

Securing Online Advertising

Nevena Vratonjic, Julien Freudiger,
Jean-Pierre Hubaux

School of Computer and Communication Sciences
EPFL, Switzerland
firstname.lastname@epfl.ch

Mark Felegyhazi

University of California, Berkeley, USA
felegyha@eecs.berkeley.edu

ABSTRACT

Online advertisement is a major source of revenues in the Internet. In this paper, we identify a number of vulnerabilities of current ad serving systems. We describe how an adversary can exploit these vulnerabilities to divert part of the ad revenue stream for its own benefit. We propose a scalable, secure ad serving scheme to fix this problem. We also explain why the deployment of this solution would benefit the Web browsing security in general.

1. INTRODUCTION

Over the last few years, online/Internet advertising has generated record revenues, reaching nearly \$21 Billion in 2007 in the USA [9], and \$41 Billion worldwide [1]. The Internet is recognized as an important distribution channel for advertisements that emerged as the main source of revenues for most online activities. Several companies are thus trying to get bigger share of the revenues: Google acquired DoubleClick [8], Microsoft is still attempting to acquire Yahoo [7], companies propose new ad platforms for ISPs to start embedding ads [1], and wireless networks sponsoring their deployment from online advertising revenues [2].

Surprisingly, online advertising and Web browsing still rely on the HTTP protocol, which does not provide any guarantees on the integrity nor the authenticity of online content. Given the money at stake and the lack of security protocols, an adversary might perform sophisticated Web traffic modifications, aiming at exploiting the online ad serving system to increase its own revenues. Thus, the on-the-fly modification of Web traffic containing advertisement is turning into one of the main concerns over the future of the Internet. Previous work suggests the use of Javascript, together with digests of Web pages (called “known-good representations”) [19] transmitted in parallel, to check whether a Web page has been modified by the network. Nevertheless, such a solution can potentially be bypassed, as it is not protected by a computationally hard problem. Other work focused on the robustness against click fraud attacks [16, 14, 11]: attacks targeting the advertisement mechanism itself by creating fake interest for a product to artificially increase the revenue of a website.

Considering the amount of money at stake, the security of online advertisement is becoming a pressing concern for advertisers. As we will show, the deployment of secure online

advertisement systems could actually reinforce Web browsing security. Indeed, not only would advertisers benefit from a secured online advertisement system; so would websites as they share part of the revenue, ad servers as they manage the ad infrastructure, and users as their browsing activities would be secured. Online advertisement might thus fuel Web browsing security.

To the best of our knowledge, this paper is the first to investigate the strategic modification of Web pages aimed at the exploitation of online advertising, and to suggest a secure ad delivery system. Note that the devised solution does not force browsers to display ads, but simply secures the delivery of the legitimate advertisement. Our contributions can be summarized as follows: (i) We present the problem of securing online advertising and propose a model of online ad serving systems. (ii) We present several attacks on ad serving systems. (iii) We propose a scalable secure online ad serving scheme that fits in today’s Internet. As a byproduct, we show that online advertisement enables the successful establishment of Web browsing security.

2. ADVERTISING ON THE INTERNET

Internet advertisement is generally constituted of a short text, an image, or an animation embedded into a Web page and linking to an advertised website. The purpose of an ad is to generate traffic to the advertised website and, consequently, to increase the revenue for the advertised products or services.

2.1 Advertisement Serving Architecture

For most of the websites users visit, a number of advertisements appear together with the content of a Web page. Ads are embedded into Web pages either through an ad serving system, or by websites themselves. We suggest a simple model of the Internet advertisement serving architecture, depicted in Figure 1.

The Internet attracts *Advertisers* (AV) to sell and advertise their products and services online. Ads are stored at *Ad Servers* (AS), which belong to an ad network. Throughout the rest of the paper we use the terms ad network and ad server (for the ad server that belong to the ad network) interchangeably. The ad network offers the service of embedding ads into Web pages. Ad networks have contracts with AVs to choose the proper *Websites* (WS) to host AVs’ ads and dis-

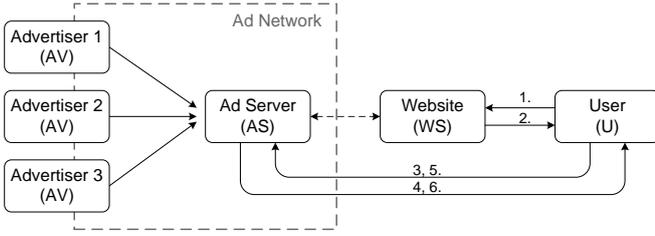


Figure 1: The advertisement serving architecture. WS and AS have a contractual agreement (dashed arrow) that lets AS add advertisement to WS’s Web pages. The protocol illustrated with arrow numbers is given below.

play them to *Users* (U). According to the terms of a contract, an AV pays money to an AS whenever an ad has generated traffic on the AV’s website. The AS gives a percentage of that money to the WS hosting the ad. The money flow explains the incentive to participate in the ad serving system: the AV earns the revenue created by ads, the AS earns money for storing the ads and finding a proper WS on whose Web pages ads will be displayed, and the WS earns money for hosting ads and directing the U towards advertised WS.

The most popular ad serving technique is to direct a user’s browser to fetch advertisements directly from ad servers. For the rest of the paper, we will use terms user and user’s browser interchangeably. WSs define ad frames (f) in their Web pages (p) to be filled with ads, and direct browsers from where to download them. After downloading a Web page, a browser automatically fetches ads from ad servers and embeds them into appropriate ad frames. The protocol can be modeled as follows:

1. $U \rightarrow WS$: `GET URLWS HTTP`
2. $WS \rightarrow U$: p
3. $U \rightarrow AS$: `GET URLAS, WSID HTTP`
4. $AS \rightarrow U$: `display.js`
5. $U \rightarrow AS$: `GET URLAS, WSID, CAS HTTP`
6. $AS \rightarrow U$: `ads`

With the first two messages (Figure 1), U fetches a Web page p from a WS, identified with URL_{WS} . Web page p contains an ad frame f with a Javascript directing U towards the ad server AS. The Javascript includes the URL of the ad server (URL_{AS}) and a unique identifier WS_{ID} of the WS, required by the AS to properly transfer potential advertisement revenue to the WS. With the next two messages, U fetches a javascript `display.js` from the AS. This script executes locally on the user’s machine, and looks for AS’s cookies C_{AS} that might have been deposited during a previous interaction with U. If they exist, the cookies uniquely identify the user and enable the profiling of his Web browsing preferences. With the last two messages the script fetches ads from the AS. The browser merges the advertisement with other, previously downloaded elements of p .

This approach has several advantages: (i) the HTML code that directs users to fetch advertisements is simple and easy to maintain, as only one line of code (a reference to the Javascript) should be added in the ad frames, (ii) it is scalable, as the workload is distributed to the users, (iii) it allows ad servers and advertisers to keep the control as ads

are stored and maintained at their servers. There are several drawbacks as well: Because users must fetch ads from different servers, the ad serving technology slows down the display of Web pages, consumes extra bandwidth and challenges the privacy of users [17].

There are other online advertisement serving techniques. For example, a PHP script running on a Website can locally embed advertisements and serve them to users together with the content of a Web page. This technique is not very popular because it puts more workload on the Web servers compared to the previous approach, thus it does not scale well.

2.2 Ad Servers

A notable difference between online ad serving and traditional ad serving (e.g., television, radio...) is that online ads can be targeted to individual user’s interests. Ads that will appear on a user’s screen are selected by ad servers in real time such that they match user’s interests at the moment. Typically, ads are targeted to the content of the Web page hosting the ad, user’s interests extracted from previous browsing history, user’s geographical location, etc. The goal of targeted advertising is to increase the probability of users actually being interested in the advertised products. At the ad server, user’s interests can be expressed with one or more *keywords*. The ad server stores a number of ads associated with each keyword and it runs auction algorithms to select the most profitable ads and the order in which they should appear on the Web page. The selection is done such that the profit is maximized for both advertisers and websites hosting the ads.

There are two main pricing models according to which a WS receives money for hosting ads. In the *Pay-Per-Click* model, a WS receives money when a user clicks on one of the ads it hosts. In the alternative, *Pay-Per-Impression* model, a website receives money for each ad that is displayed on a user’s screen together with the WS’s content.

3. THREATS

In this section, we define the adversary model considered through the rest of the paper and we identify a number of possible attacks on ad serving.

3.1 Adversary model

We consider both a selfish adversary intending to take advantage of the ad serving system and a malicious adversary intending to harm it. The ad serving system can be exploited by one of the participating entities (Figure 1), or by an access network that connects all the entities and passively forwards packets between them. As previously explained, all the entities benefit from the ad serving system, except the access network. Thus, the access network may be tempted to tamper with the transiting data and create benefits for itself. Therefore, we consider an adversary located in the access network.

We assume a Dolev-Yao adversary \mathcal{A} [13]: \mathcal{A} controls the access network, has access to the messages passing through the network, and is a legitimate member of the network. It has the ability to *eavesdrop*, *alter*, *inject* and *delete* messages, hence it can perpetrate Man-in-the-Middle attacks.

We refer to \mathcal{A} as a *Network-in-the-Middle* (NitM) adversary. Such an adversary can take various forms in practice.

An example for a NitM adversary can be an attacker controlling a wireless access point (AP). To gain control over an AP, an attacker can subvert home wireless access points. As shown in [22, 20, 21], a significant portion of home wireless access points are vulnerable to hijacking attacks.

A stronger adversary could even deploy its own network of misbehaving access points. For example, wireless social communities (WSC) [6] could deliver localized advertisement and use the ad serving revenue to fuel their development.

Today, ISPs need to invest into the infrastructure to support the increased demand for bandwidth and to accommodate government requirements (e.g. blocking part of the P2P traffic) [3]. Thus, ISPs may be tempted to abuse the control over the Internet traffic they transport to get a return on their investments. ISPs have the means to monitor and modify the packets in real time. Deep packet inspection (DPI) [12] is a packet filtering technology that allows for the automatic examination and tampering of both the header and data payload of packets.

3.2 Attacks on Ad Serving

Given the total revenue generated by online advertising, an adversary has significant economic incentive to exploit the online ad serving to divert part of the revenues to its own benefit. In this section, we describe various NitM attacks against ad serving that can be perpetrated by an adversary \mathcal{A} located in the access network.

3.2.1 Adding Advertisement

\mathcal{A} can add advertisements to Web pages traversing the access network. It scans the payload of packets, determining where to inject the appropriate html code containing its own advertisements.

Injection of advertisements to any Web page, not necessarily targeted to the Web page's content or users' interests, is called a *pollution attack*. Rogers, a Canadian ISP, was reported to add content, notably advertisement for their own services, into any Web page that traversed their access network [23]. This was done by injecting into a Web page a single line of code that will cause the user to fetch and execute a Javascript as if it was part of the content of the Web page¹.

```
<script src="www.rogers.com/index.js"
type="text/javascript"></script>
```

Injecting advertisements not only generates revenue for the adversary but might also harm the reputation of a website or spoil the appearance of a Web page. As a consequence, the traffic on the website could decrease and cause a loss of online advertising revenues for both advertisers and the website.

A more sophisticated version of the pollution attack consists in *injecting ads in search results* returned by search engines. Search engines facilitate targeted advertising as search queries indicate users' interest at the moment. Several

surveys have obtained statistical information about users' behavior regarding Search Engine Result Pages (*SERP*) [15]. *SERP* contains a listing of Web pages returned by a search engine in response to a keyword query. If we consider *SERPs* without sponsored links (i.e., websites do not pay for their domain names to appear as search results), more than half of the users click on one of the first two results: 42.3% on the first and 11.9% on the second search result. The order in which results are displayed is determined by the search engine using a dedicated optimization algorithm. Knowing this, \mathcal{A} can inject its ad in *SERPs* such that it appears at the top of the list, resulting in a substantial increase of users' traffic on a website of the adversary's choice. In online advertising terms, this is an amazingly successful advertisement and \mathcal{A} can commercialize such services to advertisers. By injecting ads, \mathcal{A} thus bypasses the traditional ad serving model.

Several ISPs actually started partnering with advertisers to legally add advertisement to Web pages. Phorm [1] for example is an ad serving platform partnering with British Telecom, TalkTalk and Virginia Media, (which together represent approximately 70% of the UK broadband ISP market) to implement this ad serving model.

3.2.2 Replacing Advertisements

An adversary can substitute ads embedded into a Web page by its own ads or those from competitors. To do so, an access network must first identify packets containing ads, which can be done using DPI technology, or checking the IP destination addresses or URLs of the requested objects in the HTTP traffic. Then, \mathcal{A} can either replace the ads themselves, or change the URLs which identify ad servers from where ads are fetched (*redirection attack*). Now \mathcal{A} can control from which ad servers and which ads are embedded into Web pages. Consequently, \mathcal{A} controls the money flow, as it can divert the revenue from original ad servers towards ad servers of \mathcal{A} 's choice.

The *redirection attack* consists in redirecting users' requests towards arbitrary ad servers, chosen by the adversary. It can be implemented by modifying *URLs* of objects referenced in ad frames. When a user fetches a Web page, \mathcal{A} modifies the payload of packets carrying the *URLs* of the ad servers, such that the ads are fetched from different ad servers. An alternative way for \mathcal{A} to implement the redirection attack is to intercept the user's HTTP requests towards ad servers, fetch ads from ad servers of its own choice, and transparently send them to the user as a response to his HTTP requests.

The amount of money earned by an adversary perpetrating the redirection attack depends on whether it has an agreement with advertisers whose ads it will place on Web pages. In such a case, ad networks and websites will lose revenue to \mathcal{A} and its affiliated advertisers. But, to implement such an attack, \mathcal{A} has to build its own advertising infrastructure to select the most appropriate ads. \mathcal{A} could instead cooperate with one of the competing ad networks to forward all the traffic to their ad servers, thus relying on an existing ad serving infrastructure.

Instead of stealing the advertisement revenue, an adversary can replace advertisements with inappropriate and in-

¹The entire Javascript can be found in [23].

trusive ads to divert users' traffic away from certain websites. In a way, such an adversary then has indirect control over users' browsing habits.

3.2.3 Deleting Advertisements

Finally, an adversary can remove ads from webpages. For example, an ISP can automatically filter out all the advertisements and offer this as a service to its customers. Blocking advertisements is already possible at the end users [4, 5], but doing it network-wide would satisfy ISPs as it would reduce the amount of traffic to carry and work transparently for users. In addition, \mathcal{A} could have an agreement with certain advertisers or ad networks to filter out ads from competitors.

4. SECURE AD SERVING

In the previous section, we identified several possible attacks on the current ad serving system, that exploit vulnerabilities in the communications: (i) between a user and a website and (ii) between a user and an ad server.

To protect against these attacks, the *authenticity* and *integrity* of both the content and the advertisement have to be provided. Confidentiality is not a design goal, as it is not required to defend against a NitM adversary. To provide authenticity and integrity, communicating parties establish security associations (SA).

There are well-known protocols to establish an SA. Usually, security at the application layer relies on security protocols of lower layers, such as Internet Protocol Security (IPSec) or Transport Layer Security (TLS). In this section, we first explain the limitations of traditional approaches and provide a new solution in Section 4.2.

4.1 Limitations of Traditional Approaches

IPSec allows users to connect securely to a remote network by securing the communication between users and an IPSEC server at the network layer. However, it does not provide end-to-end security between a user and a website, because an adversary could be located between the IPSec network and a website.

The *Transport Layer Security* (TLS) secures end-to-end communication at the transport layer. The secure version of HTTP protocol, *HTTPS*, relies on TLS. HTTPS is used on the Internet to secure sensitive browsing data and would be a straightforward solution to secure the ad serving system. However, there are two strong arguments against deploying HTTPS at large scale: first, authentication issues when deploying HTTPS in practice, and second, significant overhead HTTPS introduces.

Authentication Problem

There are certain scenarios where *HTTPS* cannot achieve authentication in practice. One scenario is when servers downgrade security parameters during the connection establishment and the second is when digital certificates are not properly used for authentication. In cases when authentication fails, data integrity and confidentiality cannot be guaranteed and *HTTPS* fails to achieve its design goals.

We consider properly configured browsers such that a malicious server cannot downgrade the TLS parameters. A major problem remains: *How does a client authenticate the*

server with which it communicates? The current authentication procedure relies on the fact that legitimate servers own a public/private key pair and a corresponding valid digital certificate to prove their identity. As there is no initial trust between a client and a server, they rely on an independent trusted third party (*TTP*) to properly verify the identity of the server and the ownership of a given public key before issuing a valid certificate. We refer to a *TTP* that issues certificates as a Certification Authority, (*CA*). With the help of root certificates built into browsers², trust chains can be used to verify transparently the validity of certificates. The *X.509* standard is used on the web for the implementation of certificates.

Certificates signed by *CAs* are not widely used by websites because they are expensive. Instead, websites use *self-signed certificates*, signed with their own private key, thus users must trust the websites. A number of surveys of users' browsing habits show that even if an invalid certificate is detected and the user is properly warned via a pop-up message, the vast majority of users do not pay attention or do not understand the content of the message and accept the certificate. If users accept invalid or self-signed certificates, they might establish an *SA* with a malicious *WS*, instead of the legitimate *WS* with which the user wanted to communicate.

Overhead

Another drawback of using *HTTPS* for web browsing is the overhead it introduces, in terms of additional communications and computations. The major part of the overhead is due to the initial key exchange [10]. In the case of web browsing, sessions would be short and frequent, thus the overhead would be too high. As investigated in [19, 10] the throughput of an *HTTPS* server can be significantly lower than the throughput of an *HTTP* server. As confidentiality is not a requirement for fighting against NitM attacks, one might consider *HTTPS* with only data integrity and authentication features. Still, as the key exchange is not avoided, the overhead remains significant. This is the reason the Internet community has made a step towards solutions that will provide data integrity without *HTTPS* [19].

4.2 Securing Online Advertising

In TLS based approaches, the *WS* lacks economic incentives for securing its content, specifically, to acquire a certificate from a *CA* and to invest in the infrastructure required to support *HTTPS*. We propose a solution based on the economic incentives of ad networks, that have an interest in securing the content of *WSs* hosting their ads. In our solution, a new entity called the *Certificate Provider*, (*CP*), is associated with one or more ad networks and provides certification services to *WSs* and *ASs*. These certification services consist of: (i) collecting and validating the *WSs'* and *ASs'* certificates (self-signed or signed by a *CA*), (ii) securely providing the certificates on demand to the users. We argue that the emergence and deployment of *CPs* should be *sponsored by ad servers and advertisers* as they are the ones who will benefit the most. By protecting their own interests, ad networks indirectly secure users' web browsing experience,

²Users must trust software vendors for the authenticity of the root certificates.

giving incentives to users to adopt this mechanism as well.

To solve the second problem, the overhead of HTTPS, we propose a light-weight, scalable mechanism, based on long term security associations called *security cookies*. The main idea is that instead of generating per session SA as in HTTPS, we use a long term SA that is unique for each user-server pair and can be reused for multiple sessions. This avoids the expensive step of key exchange in HTTPS.

4.2.1 Certificate Distribution and Authentication

A CP can be thought of as an online instance of a CA, that has more flexibility and control in managing the certificates. The role of a CP is to help users validate certificates presented by WSs and ASs. The notion of such an entity can be found for example in Google's safe browsing service [18]. The Mozilla Firefox browser integrates the possibility for each URL a user visits to check whether it is a phishing website. It does so by contacting Google's servers that store a repository of blacklisted URLs that correspond to phishing websites. In this example, Google has the role of an entity whom users trust to check the validity of URLs on their behalf.

The first task of a CP is to collect and validate the certificates of WSs and ASs. The CP guarantees that a certificate for a domain actually corresponds to the domain. Before accepting a certificate of a new website, the CP verifies that the owner of the certificate is the owner of the website domain. This might require a manual verification for WSs/ASs with self-signed certificates, but it can be done automatically for certificates of websites signed by a CA. The cost for the CP to validate certificates would be borne by ASs as it is in their best interests to secure the content of the WSs.

The second service provided by the CP is to serve user requests for certificates in a secure manner. To authenticate a given WS/AS, a user retrieves the corresponding certificate from the CP and compares it with the certificate provided by the WS/AS. The user establishes a connection with the WS/AS only if the two certificates match. Thus, the CP prevents the impersonation of the WS/AS by malicious parties.

There are two modes in which the user can obtain certificates from the CP:

- *online verification*: For any URL a user visits, the certificate of the website is obtained from the CP. After obtaining a valid certificate from the online CP authority, the user caches it for future verification of the WS. For any later communication with the given WS, the user can verify the certificate locally and does not have to communicate with the online CP again, until the certificate expires or the caching period at the user's machine expires.
- *offline verification*: A CP pushes updates of the certificate pool to users, which are then stored at users' machine. Updates are done transparently (e.g. each 30 minutes a browser initiates a connection to the CP and checks for the updates). A number of optimizations can be done depending on the limitation factors: If the storage space is the bottleneck, a user can download certificates of the most popular WSs and use the complementary online verification approach to check

certificates that are not stored locally.

In order to protect users' privacy and prevent the CP from tracking and profiling users' browsing habits, and also to reduce the latency of rendering a Web page after a user issues a URL request, offline verification appears to be a better solution.

By acting as an online instance of a CA, the CP avoids the need to sign the certificates of the WS/AS. This provides greater flexibility in the management and control of certificates. Because certificates are either validated on each request (online verification), or the CP can push updates of the certificates to users (offline verification), the CP has greater control over the lifetime of certificates. This allows more flexibility to the ad network to include (remove) a certificate to (from) the CP's repository.

To protect against a third party trying to impersonate a CP, public keys of CPs should be pre-installed in browsers or CPs should have valid certificates signed by a CA. Users should be given an option to choose the preferred CP, i.e. whom they trust.

4.2.2 Security Cookie

Securing users' communication with CPs, WSs and ASs relies on long-term security associations stored in the form of cookies. We call these *security cookies*. A security cookie is a long-term shared secret key between a user and a server (WS, AS or CP) that can be reused over multiple sessions. It consists of a (user-key, pseudo-key) pair. The server generates a key for the user, called *user-key*, and a cryptographic representation of the user-key, called *pseudo-key*. The pseudo-key is obtained by encrypting the user-key with the server's public key, such that the user-key can be obtained from the pseudo-key *only* with the knowledge of the server's private key (i.e. only by the server). The server sends the (user-key, pseudo-key) pair to the user. The user's browser accepts and stores this security cookie. A shared secret (user-key) is now established between the user and the server, and it will be used to secure their future communication. Establishing a security cookie between a user and a server requires TLS support, with all the security features it provides: confidentiality, integrity and authenticity.

The server does not have to keep per user state, because a user associates his pseudo-key with his future requests, and the server obtains the user-key from the pseudo-key. For performance optimization, the user-key can be cached at the server for a short amount of time. The user may decide to renew his user-key, in which case the previously described procedure for obtaining the security cookie is repeated.

4.2.3 Data Integrity

Once a user obtains the security cookie, he uses the user-key to compute Message Authentication Code (MAC) to protect the data integrity. Indirectly, as only the user and the legitimate server can know the shared secret, i.e. the user-key, MACs also provide authentication.

Our scheme consists of the following (Figure 2):

- The browser establishes a security cookie over TLS with a trusted CP; This happens only once per user.
- The browser periodically communicates with the CP

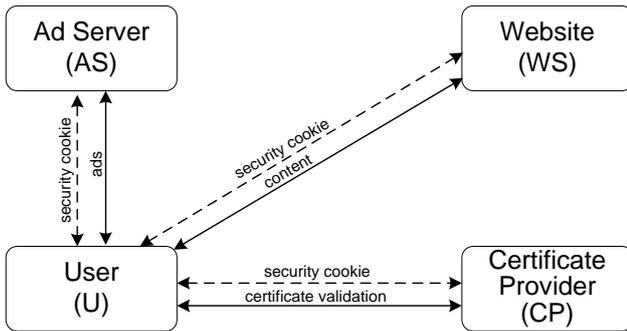


Figure 2: Secure content and ad delivery: dashed lines represent communication over TLS to establish security cookies, solid lines represent communications whose integrity is protected with the user-key.

to validate certificates in one of the two possible ways: online or offline verification. This communication is secured with the user-key and does not require TLS.

- When visiting a website for which the browser does not have a security cookie, the browser first establishes a security cookie over TLS with the website. This happens only once per user, unless the user decides to renew the security cookie, in which case this step should be repeated.
- When visiting a website for which the browser has a security cookie or after establishing a cookie, the browser associates the appropriate pseudo-key to user's requests. The server extracts the user-key to be used for MACs to protect data integrity and authenticity.
- The same security cookie mechanism is used to secure the communication between a user and an ad server, like described for the communication with websites. Security cookies are established per ad network.
- The outcome of the integrity check by the browser is binary: If the check is successful the browser renders the content, otherwise the browser does not display the content.

The drawback of maintaining a security cookie per ad network is that ad networks now have a unique identifier for a user and that they can profile the user's interests based on the type of the websites from where the user is redirected to retrieve ads. Considering that some of the ad networks already store a unique cookie on users' machines, and that others may track and profile users simply based on their IP address, we believe that the users' privacy is not violated more than it is today.

To implement the proposed mechanisms, some modifications in the browsers are required. These can be done using plug-ins without requiring modifications to the existing browsers.

4.2.4 Overhead

By extending the lifetime of security associations, instead of creating an HTTPS connection for each session, computation and communication costs are significantly reduced. In

the worst case, a user will have to create an HTTPS connection with the WS, the CP and the AS to establish security cookies with each of these entities. In total, the user must establish three TLS connections, each requiring two exponentiations from users, and three from servers. In the best case, when the user has the security cookies established with the CP, WS and AS, it does not have to do any computations, and the servers only need to derive the user-key from the pseudo-key with a single exponentiation. As MACs are computed using symmetric cryptography the overhead is negligible. Security cookies thus bring a significant performance gain, especially when users frequently browse same websites.

5. CONCLUSION

With the attacks presented in this paper, we have shown the tremendous power of the access network and the impact it can have on the ad serving system. This impact can translate into revenue loss for the advertisers, ad networks and websites and into lower security for the end users. These threats create incentives for all the entities to deploy the proposed scheme to secure the ad serving system, and protect the revenues and Web browsing. We have shown that providing authentication and data integrity is necessary, for the security of both, the content of Web pages and the ads themselves. In this way, ad serving would not only finance the Internet but would also fuel the deployment of online security.

6. REFERENCES

- [1] <http://www.phorm.com>.
- [2] <http://www.freefinet.com>.
- [3] <http://www.perftech.com>.
- [4] <http://adblockplus.org>.
- [5] <http://add-art.org>.
- [6] Fon. <http://www.fon.com>.
- [7] Microsoft proposes acquisition of yahoo! <http://www.microsoft.com/presspass/press/2008/feb08/02-01CorpNewsPR.mspx>.
- [8] Google to acquire doubleclick. <http://www.google.com/intl/en/press/pressrel/doubleclick.html>, 2007.
- [9] Internet Advertising Bureau. Iab internet advertising revenue report, 2007.
- [10] C. Coarfa, P. Druschel, and D.S. Wallach. Performance analysis of the web server. In *TOCS*, 2006.
- [11] N. Daswani and M. Stoppelman. The anatomy of clickbot.a. In *Hotbots*, 2007.
- [12] S. Dharmapurikar, P. Krishnamurthy, T.S. Sproull, and J. W. Lockwood. Deep packet inspection using parallel bloom filters. In *IEEE Micro*, 2004.
- [13] D. Dolev and A. Yao. On the security of public key protocols. *Annual Symposium on Foundations of Computer Science*, 1981.
- [14] M. Gandhi, M. Jakobsson, and J. Ratkiewicz. Badvertisements: Stealthy click-fraud with unwitting accessories. *Digital Forensic Practice*, 1(2), 2006.
- [15] Richard Hearne. Click through rate of google search results. <http://www.redcardinal.ie/search-engine-optimisation/12-08-2006/clickthrough-analysis-of-aol-datatz/>.
- [16] M. Jakobsson, P. D. MacKenzie, and J. P. Stern. Secure and lightweight advertising on the Web. *Computer Networks*, 1999.
- [17] B. Krishnamurthy, D. Malandrino, and C. E. Wills. Measuring privacy loss and the impact of privacy protection in web browsing. In *SOUPS*, 2007.
- [18] N. Provos. Phishing protection server specification. In *MozillaWiki*.
- [19] C. Reis, S. D. Gribble, T. Kohno, and N. C. Weaver. Detecting in-flight page changes with web tripwires. In *NSDI*, 2008.
- [20] S. Stamm, Z. Ramzan, and M. Jakobsson. Drive-by pharming. In *Technical Report*, 2007.

- [21] A. Tsow. Phishing with consumer electronics - malicious home routers. In *WWW*, 2006.
- [22] A. Tsow, M. Jakobsson, L. Yang, and S. Wetzel. Warkitting: the drive-by subversion of wireless home routers. *Journal of Digital Forensic Practice*, 1(3), 2006.
- [23] Lauren Weinstein. Google hijacked – major ISP to intercept and modify web pages. <http://lauren.vortex.com>.