HAL
open science

New Methods for Targeted Advertising and User Tracking on the Internet

Léo Le Taro

To cite this version:

Léo Le Taro. New Methods for Targeted Advertising and User Tracking on the Internet. Cryptography and Security [cs.CR]. 2015. hal-01167493

HAL Id: hal-01167493

https://inria.hal.science/hal-01167493v1

Submitted on 24 Jun 2015

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Master Thesis Report

New Methods for Targeted Advertising and User Tracking on the Internet

Author

Léo Le Taro
Université Claude Bernard, Lyon 1
CITIlab - INSA de Lyon
INRIA, Privatics Team
E-mail: leo.le-taro@inria.fr

Supervisor

Mathieu Cunche
Maître de Conférences at INSA de Lyon
CITIlab - INSA de Lyon
INRIA, Privatics Team
E-mail: mathieu.cunche@inria.fr

Abstract

Abstract Internet usage is increasing every day. Nowadays, since the advent of smartphones, smart tablets and smart watches, people tend to be permanently online, even in mobility conditions. Free Wi-Fi connectivity is provided in public areas such as parks, coffee shops and airports, and is becoming the norm: people are expecting it.

This trend led free Wi-Fi providers and other network agents to look for ways of monetizing their networks through targeted advertising and user tracking. However, this may be problematic because of the resulting privacy concerns.

In this thesis, we identify the possible ways of carrying out such actions, as well as methods that have been designed by the research community to study their impact.

We then present WALTER, a tool which focuses on detecting content injection in downstream HTTP traffic.

Résumé L'utilisation d'Internet grandit de jour en jour. Aujourd'hui, depuis l'apparition des smartphones, tablettes et montres connectées, les internautes ont tendance à rester en ligne de façon permanente, même en mobilité. Les points d'accès Wi-Fi publics sont de plus en plus présents dans les parcs, les cafés et les aéroports : avoir un accès à Internet gratuitement dans les lieux publics devient la norme.

Cette tendance incite les fournisseurs d'accès et intermédiaires réseau à élaborer des moyens de rentabiliser leurs infrastructures réseaux grâce à la publicité ciblée et le traçage des utilisateurs. Cependant, cela pose des problèmes de confidentialité et de vie privée.

Dans ce rapport, nous identifions les moyens techniques permettant aux intermédiaires réseau de mettre en oeuvre de telles pratiques. Nous analysons également les méthodes qui ont été développées par la communauté scientifique pour étudier leur impact.

Nous présentons ensuite WALTER, un outil dont le but est de détecter l'injection de contenu dans le trafic HTTP descendant.

Acknowledgements

Foremost, I would like to express my sincere gratitude to my supervisor Mathieu Cunche for the continuous support of my research internship, for his patience, motivation, enthusiasm and insightful suggestions. His guidance helped me in all the time of research and writing of this thesis.

Besides my advisor, I would like to thank the CITIlab for having welcomed me warmly during this internship.

My sincere thanks also go to INRIA for believing in my research topic and funding this work.

Finally, I would like to thank the Université Claude Bernard Lyon 1 and the INSA de Lyon for teaching me the material that helped me carry out this research.

Contents

1 Introduction 6
1.1 Motivation 6
1.2 Problem Statement & Research Questions 7
1.3 Thesis Structure 7

2 State of the Art 9
2.1 Upstream HTTP Injection 9
2.1.1 Principle 9
2.1.2 Possible Ways to Exploit & Examples 9
2.1.3 Concerns 11
2.2 Downstream HTTP Injection 12
2.2.1 Principle 12
2.2.2 Possible Ways to Exploit & Examples 12
2.2.3 Concerns 14
2.3 Content Injection by Malicious Extensions 14
2.4 Non-HTTP Injection and Tracking Techniques 15
2.4.1 DNS 15
2.4.2 SMTP 15
2.4.3 SSL/TLS 16
2.5 Focusing Our Work 17

3 Solution Design 19
3.1 Project Architecture 19
3.2 Discrimination Decision Function 20
3.3 Results Logging 21

4 Implementation Details 22
4.1 Technologies Used 22
4.2 Activity Diagram 22
4.3 Logging & Performance 22
4.4 Security 24
4.4.1 Reporting traffic protection 24

4.4.2 DoS mitigation 25

5 Conclusion 26
5.1 Results 26
5.2 Future Work 27

References 28

1

Introduction

Targeted advertising and user tracing on the Internet is nothing new. Since dynamic websites are the norm on the World Wide Web, advertising agencies have been working alongside Web content publishers in order to serve relevant, targeted advertising to visitors.

User tracking has also been invented in an effort to determine and study the behavior, preferences and habits of Internet users. Tracking is also used to monitor statistics such as unique vs. recurring ad impressions, namely for billing purposes.

Although such practices are relatively *old* on the Internet timeline, we observe today new methods for them to be carried out. Multiple users have been reporting strange advertisements appearing on websites known for their cleanliness. Other sites appeared to show ads for their direct competitors, a behavior that content publishers usually avoid. This led us to believe that a new targeted advertising architecture is being employed, driven by Internet Service Providers themselves.

In this paper, we confirm our intuition, we identify the scientific contributions related to this topic, and we present a tool that enables the detection and the quantification of such practices.

1.1 Motivation

Targeted advertising and user tracking has always been a controversial topic regarding privacy. We believe that Net neutrality should be enforced, and that users should have the right to receive untouched information when browsing the Web.

This project provides a way for Internet users to gain insight about the possible alterations that their traffic may be subject to. We further strive to determine the main actors of such alterations.

1.2 Problem Statement & Research Questions

The World Wide Web relies on two main underlying applicative protocols: HTTP and HTTPS.

HTTP is a simple, stateless client-server protocol based on request-response exchanges. In plain HTTP, those exchanges cross the network encrypted. ISPs and intermediary network nodes have access to requests from end-users (clients) to content publishers (servers), as well as responses passing through the opposite direction. Hence, from the point of view of a user browsing the Web *via* HTTP, there is no guarantee of confidentiality nor integrity regarding sent and received information. Censorship, alteration and injection may occur.

HTTPS is a secured protocol consisting of HTTP encapsulated into SSL/TLS. This protocol relies on the Public Key Infrastructure to provide confidentiality, integrity and authentication guarantees. However, HTTP vs. HTTPS usage is still at the content publisher's discretion when one decides to visit a webpage. Although more and more deployed by website owners, one cannot access the whole Web limiting him/herself to using HTTPS only.

Moreover, secondary protocols meant to make Web browsing a better experience, such as DNS, may also be subject to censorship, alteration and injection.

In this thesis, we aim at answering the following questions:

- *What risks do I face if I browse the Web using unencrypted communication channels?*
- *Who actually exploits the current WWW infrastructure to inject ads and tracking material for their own benefit?*
- *Is my Web traffic currently being altered? In which way?*

1.3 Thesis Structure

The rest of this thesis is structured in four main parts:

- The **State of the Art**, which names the different entry points for traffic alteration and injection, and includes a comprehensive study of related scientific contributions,

- The **Solution Design**, which explains the architecture of WALTER, our proposed injection detection solution,
- The **Implementation Details**, which details the technical choices made during WALTER's development,
- The **Conclusion**, which takes a step back from WALTER, discusses its relevance and suggests perspectives for future work.

2

State of the Art

In this paper, we focus on possible content injection by ISPs, notably free Wi-Fi hotspots providers that have high incentives to indirectly monetize their network through advertising [1, 2]. Such techniques are relatively new, and have received little attention by the research community.

This section explores the different entry points for targeted advertising and tracking material on the network and client side, as opposed to server-side techniques that have been extensively studied such as cookie tracking.

2.1 Upstream HTTP Injection

2.1.1 Principle

The basic principle behind upstream HTTP injection is for an ISP or network intermediate to tamper with an HTTP request in order to add identification data before forwarding the request towards its destination.

2.1.2 Possible Ways to Exploit & Examples

Unique identifier

Verizon Wireless has been found to use upstream HTTP injection in order to add a header line, named *X-UIDH*, to all web traffic [3, 4]. Such a header line can be viewed as a supercookie, sent as is to every visited site, regardless of the Same Origin Policy normally enforced by Web browsers when disclosing regular cookie content.

Origin Information

Another way to exploit upstream HTTP injection is to add an *X-Origin* line to the header section of outbound HTTP requests. This line can contain values such as the ISP's identification in order to accurately inform the target website about the ISP from which the request came from.

On their own, website owners, that we will call **publishers**, can try to infer location and ISP from the originating IP address, but such assumptions are known to be unreliable, and can at most provide a city-level accuracy [5].

Now, consider a campus residence implementing free Wi-Fi for its student residents *via* a private ISP such as Wifirst. The ISP could, for example, partner with a publisher, like Amazon, and agree to send them the *X-Origin* header line whenever a student connects to their website, in return for compensation. This way, the publisher could serve targeted advertising with the knowledge that their visitor must be a student connecting from this specific campus. In return, the ISP, which has knowledge of the visitor's traffic, would then charge the publisher based the number of requests forwarded with the *X-Origin* line.

The publisher is more likely to achieve higher conversion rates because his ads are better targeted, and the ISP effectively indirectly monetizes its free-of-charge Wi-Fi network.

Extended Referral Information

The HTTP 1.1 *Referer* header line, defined in RFC 2616 [6], allows visitors to inform the publisher about the page which led them to the page being requested. Implemented in all major browsers, this field poses privacy concerns since it allows publishers to associate browsing patterns with IP addresses or other identification information [6, 7].

However, the *Referer* line itself is limited in the way that it only contains the one previously visited web page. In addition, it will only be sent by browsers when the referred page actually directly leads to the request of the new page. That is to say, no *Referer* line will be sent when a user visiting Google decides to visit Amazon by manually entering Amazon's URL in the address bar, or by clicking on a bookmark.

Yet, the ISP has knowledge of virtually all previous traffic made by the user during his/her session. It is then easy for the ISP to inject a number k of *X-Referer* lines informing the publisher of the user's latest visits (see Fig. 2.1).

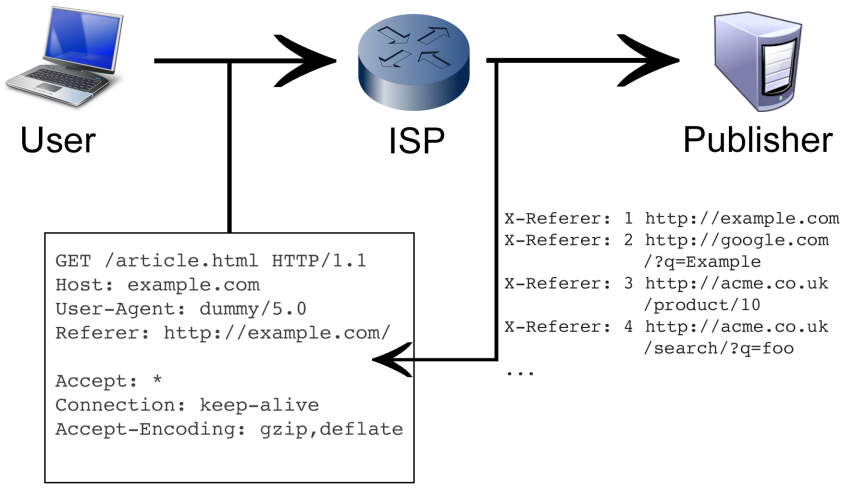

Fig. 2.1: Exploiting HTTP Upstream Injection by sending extended referral information.

To go even further, an ISP that implements identification services through the use of captive portals or WPA/WPA2 802.1X authentication can log a user's traffic coming from various sources, such as computers, smartphones, smart TVs and other connected objects. The ISP is then able to aggregate this info into an exhaustive chronological timeline of previously visited sites, ready to be sent to publishers as in Fig. 2.1.

2.1.3 Concerns

While methods described above enable publishers to serve more relevant advertisements as well as having more accurate insight about their audiences, they also lead to obvious concerns when it comes to user privacy. Although it is true that these techniques can only be directly applied to unencrypted HTTP flows, it is still a problem since most

web pages are accessed via regular HTTP and sensitive information can be leaked to publishers without the user's consent.

2.2 Downstream HTTP Injection

2.2.1 Principle

Downstream HTTP Injection shares the same principle as Upstream HTTP injection seen in 2.1.1, except that it takes place in the opposite direction. Unencrypted HTTP responses, such as HTML pages, images and JavaScript's can be tampered with in order to add or remove specific elements.

Such practices have been studied by Charles Reis *et al.* who demonstrated increased usage and offered a detection and prevention solution called Web Tripwires [8].

2.2.2 Possible Ways to Exploit & Examples

Advertising

The most obvious way an ISP or network agent would want to set up downstream HTTP injection is advertising. This time, the ISP is in charge of carrying out content- and behavior-based targeting, since it holds both the whole response from the publisher and the user's behavioral profile (see Fig. 2.2).

From the ISP's point of view, this is a simpler way to monetize a network infrastructure, because compared to the method described in 2.1.2 and 2.1.2, the ISP only has to hire a single advertising company, instead of having to make agreements with every appealing publisher and website owner. It can even draw advertisers and develop the targeting and serving aspects itself.

Imagine a Wi-Fi-enabled airline company which offers in-flight connectivity to its customers, based on a captive portal or WPA/WPA2 802.1X authentication. With knowledge of each customer's entire flight plan, the company can inject targeted advertising for hotels or car rental companies located at the customer's final destination, even in the case of stopovers.

In addition, several users stated that ISPs not only added advertisements, but also blacklisted their competitors' ads already present on publishers' pages.

Although few rigorous usage studies of this technique can be found in the literature [8], one can find numerous user blogs reporting that downstream HTTP injection is increasingly taking place today [9, 10, 11]. Furthermore, brochures designed to draw advertisers clearly imply that such practices are indeed being used and sold around the world [1, 2].

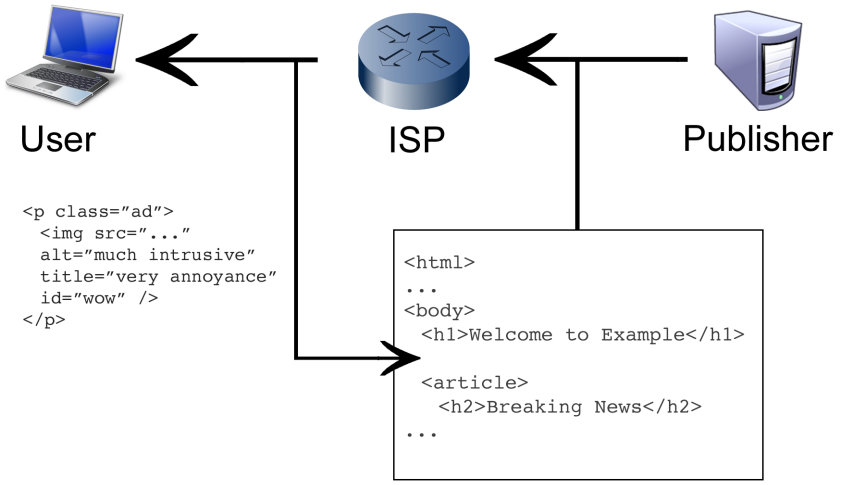

Fig. 2.2: Exploiting HTTP Downstream Injection by adding extra HTML content.

Tracking

It is also possible for an ISP to inject, instead of advertisement material, arbitrary JavaScript code that will be executed in the user's browser's context. Since the user's browser will think the script originates from the actual, "true" website, such code may be used to track changes to the Document Object Model (DOM) without having to run JavaScript engines on the ISP's infrastructure.

Malicious Goals

GitHub, the online community-based source forge, was recently targeted by a massive distributed denial of service attack which crippled it for several days. After investigation, it appears that malicious scripts were injected into unsuspecting users' traffic. Each time a user visited pages that contained Baidu's analytics code, a network intermediary, most likely located in China between the user and Baidu, injected a JavaScript that made the user's browser fetch pages from GitHub. Baidu being China's number one search engine, the number of affected users were huge, thus the attack on GitHub succeeded.

2.2.3 Concerns

Injecting advertisements in pages directly as plain HTML and CSS bypasses browser extensions such as AdBlock Plus, since such extensions rely on URL blacklists, and will typically only block content, like images and external JavaScript, that is to be fetched from a third-party advertiser's website.

Script injection could also monitor user interactions like mouse movements and even keystrokes, and send them over to an arbitrary server using JSONP or other communication means that bypass the Same Origin Policy. Consider a common setup on many websites: a login form pointing at an HTTPS login page, but served within an unencrypted page. With such script injection, the ISP can effectively steal user passwords.

Finally, we have seen thanks to the GitHub example that injection can be used by hackers to obtain a *botnet* of legitimate machines.

2.3 Content Injection by Malicious Extensions

Browser extensions and binaries do not share the same security constraints as in-page JavaScripts: they can access and modify content from any open tab, and send HTTP requests to any server with no Same Origin Policy being enforced [12].

Kurt Thomas *et al.* [13] recently published a survey of malicious extensions. They automatically ran multiple extensions and binaries

found on the most popular web stores inside a sandbox, looking for injection patterns and suspicious permissions. In their work, published in 2015, they report than "more than 5% of unique daily IP addresses accessing Google" were running local code that injected ads into their pages.

Alexandros Kapravelos *et al.* [14] identify malicious extensions by analyzing event handlers as well as luring malware into honey pots.

2.4 Non-HTTP Injection and Tracking Techniques

2.4.1 DNS

DNS plays a crucial role on the Internet since every request to a hostname has to be resolved into the corresponding IP address. Distributed in nature, caching mechanisms have been implemented on ISP DNS servers to limit traffic and improve overall performance for their end users. However, we are witnessing new abusive practices carried out by ISPs (and even public, non-ISP DNS's such as OpenDNS) which consist of replacing *NXDOMAIN* and other error answers by a "fake" *A* or *AAAA* record to redirect users towards an ad-filled page whenever they mistype an URL [15, 16].

Such behavior violates the DNS specification itself, which stipulates [17]:

> A negative answer that resulted from a name error (NXDOMAIN) should be cached such that it can be retrieved and returned in response to another query for the same <QNAME, QCLASS> that resulted in the cached negative response.

2.4.2 SMTP

SMTP is one of the most popular application-layer protocols on the Internet, since its purpose is to route e-mails to their destination. Many ISPs are blocking outbound traffic to TCP port 25, SMTP's default port, to force customers into relaying e-mail through the ISP's own mail servers, when the normal delivery procedure is (i.) the customer's Message Transfer Agent (MTA) queries the recipient's domain's MX

record *via* DNS, (ii.) the customer's MTA opens an SMTP session with the recipient's SMTP server for message delivery.

While this is supposed to fight spam better by preventing compromised user machines from relaying unsolicited messages, such practices can be considered abusive since customers only have, *de facto*, limited Internet access.

ISPs can then engage in e-mail eavesdropping, user tracking and/or ad injection. Because the ISP's mail server is the endpoint of the TCP transmission established by the user, using SMTP over TLS cannot prevent the ISP from reading and/or altering e-mail content without the user's consent.

2.4.3 SSL/TLS

Users have been recently reporting that some ISPs started engaging in issuing forged SSL/TLS certificates to their users [18] (see Fig. 2.3).

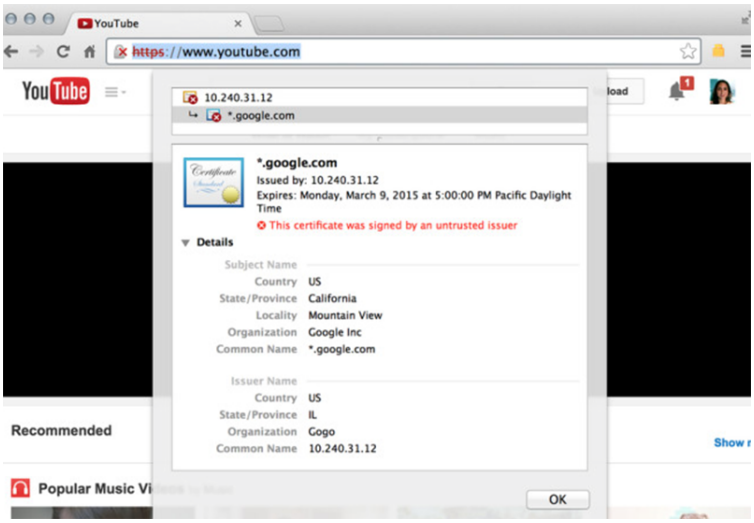

Fig. 2.3: Gogo Inflight Wireless issuing SSL/TLS certificates on behalf of Google. From [18]. Note the issuer's "Common Name" consisting of a *private* IPv4 address.

Such behavior is nothing but a Man-in-the-Middle attack toward their users, as well as an identity theft since they issue certificates for the original site, thus claiming to represent them.

Their motives could be varied: spying on and tracking customers, or injecting unwanted advertisements: all techniques described in this paper become feasible even on SSL/TLS communications.

This type of behavior works because most users are not aware of Internet security problematics. With the increase of open Wi-Fi hotspots throughout the world, often serving captive portals with self-signed or otherwise untrusted SSL/TLS certificates[1], unsuspecting users are accustomed to just "adding an exception" in their browser without inspecting the condition any further.

2.5 Focusing Our Work

In this internship, we decided to focus on downstream HTTP injection. Our tool, WALTER, extends the features of Web Tripwires [8][2] by giving the users extensive and visual feedback over the altered areas of the page. We are collecting, at the time of writing, updated data which will allow us to identify and rank the ISPs that alter their customers' traffic. We use a dummy webpage featuring fake, lure ads ready to be replaced. We assume ad injection does not rely on specific URL patterns, but occurs on every browsed page.

Content injection by extensions has already been studied in-depth by researchers [13, 19, 14]. Such a technique requires manual installation by users. Hence, we did not mean to focus on extensions. However, WALTER is still able to discriminate network-layer alteration from local alteration, which allows us to better classify our results.

In this paper, we assume than network intermediaries are not issuing fake SSL certificates, and we consequently make the assumption that HTTPS exchanges between the client and WALTER were not subject to a *man-in-the-middle* attack.

[1] as one can witness even at Lyon 1 and the INSA.
[2] whose website is currently down

In a nutshell, we aim at spreading awareness: we recommend that publishers make their sites available over HTTPS, and we urge users to pay more attention to certificate-warning messages.

3

Solution Design

The main goal of this project, which we named WALTER[1], was to design a user-friendly, simple to use tool that did not require prior installation by the client to function. Besides, we wanted to make sure that results would be logged so that we could have feedback over 100% of the client-initiated checks for alteration.

This is why we chose to develop a self-testing webpage, which can be accessed by any desktop or mobile browser that can run JavaScript. We thus chose to deal with browsers' security policies rather than developing fat client apps that require setup.

3.1 Project Architecture

The architecture of WALTER is detailed in the following sitemap:

/index.html Welcome page explaining WALTER's concept.

/about-us.html Description of the INRIA Privatics team and contact info.

/check.html Generates a dummy page along with client-side checking code. Logs the whole response before returning it to the client.

- **/check/doCheck.html** Collects */check.html*'s DOM as viewed by the client's browser. Compares this DOM with that of the logged response.
- **/check/verdict.html** Displays the check result, whether alteration are network-based or local, and a side-by-side view of both DOMs where applicable.
  - **/check/verdict/render.html** Shows a visual rendering of differences, if applicable.

[1] Is my Web Traffic Altered?

3.2 Discrimination Decision Function

In order to determine the cause of alterations, i.e. network or local, WALTER uses the decision pattern illustrated in Fig. 3.1.

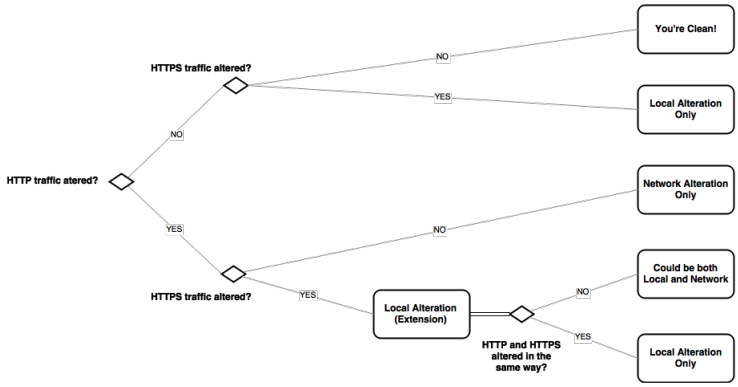

Fig. 3.1: WALTER's Alteration Cause Decision Algorithm.

When we detect that the HTTPS traffic was altered, we are positive that a browser extension or local code is tampering with the client's traffic. However, as we can observe in Fig. 3.1, when alteration occurs on both HTTP and HTTPS checks in different ways, we cannot determine whether network-level alteration occured.

For example, imagine a situation where an extension carries out an alteration E of both HTTP and HTTPS traffic, and a network agent conducts an alteration I of HTTP traffic only. Upon running WALTER, the client would report that both HTTP and HTTPS were altered in a different way (resp. $HTTP_{received} = E(I(HTTPS_{sent}))$ and $HTTPS_{received} = E(HTTPS_{sent})$).

However, the AdBlock Plus extension will add a random *class* HTML attribute to all ad-resembling elements found on a given page [20]. Because of the randomness of this class name, our dummy page sent over HTTP would be altered by AdBlock Plus in a technically different way than that sent over HTTPS. In such a case, no network agent is guilty of alteration, but the client would still report both HTTP and HTTPS altered in a different way.

When such cases are detected by WALTER, the client is shown an explanative message inviting him/her to disable extensions. Two side-by-side difference views are shown: expected vs. HTTP, and expected vs. HTTPS.

3.3 Results Logging

After each check, WALTER logs results into a database. The collected data is structured as follows:

id Unique identifier,
os Client operating system,
browser Client browser,
verdict_regular Result of the HTTP check: clean or altered,
verdict_https Result of the HTTPS check: clean or altered,
ip Client IP address,
hostname Client hostname,
expected_source HTML source code sent by WALTER to the client (both checks),
received_source_regular HTML source code interpreted by the client's browser (HTTP check),
received_source_https HTML source code interpreted by the client's browser (HTTPS check),
traceroute Results of a traceroute from WALTER to the client.

When a client's traffic is detected altered by a network agent, his/her direct ISP may or may not be responsible: any network intermediary between WALTER and the client may be guilty. For this reason, we keep the entire traceroute from the client to WALTER, in order to later infer and rank altering network agents by cross-analysis.

4

Implementation Details

4.1 Technologies Used

WALTER is a Web application developed in object-oriented PHP5 with the Symfony2 framework. The client-side logic is coded in JavaScript with jQuery v2.1.3, while the user interface uses HTML5 and CSS3 with the Bootstrap v3.3.2 framework.

WALTER can be easily deployed on any HTTP and HTTPS enabled web server with command-line access by following the Symfony2 deployment procedure. It is currently running on a CentOS 6.6 virtual server running Apache v2.2.15, and MySQL v5.1.73 as the database management system.

WALTER's command-line interface tools help administrators to automatically execute pending traceroutes (see 4.3), (re)generate RSA keypairs (see 4.4.1) and clear the application cache after an update. On our production system, such tasks are scheduled thanks to *cron* jobs.

4.2 Activity Diagram

The detailed activity diagram, explaining state shifts and network exchanges, is shown in Fig. 4.1.

4.3 Logging & Performance

We chose to save complete source codes into the database instead of just *diff* results because we want to be able to use other comparison methods in the future, and try to infer keyword-based patterns that may lead to alteration.

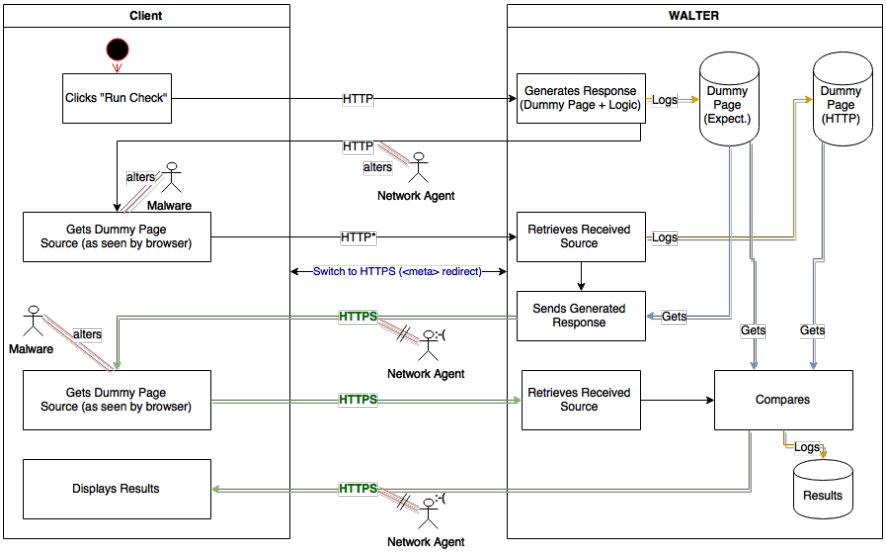

Fig. 4.1: WALTER's Activity Diagram.

The *expected_source* field is kept for each single check in order to increase flexibility: we trade storage space for the ability to change our dummy page's content anytime without compromising our previously logged results.
Note that only in the event of an alteration will the appropriate received source codes be logged. That is to say, a client which receives an altered version of the page over HTTP but the clean version over HTTPS will only have his/her HTTPS source code field set to *NULL* in the database in order to save space.
The expected source code will always be logged, whatever the check results, in order to make future guesses about contents that trigger alteration.

Large database text fields like source codes and traceroutes are stored gzipped to minimize storage space usage. For indexing and lookup speed purposes, duplication is used on client IP addresses and hostnames (visible in traceroutes) and check results (recoverable by running *diff*s on source codes).

Traceroutes are network-intensive tasks that usually take several seconds to complete. To increase client-side responsiveness, we do not perform traceroutes synchronously; instead, we set the appropriate

database field to NULL at the time of request, and use a separate task which executes the missing traceroutes periodically.
Note that even though the delay between client request and traceroute does not exceed one hour, the route may change in that time interval. Keeping this in mind, we still believe our results will be accurate enough on a large scale.

4.4 Security

4.4.1 Reporting traffic protection

WALTER works by receiving HTML source codes from clients and comparing them with its expected source code. To prevent network agents from tampering with this reporting traffic, we had to use encryption techniques.

Using HTTPS was excluded because of the Same Origin Policy enforced by browsers: a page fetched over HTTP cannot perform AJAX calls to HTTPS locations and *vice-versa*. At this point, we are using our own cryptographic solution relying on the RC4 stream cipher, fed with a randomly-generated symmetric key exchanged over RSA. The cryptographic workflow is described below:

1. The server generates (and periodically updates) an RSA keypair (rsa_pri, rsa_pub).
2. Upon receiving a checking request, WALTER sends rsa_pub to the client along with the dummy page and client-side JavaScript logic.
3. The client generates an RC4 key $rc4key$.
4. The client encrypts the source code $enc_sc = RC4(sc, rc4key)$.
5. The client encrypts $enc_rc4key = RSA_encrypt(rc4key, rsa_pub)$.
6. The client sends an AJAX call to WALTER containing (enc_sc,-enc_rc4key).
7. The server decrypts $rc4key = RSA_decrypt(enc_rc4key, rsa_pri)$.
8. The server decrypts the source code $sc = RC4(enc_sc, rc4key)$.

Such a method guarantees integrity and confidentiality, but lacks client authentication. An eavesdropper, having access to rsa_pub, cannot pre-determine the $rc4key$ that the legitimate client will generate. However, it can block traffic from the client to WALTER, gen-

erate its own $rc4key$, encrypt it with rsa_pub and return a forged (enc_sc, enc_rc4key) to WALTER on behalf of the legitimate client.

4.4.2 DoS mitigation

Each checks leads to heavy network tasks (traceroutes, reverse DNS lookups), computationally-intensive tasks (asymmetric and symmetric cryptography) and database insertions. In order to mitigate potential denial-of-service attacks, we decided to set a limit of one check every 20 seconds (configurable) per user.

Since WALTER heavily relies on the HTTP session to perform checks, we chose to enforce this limitation by storing a timestamp along with the client IP address into the session at the beginning of the checking process. Any subsequent request made during the next 20 seconds will be denied.

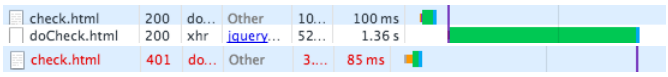

Fig. 4.2: WALTER DoS Protection:
Accepted (Black) vs. Filtered (Red) Request Times.

5

Conclusion

5.1 Results

In this thesis, inspired by Web Tripwires [8], we designed and implemented an integrated application to tell users about the integrity of their web traffic, as well as collect experimental data that shall allow us to identify network agents which tamper with customers' traffic.

We are successfully discriminating network-level alteration from local alteration (e.g. browser extensions) (See Fig. 5.1).

```
89 </div>
90 <p>
91
   <img src="/bundles/app/img/ad_img.png"
   class="dummy" alt="If You Can Read This, Disable
   AdBlock!" style="display: none;">
92 </p>
93 </aside>
```

```
91 </div>
92 <p>
91 <img src="/bundles/app/img/ad_img.png"
   class="dummy" alt="If You Can Read This, Disable
   AdBlock!" style="display: none !important;
   visibility: hidden !important; opacity: 0
   !important; background-position: 0px 0px;"
   width="0" height="0">
94 </p>
95 </aside>
```

Fig. 5.1: AdBlock's Alteration Detected by WALTER.

WALTER's architecture makes it really user-friendly (a simple, cross-browser website); however, it has a fixed URL and will not detect network agents carrying out injection on a per-URL basis.

We are, at the time of writing, collecting updated data that will help us better understand some of the new downstream injection threats presented in 2.2. Unfortunately, we do not have enough reports to build a representative sample, and are hoping for that purpose that WALTER will go viral.

You can visit our deployed WALTER installation at `http://walter-experiment.inria.fr` and help us spread awareness. May your check results come out clean!

5.2 Future Work

Downstream HTTP injection provides wide perspectives for future work, here are the most obvious and relevant ones that we have identified during this project:

- Upon gathering a representative sample of check results, develop a data analysis tool that will efficiently parse the traceroutes and infer the most altering network agents.
- Dynamically change the dummy page to sometimes add content that may trigger injection (keywords, presence of ads, links, videos...). Develop an data analysis tool to identify which content triggers injection.
- Develop a browser extension or binary to gather more relevant data about per-URL, upstream+downstream injection.

References

1. Ragapa LLC. Hotspot Monetization Datasheet, 2014. [Online; accessed June 5, 2015] `http://s3.amazonaws.com/ragapa-static-content/website/literature/ragapa_monetization_datasheet.pdf`. 2, 2.2.2
2. Gogo Inflight Internet. Partner with gogo, 2014. [Online; accessed June 5, 2015] `https://www.gogoair.com/gogo/partner.do`. 2, 2.2.2
3. Verizon. Recent changes to the policy, 2014. [Online; accessed June 5, 2015] `https://www.verizon.com/about/privacy/recent/`. 2.1.2
4. Slashdot. Verizon injects unique ids into http traffic, 2014. [Online; accessed June 6, 2015] `http://yro.slashdot.org/story/14/10/24/2052218/verizon-injects-unique-ids-into-http-traffic`. 2.1.2
5. Ingmar Poese, Steve Uhlig, Mohamed Ali Kaafar, Benoit Donnet, and Bamba Gueye. IP geolocation databases: Unreliable? *ACM SIGCOMM Computer Communication Review*, 41(2):53–56, 2011. 2.1.2
6. Roy Fielding, Jim Gettys, Jeffrey Mogul, Henrik Frystyk, Larry Masinter, Paul Leach, and Tim Berners-Lee. Hypertext transfer protocol–HTTP/1.1, 1999. 2.1.2
7. Balachander Krishnamurthy, Konstantin Naryshkin, and Craig Wills. Privacy leakage vs. protection measures: the growing disconnect. In *Proceedings of the Web*, volume 2, pages 1–10, 2011. 2.1.2
8. Charles Reis, Steven D Gribble, Tadayoshi Kohno, and Nicholas C Weaver. Detecting in-flight page changes with web tripwires. In *NSDI*, volume 8, pages 31–44, 2008. 2.2.1, 2.2.2, 2.5, 5.1
9. Zachary Henkel. ISP advertisement injection - CMA communications, 2013. [Online; accessed June 3, 2015] `http://zmhenkel.blogspot.fr/2013/03/isp-advertisement-injection-cma.html`. 2.2.2
10. StackExchange. My ISP is injecting strange codes to every website i visit, 2014. [Online; accessed June 4, 2015] `http://security.stackexchange.com/questions/70970/my-isp-is-injecting-strange-codes-to-every-website-i-visit`. 2.2.2
11. Techdirt. Mediacom puts its own ads on other websites, including Google & Apple, 2011. [Online; accessed June 4, 2015] `https://www.techdirt.com/articles/20110228/11332813302/mediacom-puts-its-own-ads-other-websites-including-google-apple.shtml`. 2.2.2
12. Stefan Heule, Devon Rifkin, Alejandro Russo, and Deian Stefan. The most dangerous code in the browser. In *15th Workshop on Hot Topics in Operating Systems (HotOS XV)*. USENIX Association, 2015. 2.3

13. Kurt Thomas, Elie Bursztein, Chris Grier, Grant Ho, Nav Jagpal, Alexandros Kapravelos, Damon McCoy, Antonio Nappa, Vern Paxson, Paul Pearce, et al. Ad injection at scale: Assessing deceptive advertisement modifications. 2015. 2.3, 2.5
14. Alexandros Kapravelos, Chris Grier, Neha Chachra, Christopher Kruegel, Giovanni Vigna, and Vern Paxson. Hulk: Eliciting malicious behavior in browser extensions. In *Proceedings of the 23rd Usenix Security Symposium*, 2014. 2.3, 2.5
15. Nicholas Weaver, Christian Kreibich, and Vern Paxson. Redirecting DNS for ads and profit. In *USENIX Workshop on Free and Open Communications on the Internet (FOCI), San Francisco, CA, USA (August 2011)*, 2011. 2.4.1
16. David Dagon, Niels Provos, Christopher P Lee, and Wenke Lee. Corrupted DNS resolution paths: The rise of a malicious resolution authority. In *NDSS*, 2008. 2.4.1
17. IETF. RFC2308, 1998. [Online; accessed June 3, 2015] `https://tools.ietf.org/html/rfc2308`. 2.4.1
18. Steven Johns. Gogo inflight internet is intentionally issuing fake SSL certificates, 2015. [Online; accessed June 4, 2015] `http://www.neowin.net/news/gogo-inflight-internet-is-intentionally-issuing-fake-ssl-certificates`. 2.4.3, 2.3
19. Trevor Jim, Nikhil Swamy, and Michael Hicks. Defeating script injection attacks with browser-enforced embedded policies. In *Proceedings of the 16th international conference on World Wide Web*, pages 601–610. ACM, 2007. 2.5
20. AdBlock Plus. Adblock Plus internals, 2015. [Online; accessed June 9, 2015] `https://adblockplus.org/faq_internal`. 3.2