect(table_redirect,id_domain,resp.status_code,resp.
    url)
        else:
            print("Request was not redirected")
        #check if the final url is over HTTPS
        if 'https://' in response.url:
            print('HTTPS')
            db.add_record_HTTPS(table_HTTPS,id_domain,'yes',redirection)
            return
    except (requests.exceptions.ConnectionError, requests.exceptions.Timeout, requests.
    TooManyRedirects):
    ...
```

Listing C.4: HTTPS detection

Detecting if a website is loaded over an insecure connection can be a hard task [19]. For example, *amazonaws.com* correctly redirects insecure connections to HTTPS and changes the final landing domain name; however, if the domain is accessed directly over HTTPS, an error is triggered. Some websites, for example *momoshop.com.tw*, support HTTPS connections but they downgrade to HTTP. Many websites presented a server error status code in a landing URL with the HTTPS protocol; in some cases, this is due to the fact that Web hosting providers support HTTPS connections but the owner of the website does not provide any content on this port.

C.0.5 DNSSec and DANE scripts

DNSsec provides data origin authentication and data integrity to protect the DNS infrastructure against spoofing and MITM attacks.
DNS-based Authentication of Named Entities (DANE) is an alternative to the Certification Authorities mechanism and allows domains to announce, in DNS response secured via DNSsec, information about the PKIX certificates.
To detect if an authoritative NS implements DNSsec and DANE, the script generates DNS queries to evaluate the presence of protocol-specific records. The *dnspython* library is used to create a resolver and craft queries for the target domains.
DANE information is announced through TLSA records; a TLSA record associates a TLS certificate to a specific domain. The record is composed by:

- **Certificate Usage Field:** specifies how to consider the certificate provided in the data field;
- **Selector Field:** specifies which part of the server certificate must match the data field;
- **Matching Type Field:** specifies how the certificate is presented;
- **Certificate Association Data Field:** contains the full certificate or a hash;

```
resolver = dns.resolver.Resolver()
resolver.timeout = 1
resolver.lifetime = 1

try:
    ...
    for domain in domains:
        ...
        try:
            #get id of the domain from the DB
            db.get_ID_Domain(domain)
            id = db.get_result()
            #check if DANE TLSA record is implemented
            try:
                answers = resolver.query('_443._tcp.'+domain, 'TLSA')
                for rdata in answers:
                    db.add_record_DANE(table_DANE,id,str(rdata))
                enable_dane = 'True'
            except Exception as e:
                ...
```

Listing C.5: Query TLSA record

In Listing C.5 is presented a snippet of the script for the analysis of DANE. A resolver is created using the *dnspython* library and a timeout of 1 s is set for the response. The script generates a DNS query to access the TLSA record; due to the fact that a NS can manage a list of TLSA records, the entry is uniquely identified by port, protocol, and domain.
The second part of the script is dedicated to the analysis of DNSsec. The list of authoritative name servers for the domain is collected and each NS is queried by the script. As shown in Listing C.6, the script generates a DNS query for the authoritative NS asking for the DNSKEY records; the status of the response is checked and the answers are examined. DNSsec is considered enable only if both DNSKEY and RRSIG records are present in the answer.

```
            list_ns = query_authoritative_ns(domain,'NS')
            for nsname in list_ns:
                #by default consider DNSsec enabled=False
                enable_dnssec='False'
                try:
                    #get the IP address of the NS
                    response = resolver.query(str(nsname),dns.rdatatype.A)
                    nsaddr = response.rrset[0].to_text()
                    # get DNSKEY
                    request = dns.message.make_query(domain+'.', dns.rdatatype.DNSKEY,
    want_dnssec=True)
                    # send the query
                    response = dns.query.udp(request,nsaddr,timeout=timeout)
                    if response.rcode() != 0:
                        print('No DNSkey')
                    # answer should contain DNSKEY and RRSIG(DNSKEY)
                    answer = response.answer
                    #if the answer does not contain two element then the NS does not
    implement DNSsec
                    if len(answer) != 2:
                        print(domain+": NO DNSsec")
                    else:
                        enable_dnssec = 'True'
```

```
22                print(domain+": DNSsec")
23                ...
```

Listing C.6: Query DNSKEY record

Appendix D

Planner Problem Generator

A planning problem is composed by an initial state, a goal condition and a set of possible actions; the planning aims to provide a sequence of actions that, from the initial state, bring to a state where the goal condition holds. The planning algorithm used in this project is based on the classical planning, where it is assumed that:

1. *The initial state is unique and completely known*
2. *The environment is deterministic, discrete and with a single agent*
3. *Actions are executed sequentially*

D.0.1 The PDDL language

The Planning Domain Definition Language is the standard input language in the planning area. PDDL allows to represent complex problems in a compact way using an *object-style* representation.
The planner input is separated into two files: a *domain* file that contains the predicates, types and action schema, and a *problem* file that contains objects, initial state and goal definition.
Most of the planners translate the input files in a preprocessed intermediate file, where the variables are instantiated with all their possible values; this procedure is called *grounding*. The cost of this precomputation is exponential in operator and predicate arity but leads to more efficient planning.

D.0.2 Fast Downward and Translator file

The planner used in this project is called Fast Downward; a classical planning system based on heuristic search [15]. The translator component of the planner traduces the PDDL file in an alternative representation, generating a file called *Translator*. Due to the fact that this procedure is expensive in terms of memory and time, the planning problem is directly represented in the *Translator* file. Starting from the dataset, the process allows to generate only the actions for which it is possible to fulfill all the requirements, reducing the number of combinations for the planner.

The structure of the *Translator* file[1] is composed by 8 sections, among which the *variable*, the *initial state*, the *goal* and the *operator* section.

Variable section

This section contains the instantiation of the objects described in the threat model, as variables that can assume value True (`Atom`) or False (`NegatedAtom`). The section is optimized to include only the variables for which it is possible to change the value through the planner's actions. For example, the predicate $DNSsec(n)$ with $n \in NS$ is instantiated with all the authoritative NS for which it is possible to set the value from False to True through a sequence of actions.[2] The predicate $XSS(d)$ with $d \in D$

[1] *http://www.fast-downward.org/TranslatorOutputFormat.*

[2] For the NSs with DNSsec already enabled, the variable is not created because in the model developed, planner's actions cannot disable DNSsec mitigation.

it is not instantiated due to the fact that the value is fixed for each domain;[3] this information is directly encoded in the actions definition.[4]
The following example describes the instantiation of the predicate $DANE(c)$ for $c \in NS$ for an authoritative NS of Google.
EXAMPLE:

```
begin_variable
var58
-1
2
NegatedAtom DANE(ns1.google.com)
Atom DANE(ns1.google.com)
end_variable
```

Initial state section

This section contains the definition of the initial value for all the variables defined in the previous section. It is composed by a list of 0 and 1, where the index identifies the variable; 0 corresponds to the value False (`NegatedAtom`) and 1 corresponds to the value True (`Atom`). The following example describes the initial state for the first 4 variables; `var0`,`var1` and `var2` have initial value set to False, `var3` has value True.
EXAMPLE:

```
begin_state
0
0
0
1
...
end_state
```

Goal section

This section contains a list of pairs *(variable,value)* for a subset of the declared variables; the planner reaches the goal state when all the variables present the specified value.
The following example describes a possible goal section; suppose that `var632` corresponds to the instantiation of $C(d)$ for $d \in D$ with the domain *github.com*, then the final goal of the planner is to compromise the Github domain.
EXAMPLE:

```
begin_goal
1
632 1
end_goal
```

Operator section

This section contains the instantiation of the attacker and defender actions for those elements for which it is possible to fulfill all the preconditions.
For example consider the rule B.4, that describes a content compromise attack in the Threat model. It presents two preconditions:

- $XSS(d)$: the domain must be vulnerable to XSS attacks;

[3] A domain is either vulnerable or not depending on the information presents in the dataset. The planner's actions cannot sanitize or insert XSS vulnerabilities in the website.
[4] This is the core of the optimization.

- $\neg CSP(d)$: the domain must not implement a *secure* Content security policy;

This action is generated only for those domains that present XSS vulnerabilities and do not implement a CSP; this is achieved through an analysis of the collected data. For example, in the dataset, the *github.com* domain presents a secure CSP; the B.4 rule will never be executed by the planner and as a result, it is not included in the set of possible actions. Instead, the website of a well-known computer vendor[5] is vulnerable to reflected XSS attacks and does not implement a secure Content security policy; in this case, the action is instantiated. The following example describes how it is represented the instantiated action: suppose `var6` and `var8` are respectively the instantiation of $CSP(d)$ and $C(d)$ for the vulnerable domain, the following code describes the content compromise action for the well-known computer vendor.[6]

EXAMPLE:

```
begin_operator
attack_Content_Compromised_1/1 well-known-computer-vendor
1 #number of preconditions
6 0 #precond:  var6 must be set to 0, i.e., must be False
1 #number of postcond
0 8 0 1 #var8 changes value from 0 to 1, i.e., from False to True
0 #cost of the action
end_operator
```

D.0.3 PDDL generator script

The procedure to generate the *Translator* file from the PostgreSQL database, is implemented through two major files: the *utility* and *PDDLGenerator* scripts. The first script contains a set of functions to access and process data from the database; the second script utilizes these functions to generate the final input for the planner.

Some functions, defined in the *utility* script, present a complex logic that allows to process and extract data from the dataset. Listing D.1 shows the function used to extract valid `Strict-Transport-Security` headers from the collected data.

```
1  def getValidHSTSDomain():
2      try:
3          global db
4          db.get_HSTSDomain()
5
6          #An STS header is correctly formed if:
7          #1)directives (max-age, includeSubDomains) MUST NOT BE repeated
8          #2)directives are separated by a ;
9
10         #HSTS information are stored if max-age parameter != 0
11         #list of tuple of (domain,hsts header) that have an hsts header
12         domains_hsts = db.get_results_columns()
13
14         #extract the list of domain over HTTPS
15         list_domain_https = get_domain_HTTPS()
16         domains = []
17         for domain_hsts in domains_hsts:
18             #if the domain does not implement HTTPS this header is ignored
19             #split the tuple between domain and HSTS header
20             domain = domain_hsts[0]
21             hsts = domain_hsts[1]
22             if not domain in list_domain_https:
23                 continue
24             #RFC 6797: only the first HSTS header MUST be considered
```

[5] We do not reveal the identity for security reasons.

[6] The precondition $XSS(d)$ is embedded in the action definition.

```
            first_hsts = re.split(',', hsts)
            first_hsts = first_hsts[0]

            #control that the directives are not repeted in a single HSTS HEADER
            repeated_max_age = re.split('max-age=', first_hsts)
            repeated_subDomains = re.split('includeSubDomains', first_hsts)
            #if the len is greater than 2 the HSTS header is illegally formatted
            if len(repeated_max_age) > 2 or len(repeated_subDomains) > 2:
                #ignore this domain
                print('Repeated directives in: ' + domain)
                continue

            #the HSTS header is legal
            #extract the directives
            directives = re.split(';', first_hsts)

            #extract max age value if present
            for directive in directives:
                if 'max-age=' in directive:
                    max_age = re.split('max-age=', directive)

                    if len(max_age) > 1:
                        value = int(max_age[1])
                        if value == 0:
                            # ignore the HSTS header
                            print('Max-age value = 0 in: ' + domain + ' -> HSTS is
    ignored')
                            continue
                        else:
                            # this is a valid HSTS header with a max-age greater than 0
                            domains.append(domain)

        return domains
    except Exception as e:
        print(str(e))
```

Listing D.1: HSTS extraction and analysis

The planning problem is specified through a configuration file that defines the initial asset of the attacker and the number of domains to consider. The latter setting influences the complexity of the planning problem and the required amount of time needed to find a solution.
The *PDDLGenerator* script is composed by different parts, one for each section of the *Translator* file. Initially, the reward of the attacker, in terms of number of visitors, is computed for each domain; after that, the script is dedicated to the creation of the *Variable* section. Listing D.2 presents a snippet of the code; the functions *get_variable_Compromised()*, *get_variable_UpdateRequest()* and *get_variable_HSTS()* instantiate a variable with the specific element, following the structure described in Section D.0.2.

```
        #C(x) for x in CDN
        list_CDN = []
        for domain in list_domains:
            #get list of all js retrieved by the domain
            js_domain = utility.get_js_domain(domain)
            #for each of them extract the cdn
            for js in js_domain:
                information = utility.extract_information__js(js)
                #get the CDN
                cdn = information['cdn']
                if cdn != None:
                    #create a Compromised variable for the domain
                    self.get_variable_Compromised(cdn,'domain')
                    list_CDN.append(cdn)


        #C(c) for c in Country
```

```
18          for country in countries:
19              self.get_variable_Compromised(country,'country')
20
21          #C(x) for x in NS
22          list_NS = []
23          #add the NS of the domain
24          for domain in list_domains:
25              ns_domain = utility.get_NS_domain(domain)
26              list_NS+=ns_domain
27
28          #add the NS of the CDN
29          for cdn in list_CDN:
30              ns_cdn = utility.get_NS_CDN(cdn)
31              list_NS+=ns_cdn
32
33          #eliminate duplicates
34          list_NS = list(set(list_NS))
35
36          for ns in list_NS:
37              self.get_variable_Compromised(ns,'ns')
38
39          #UpdateRequests(d) for d in Domain
40          #extract the list of domains that have, in the CSP, the Update Request field
41          domain_with_UpdateRequest =  utility.getUpdateRequestsDomain()
42          for domain in list_domains:
43              if not domain in domain_with_UpdateRequest:
44                  #create the variable
45                  self.get_variable_UpdateRequest(domain)
46
47          #HSTS(d) for d in Domain
48          #extract the list of domains with a correct HSTS header
49          domain_with_HSTS = utility.getValidHSTSDomain()
50          for domain in list_domains:
51              if not domain in domain_with_HSTS:
52                  #create the variable
53                  self.get_variable_HSTS(domain)
```

Listing D.2: Variable section generation

The *Operator* section is implemented using a *create_operator()* function, following the *Translator* specification. Listing D.3 shows the implementation of the rule B.1.7, that describes the route to NS compromise in the Threat model. This rule requires as preconditions:

- $\neg DNSsec(f)$ *with* $f \in NS$: the NS must not implement DNSsec; this is checked at line 12 and added in the precondition list at line 32. The optimization presented in Section D.0.2 is implemented in this code at line 12: for all the NSs that already implement DNSsec, the rule is not created because it will never be executed.
- $I^{\mathrm{R}}(a, c)$ *with* $a, c \in AS$: the route from the AS of the client to the AS of the NS is compromised; this is added in the precondition list at line 34.

The postcondition is define at line 39, where the variable $I^{\mathrm{DNS}}(d, e)$ *with* $d \in D, e \in Country$, that describes DNS resolution compromise of the domain d for the Country e, is set from value False to value True.

```
1           #ROUTE TO NS COMPROMISED RULE
2           print('ROUTE TO NS COMPROMISE')
3           #evaluate NS of Website and CDN
4           final_list = list(set(list_domains+list_CDN))
5           for domain in final_list:
6               if domain in list_domains:
7                   NSs = utility.get_NS_domain(domain)
8               else:
9                   NSs = utility.get_NS_CDN(domain)
10              for ns in NSs:
```

```
                #create the rule only if the NS does not implement DNSsec
                if not ns in list_DNSsec:
                    #get the DNSsec variable
                    id_variable_DNSsec = self.get_variable_DNSsec(ns)
                    #search in which AS they belong to
                    ASs = utility.get_AS_element(ns)
                    for AS in ASs:
                        #get variable I_r(as1,AS) : as1 in list_AS AND as1!=AS
                        for AS1 in list_AS:
                            if AS1!=AS:
                                #get the variable
                                id_variable_route_compromised = self.
    get_variable_Compromised_Route(AS1,AS)
                                #get the country associated with AS1
                                countries_AS = utility.get_country_element(AS1)
                                #check if one of these countries is in the considered
    set
                                for country in countries_AS:
                                    if country in countries:
                                        #create the rule
                                        #precond
                                        precond = []
                                        #no DNSEsec
                                        precond.append((id_variable_DNSsec, 0))
                                        #route must be compromise
                                        precond.append((id_variable_route_compromised,
    1))

                                        #postcond
                                        #the variable will change state from 0 to 1
                                        id_variable_dns = self.
    get_variable_Compromised_DNS_Country(domain,country)
                                        postcond = [(id_variable_dns, 0, 1)]
                                        #create the operator
                                        self.create_operator('
    attack_Route_to_NS_Compromise_1/1 {0} {1} (via NS {2})\n'.format(domain,country,ns)
    ,precond,postcond)
```

Listing D.3: Attacker rule generation

Along with the attacker rules, the *Operator* section contains the defender mitigations. The structure is similar to the one presented in Listing D.3; for example, Listing D.4 contains the code used to create the DANE mitigation for the planning problem.

```
        #Implement DANE
        print('IMPLEMENT DANE')
        considered_NSs = []
        final_list = list(set(list_domains+list_CDN))
        for domain in final_list:
            postcond = []
            precond = []
            name_NS = ""
            if domain in list_domains:
                NSs = utility.get_NS_domain(domain)
            else:
                NSs = utility.get_NS_CDN(domain)
            for ns in NSs:
                if not ns in considered_NSs:
                    considered_NSs.append(ns)
                    if not ns in list_DANE:
                        id_variable_DANE = self.get_variable_DANE(ns)
                        postcond.append((id_variable_DANE, 0, 1))
                        name_NS=name_NS+" "+ns
                        if not ns in list_DNSsec:
                            id_variable_DNSsec = self.get_variable_DNSsec(ns)
                            precond.append((id_variable_DNSsec, 1))
```

```
23
24                 if len(postcond)!=0:
25                     self.create_operator('fix_Implement_DANE {0}\n'.format(name_NS),precond
      ,postcond,cost_DANE)
```

Listing D.4: Defender mitigation generation

Appendix E

Additional Evaluations

To analyze the results of the planner algorithm and to evaluate the correctness of the Pareto frontiers, we compared the results of the mitigation analysis in a certain scenario with tainted versions of the problem.
We proceeded in two main directions:

- **DB taint:** in this scenario, we assume that the DB used for the implementation of the planning problems is tainted. We evaluated two different cases: in the first case, some information regarding the Internet infrastructure is ignored, for example, the information regarding which name servers implement DNSsec; in the second case, some information regarding the Internet infrastructure is overestimated, for example, we assumed that all the name servers implement DNSsec.
- **Action taint:** in this scenario, we assume that the formal threat model is tainted. We evaluated two different cases: in the first case we ignored some mitigations but we allowed the implementation of the attacker's actions that are related to these defender actions; in the second case we also ignored the attacker actions.

E.1 DB Taint

The data collected during the crawling phase provides a representation of the Internet infrastructure that allows to evaluate our threat model in a concrete scenario. We assess the impact of errors in the crawling phase and the influence in the final result of the planner. We focused on the mitigation currently implemented in the Internet[1] and we tainted using two different approaches:

- *Missing information:* we assume that the crawling phase fails to detect right implementations of certain mitigations. For example, the information regarding the implementation of HTTPS in web servers. In this case, we generate a planning problem assuming that all web servers do not implement HTTPS.
- *Overestimated information:* we assume that the crawling phase fails to detect wrong implementations of certain mitigations. For example, the evaluation of a secure CSP can be implemented with some errors that bring the crawler to judge as secure bypassable CSP. In this case, we generate a planning problem assuming that all domains implement a secure CSP.

Fig. E.1 shows the results of the planning algorithm in 3 different cases: the first case represents the *Missing information* taint, where it is assumed that DNSsec and HTTPS are not implemented by any name server and domain; the second case represents the *Overestimated information* taint, where it is assumed that all the domains implement HTTPS and all the related name servers enforce DNSsec. These two scenarios are compared with the result of the planner without any taint in the DB.
The comparison of the Pareto frontiers shows that the inadequacy of the crawling information generates

[1] We did not taint the networking structure of the Internet; for example, we did not modify the information regarding the name servers for a certain domain.

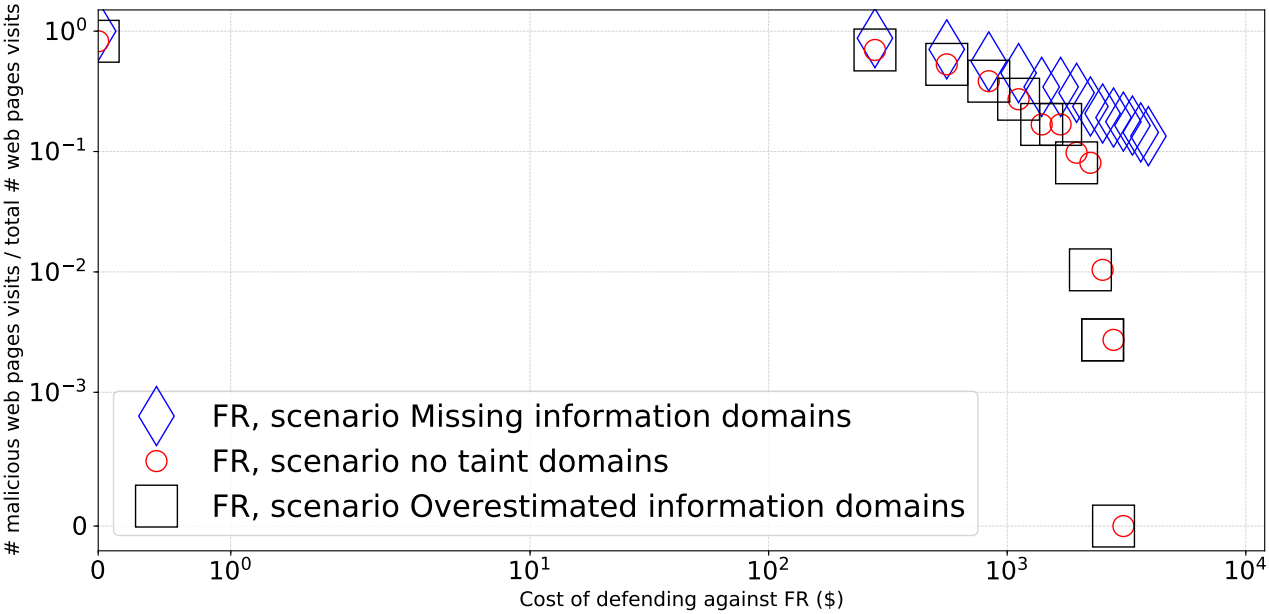

Figure E.1: Pareto frontiers for France with HTTPS and DNSsec DB taint over the top 20 Alexa domains

different defender's strategies. In the *Missing information* case, the result of the planner shows a pessimistic scenario with a higher number of malicious pages accessed and a more expensive cost to mitigate the attacker actions. The *Overestimated information* case presents a result similar to the not tainted scenario, except for the final part of the Pareto, where the number of malicious web pages retrieved by the users decreases faster than the other problem. This situation can be explained due to the fact that the problem ran over the top 20 Alexa domains, where almost all the domains and name servers already implement these mitigations.

E.2 Actions Taint

The results provided by the planning algorithm strictly depend on the formal threat model developed. The description of the Drive-by download mechanisms with this formal representation makes some assumptions regarding the feasible attacks and the possible mitigations. To evaluate the impact of these choices we taint the threat model in two different ways:

- *Mitigation taint:* we assume that some of the mitigations are not effective against certain types of attacks, consequently, they are not implemented in the threat model. For example, we force the planner to not execute HTTPS and SRI mitigations.
- *Mitigation+Action taint:* we assume that some of the rules we developed for our model are worthless; we compute the Pareto frontiers for smaller versions of the threat model without specific attacker and mitigation actions. For example, we do not consider *Inline JS injection* and *Route compromise* attacks.

Fig. E.2 shows the results for the different cases for the top 40 Alexa domains. The Pareto frontier of the formal threat model is compared with two tainted version. In the *Mitigation taint* scenario, the planner algorithm is forced to not execute SRI and HTTPS mitigation actions; the Pareto frontier generated indicates that the attacker reward cannot be reduced to zero without implementing these mitigations. Moreover, the comparison with the normal result of the Pareto frontier shows that the HTTPS and SRI mitigations provide a better defense, given a similar cost, with respect to other countermeasures. The *Mitigation+Action taint* scenario describes a threat model without certain types of attacks; in the specific case the model does not consider *Third-party JS injection* through malicious CDN, *Route to Web Server compromise* through MITM attacks and *From DNS to Domain compromise* through malicious DNS responses for the target domain. The resulting Pareto frontier describes an optimistic

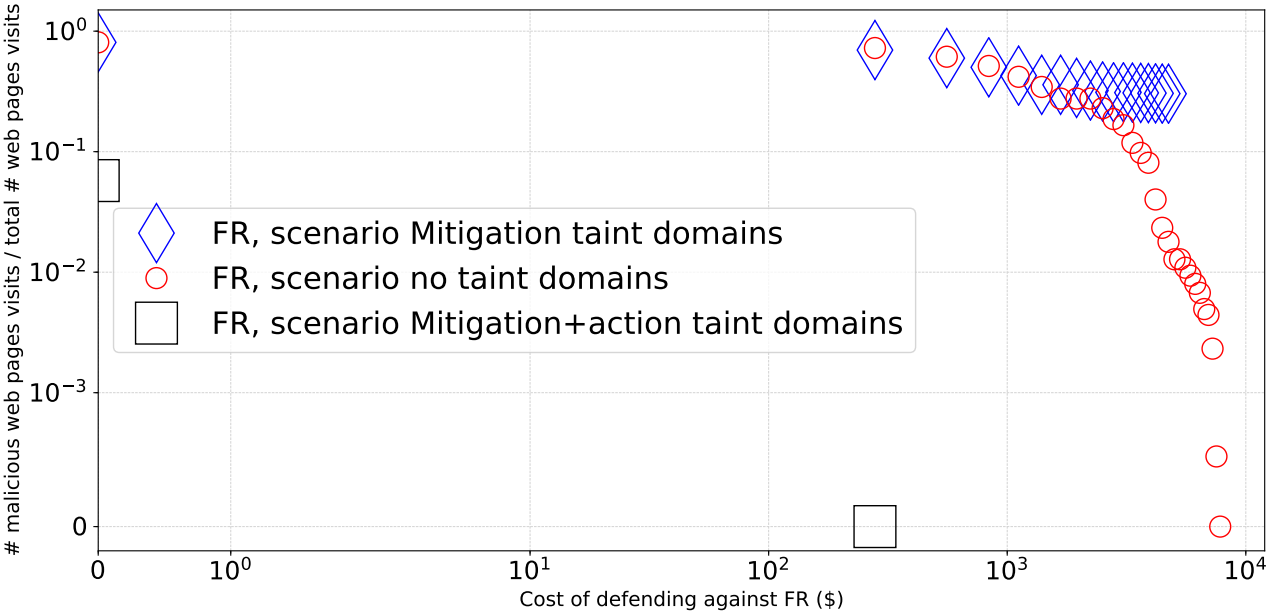

Figure E.2: Pareto frontiers for France with HTTPS and SRI actions taint over the top 40 Alexa domains

scenario where it is assumed that only *Inline JS injection* and *From DNS to Domain compromise* through malicious DNS responses for the domain's CDNs are possible. The number of malicious web pages loaded by the users is smaller with respect to the normal problem, with an underestimation of the attacker's power. The real impact of the attacker is roughly ten times bigger. In this specific scenario, the attacker is only able to implement a DNS poisoning attack to the resolution of a CDN for the domain *jd.com*. The reward can be set to zero through the enforcement of the `upgrade-insecure-requests` field in the CSP.

Bibliography

[1] Towards a comprehensive picture of the great firewall's DNS censorship. In *4th USENIX Workshop on Free and Open Communications on the Internet (FOCI 14)*, San Diego, CA, 2014. USENIX Association.

[2] APNIC. Dnssec validation rate by country, 2018.

[3] R. Arends, R. Austein, M. Larson, D. Massey, and S. Rose. DNS Security Introduction and Requirements. RFC 4033, 2005.

[4] Nick Biasini, Tom Schoellhammer, and Emmanuel Tacheau. Threat spotlight: Spin to win...malware, 2016.

[5] Mike Chapple. How expensive are ipsec vpn setup costs?, 2017.

[6] World Wide Web Consortium. Content security policy level 3, 2016.

[7] World Wide Web Consortium. Mixed content, 2016.

[8] World Wide Web Consortium. Subresource integrity, 2016.

[9] Marco Cova, Christopher Krügel, and Giovanni Vigna. Detection and analysis of drive-by-download attacks and malicious javascript code. In *Proceedings of the 19th International Conference on World Wide Web, WWW 2010, Raleigh, North Carolina, USA, April 26-30, 2010*, pages 281–290, 2010.

[10] R. Fielding and J. Reschke. Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content. RFC 7231 (Proposed Standard), June 2014.

[11] FireEye. Advanced persistent threat groups, 2018.

[12] Chris Grier, Lucas Ballard, Juan Caballero, Neha Chachra, Christian J. Dietrich, Kirill Levchenko, Panayiotis Mavrommatis, Damon McCoy, Antonio Nappa, Andreas Pitsillidis, Niels Provos, M. Zubair Rafique, Moheeb Abu Rajab, Christian Rossow, Kurt Thomas, Vern Paxson, Stefan Savage, and Geoffrey M. Voelker. Manufacturing compromise: the emergence of exploit-as-a-service. In *the ACM Conference on Computer and Communications Security, CCS'12, Raleigh, NC, USA, October 16-18, 2012*, pages 821–832, 2012.

[13] Scott Helme. How widely used are security based http response headers?, 2015.

[14] Scott Helme. Protect your site from cryptojacking with csp + sri, 2018.

[15] Malte Helmert. The fast downward planning system. *J. Artif. Intell. Res.*, 26:191–246, 2006.

[16] J. Hodges, C. Jackson, and A. Barth. HTTP Strict Transport Security (HSTS). RFC 6797 (Proposed Standard), November 2012.

[17] P. Hoffman and J. Schlyter. The DNS-Based Authentication of Named Entities (DANE) Transport Layer Security (TLS) Protocol: TLSA. RFC 6698 (Proposed Standard), August 2012. Updated by RFCs 7218, 7671.

[18] R. Housley. Using Advanced Encryption Standard (AES) CCM Mode with IPsec Encapsulating Security Payload (ESP). RFC 4309 (Proposed Standard), December 2005.

[19] Troy Hunt. Why no https? here's the world's largest websites not redirecting insecure requests to https, 2018.

[20] Troy Hunt and Scott Helme. Why no https?, 2018.

[21] ICANN. Tld dnssec report, 2018.

[22] Google Inc. Comparison with other technologies, 2018. Accessed: 12 Jul 2018 14:33:34 CET.

[23] Joshua Mervine Justin Dorfman. How to implement sri in your build process, 2016.

[24] S. Kent and K. Seo. Security Architecture for the Internet Protocol. RFC 4301 (Proposed Standard), December 2005. Updated by RFCs 6040, 7619.

[25] Vadim Kotov and Fabio Massacci. Anatomy of exploit kits - preliminary analysis of exploit kits as software artefacts. In *Engineering Secure Software and Systems - 5th International Symposium, ESSoS 2013, Paris, France, February 27 - March 1, 2013. Proceedings*, pages 181–196, 2013.

[26] B. Laurie, A. Langley, and E. Kasper. Certificate Transparency. RFC 6962 (Experimental), June 2013.

[27] Bill Marczak, Nicholas Weaver, Jakub Dalek, Roya Ensafi, David Fifield, Sarah McKune, Arn Rey, John Scott-Railton, Ron Deibert, and Vern Paxson. An analysis of china's "great cannon". In *5th USENIX Workshop on Free and Open Communications on the Internet (FOCI 15)*, Washington, D.C., 2015. USENIX Association.

[28] MaxMind. Ip geolocation and online fraud prevention.

[29] Mozilla. Pki:ct, 2014.

[30] Paul Mutton. 95% of https servers vulnerable to trivial mitm attacks, 2016.

[31] Julia Narvaez, Barbara Endicott-Popovsky, Christian Seifert, Chiraag Uday Aval, and Deborah A. Frincke. Drive-by-downloads. In *43rd Hawaii International International Conference on Systems Science (HICSS-43 2010), Proceedings, 5-8 January 2010, Koloa, Kauai, HI, USA*, pages 1–10, 2010.

[32] European Network and Information Security Agency. The cost of dnssec deployment, 2010.

[33] RIPE NNC. Information about specific ip addresses and prefixes.

[34] Eric Osterweil, Burt Kaliski, Matt Larson, and Danny McPherson. Reducing the x . 509 attack surface with dnssec ' s dane. 2012.

[35] Michalis Polychronakis and Niels Provos. Ghost turns zombie: Exploring the life cycle of web-based malware. In *First USENIX Workshop on Large-Scale Exploits and Emergent Threats, LEET '08, San Francisco, CA, USA, April 15, 2008, Proceedings*, 2008.

[36] Niels Provos, Panayiotis Mavrommatis, Moheeb Abu Rajab, and Fabian Monrose. All your iframes point to us. In *Proceedings of the 17th Conference on Security Symposium*, SS'08, pages 1–15, Berkeley, CA, USA, 2008. USENIX Association.

[37] Niels Provos, Dean McNamee, Panayiotis Mavrommatis, Ke Wang, and Nagendra Modadugu. The ghost in the browser analysis of web-based malware. In *Proceedings of the First Conference on First Workshop on Hot Topics in Understanding Botnets*, HotBots'07, pages 4–4, Berkeley, CA, USA, 2007. USENIX Association.

[38] Moheeb Abu Rajab, Lucas Ballard, Nav Jagpal, Panayiotis Mavrommatis, Daisuke Nojiri, Niels Provos, and Ludwig Schmidt. Trends in circumventing web-malware detection. Technical report, 2011.

[39] D. Raumer, S. Gallenmüller, P. Emmerich, L. Märdian, and G. Carle. Efficient serving of vpn endpoints on cots server hardware. In *Cloud Networking*, 2016.

[40] Y. Rekhter, T. Li, and S. Hares. A Border Gateway Protocol 4 (BGP-4). RFC 4271 (Draft Standard), January 2006. Updated by RFCs 6286, 6608, 6793, 7606, 7607, 7705.

[41] E. Rescorla. HTTP Over TLS. RFC 2818 (Informational), 2000.

[42] RIPE Atlas. Internet data collection system. `https://atlas.ripe.net/`, 2017.

[43] Ronald W. Ritchey and Paul Ammann. Using model checking to analyze network vulnerabilities. In *IEEE Symposium on Security and Privacy*, 2000.

[44] B. Schneier. Attack trees. *Dr. Dobbs Journal*, 1999.

[45] Oleg Sheyner, Joshua W. Haines, Somesh Jha, Richard Lippmann, and Jeannette M. Wing. Automated generation and analysis of attack graphs. In *IEEE Symposium on Security and Privacy*, 2002.

[46] C. A. Shue, A. J. Kalafut, and M. Gupta. Abnormally malicious autonomous systems and their internet connectivity. *IEEE/ACM Transactions on Networking*, 20(1):220–230, Feb 2012.

[47] Craig A. Shue, Andrew J. Kalafut, and Minaxi Gupta. Abnormally malicious autonomous systems and their internet connectivity. *IEEE/ACM Trans. Netw.*, 20(1):220–230, 2012.

[48] Milivoj Simeonovski, Giancarlo Pellegrino, Christian Rossow, and Michael Backes. Who controls the internet?: Analyzing global threats using property graph traversals. In *International Conference on World Wide Web*, 2017.

[49] Patrick Speicher, Marcel Steinmetz, Michael Backes, Jörg Hoffmann, and Robert Künnemann. Stackelberg planning: Towards effective leader-follower state space search. In *AAAI'18*, 2018. forthcoming.

[50] Patrick Speicher, Marcel Steinmetz, Robert Künnemann, Milivoj Simeonovski, Giancarlo Pellegrino, Jörg Hoffmann, and Michael Backes. Formally reasoning about the cost and efficacy of securing the email infrastructure. In *2018 IEEE European Symposium on Security and Privacy, EuroS&P 2018, London, United Kingdom, April 24-26, 2018*, pages 77–91, 2018.

[51] Sid Stamm, Brandon Sterne, and Gervase Markham. Reining in the web with content security policy. In *Proceedings of the 19th International Conference on World Wide Web, WWW 2010, Raleigh, North Carolina, USA, April 26-30, 2010*, pages 921–930, 2010.

[52] Steven J. Templeton and Karl E. Levitt. A requires/provides model for computer attacks. In *New Security Paradigms Workshop*, 2000.

[53] Ben Toews. Subresource integrity, 2015.

[54] Lukas Weichselbaum, Michele Spagnuolo, Sebastian Lekies, and Artur Janc. CSP is dead, long live csp! on the insecurity of whitelists and the future of content security policy. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016*, pages 1376–1387, 2016.

[55] Ali Zand, Giovanni Vigna, Richard A. Kemmerer, and Christopher Kruegel. Rippler: Delay injection for service dependency detection. In *2014 IEEE Conference on Computer Communications, INFOCOM 2014, Toronto, Canada, April 27 - May 2, 2014*, pages 2157–2165, 2014.