Detecting and Defending against Certificate Attacks with Origin-Bound CAPTCHAs

Adil Ahmad[1], Faizan Ahmad[2], Lei Wei[3], Vinod Yegneswaran[4], and Fareed Zaffar[2]

[1] Purdue University
[2] Lahore University of Management Sciences (LUMS)
[3] Apple Inc.
[4] SRI International

Abstract. Published reports have highlighted various attacks on secure Public Key Infrastructure (PKI)-based SSL/TLS protocols. A well-known example of such an attack, that exploits a flaw in the Certificate Authority (CA) model of the PKI, is the compelled Man-in-the-Middle (MITM) attack, in which governments or affiliated agencies compel a CA to issue false but verifiable certificates for popular websites. These certificates are then used to hijack secure communication for censorship and surveillance purposes. Such attacks significantly undermine the confidentiality guarantees provided by SSL and the privacy of Internet users at large.

To address this issue, we present Origin-Bound CAPTCHAs (OBCs), which are dual CAPTCHA tests that elevate the difficulty of launching such attacks and make their deployment infeasible especially in cases of mass surveillance. An OBC is linked to the public key of the server and by solving the OBC, the client can use the certificate to authenticate the server and verify the confidentially of the link. Our design is distinguished from prior efforts in that it does not require bootstrapping but does require minor changes at the server side. We discuss the security provided by an OBC from the perspective of an adversary who employs a human work force and presents the findings from a controlled user study that evaluates tradeoffs in OBC design choices. We also evaluate a software prototype of this concept that demonstrates how OBCs can be implemented and deployed efficiently with 1.2-3x overhead when compared to a traditional TLS/SSL implementation.

Keywords: Compelled-certificate attacks, Man-in-the-Middle attacks, CAPTCHAs

1 Introduction

Research in cryptographic security schemes has come a long way in making communication on the Internet more robust and secure. While the underlying cryptographic protocols are provably secure and theoretically sound, most attacks on the Internet take place due to implementation bugs and exploitation of factors such as false certificates, untrustworthy plugins, and vulnerable browsers. The problem of false certificates has plagued Internet for a long time, and there has been a concerted effort from major companies to mitigate this using solutions like certificate blacklisting. A Certificate Revocation List (CRL) is programmed into all browsers but needs to be constantly updated, a fact that most users remain completely oblivious to. Even with constant updates, there are numerous certificates that are uncaught.

An important and common certificate-based attack is the ***compelled*** Man-in-the-Middle attack [27]. The current PKI model, used for server validation, relies on the legitimacy of the Certificate Authority (CA). Therefore, when a CA issues certificates to an entity, users trust that entity without worrying about the integrity of the certificate. This poses a serious risk because attackers have, in the past, hijacked certificates from certificate authorities and launched massive attacks. A detailed analysis of the risks and problems associated with current PKI has been provided by Roosa et al [25].

The problem of false but verifiable certificates is a significant issue undermining contemporary Internet security. There have been several reports highlighting the theft of verifiable certificates from CAs. Comodo is one of the CAs to publicly acknowledge the theft of certificates of various high-profile sites including the highly coveted *google.com* [4]. Another example is DigiNotar, a Dutch CA, which took a month to acknowledge that its certificates had been stolen, which resulted in the Dutch government seizing its operations [26]. Nation states have also been suspected to compel Certificate Authorities to issue SSL certificates which can then be used to carry out automated MITM attacks. Such reports are highly disturbing because if government agencies control CAs, then mass surveillance becomes trivially deployable.

Consider, for example, the case in which an attacker gets hold of verified certificates and wants to hijack communication between a legitimate client and server. The attacker can simply impersonate the legitimate server, using the verified certificate. Such MITM attacks are extremely dangerous because they enable the adversary to (*i*) passively eavesdrop, (*ii*) tamper with information, and (*iii*) coerce the user into sharing sensitive information. Both the user and the legitimate server may remain oblivious due to the stealthy nature of such attacks. The attacker obtains the data from the user, forwards to the server by pretending to be a user, and then transmits any response it gets from the server to the user. Also, this attack can easily be perpetrated in an *automated* fashion to eavesdrop on the browsing activities of a large group of people for surveillance and censorship purposes.

In recent years, a number of efforts [15,20,16,31] have tried to mitigate the problem of a compelled MITM but as yet there is no solution that is (*i*) easily deployable, (*ii*) user-friendly, (*iii*) does not rely on trusted third-parties, and (*iv*) significantly reduces the attack capability of the attacker. One of the major concerns with previous designs is that it may reveal a surveillance attack is taking place, but does not hamper capabilities of the attacker. Our work tries to address this problem with a CAPTCHA (Completely Automated Public Turing Test to Tell Computers and Humans Apart)-based solution that we believe addresses all of the above requirements with only minor modifications to the systems in place and a moderate usability cost.

A CAPTCHA, introduced by von Ahn et al. [30], is a program that makes a human solvable test that goes beyond the capabilities of current computers. The idea is for the server to challenge any communicating endpoint with a test which, if solved, allows the server to confidently assume it is interacting with an actual human being. Traditionally, servers have used CAPTCHAs to validate that the clients are real humans and not bots programmed by an attacker to perpetrate Denial-of-Service (DoS) attacks. The race to develop improved and resilient CAPTCHAs is an active area of research. This has also stimulated adversarial research into developing automated CAPTCHA solvers. Although Machine Learning (ML)-based attacks have proven to be effective on leading text-based CAPTCHAs [11], there is also new research into novel CAPTCHA designs that resist automated and human-solver relay attacks [19,24,18].

In this paper, we make the following contributions:

1. Present the Origin-Bound CAPTCHA (OBC), which is a novel CAPTCHA designed to validate the legitimacy of the server. It consists of a dual-CAPTCHA test that has an embedded proof telling the user whether or not the OBC and the certificate originated from the same server.
2. Provide a detailed security analysis of the OBC and demonstrate that it is infeasible for an attacker to solve a large amount of CAPTCHAs because it would require hiring a large human workforce.
3. Provide the findings of our user study to demonstrate the usability tradeoffs our design has to make in order to mitigate the impact of such attacks.
4. Provide a reference implementation and show that OBC is deployment-friendly in an incremental way.

The paper is divided into 10 sections. Section §2 discusses the related work in this area. In section §3, we dissect the attack, and in section §4, we explain the design considerations. Section §5 explains how an OBC is created, and section §6 deals with a detailed security analysis of the design. We document the findings of our user study in §7, discuss our reference implementation in §8, and show the performance overhead incurred in §9. We discuss the limitations of the approach and potential avenues for future work in §10.

2 Related Work

Significant effort has been dedicated to finding and mitigating the effects of MITM attacks. Designs that have been considered include better CA models, more secure client-authentication using channel-IDs, reviewing historical data of visited sites, and using trusted third-parties.

Researchers in [16,23] attempt to tackle the problem by improving the Certificate Authentication model to account for all certificates currently in circulation. Some [29,28] have attempted to introduce publicly verifiable logs and monitoring solutions to mitigate problems with current PKI models. For these designs, bootstrapping and deployment in the current Internet ecosystem continue to be a huge concern, which is not likely to be the case with our proposal. Others have tried to strengthen client-authentication by using TLS-based Channel IDs [20,10], but such a design has been shown to be susceptible to other kinds of MITM attacks [22].

Perhaps the most closely related work is Certlock [1] – a system that employs a similar plugin-based approach to deal with compelled certificate attacks. The key difference between their system and ours is that Certlock relies on prior historical data of visited websites to detect anomalous behavior. In contrast, our system requires no bootstrapping. Other prior work has dealt with solving the issue of server impersonation by employing enhanced server verification techniques such as multi-path probing, pinning, etc. [16,31]. The Perspectives [31] and DoubleCheck [9] systems use notary servers to address such SSL MITM attacks. However, these systems rely on sharing browsing information to a trusted third party. Our approach does not have this requirement. The Heise SSL Guardian system detects weak SSL certificates [3]. Other work in this field includes the Multi-Factor Authentication (MFA) that saves user information from being accessible to the adversary in case of password leakages through MITM attacks. This method is currently employed by many top companies [5] in their applications but is rarely used by the end-users because it is veryinconvenient. Recent work [21] has also focused on ways to improve the two-step verification, but it remains an unpopular means of authentication.

Furthermore, prior work [17] has also drawn attention to the utility of server-bound CAPTCHAs, addressing the problem of CAPTCHAs being redirected to other websites. Their technique fails to protect users from a compelled MITM attack. The authors of [17] bind the IP and URL address to CAPTCHAs, which fails to control the attack.

3 Threat Model

In this section, we describe the main focus of the work, a targeted attack on unsuspecting users by snooping on their normal browsing activity. This kind of attack can only be perpetuated by an ISP-level attacker or an attacker that has hijacked a router along the pathway that a user or a group of users use to connect to the Internet. While discussing the attack, we make the following assumptions:

1. The adversary targets *all or a major chunk* of users in his/her domain through automated attacks for surveillance purposes.
2. The adversary possesses the ability to perform *DNS/IP hijacking* and *Deep Packet Inspection (DPI).*
3. The adversary has access to *false but verified certificates* through colluding CAs.

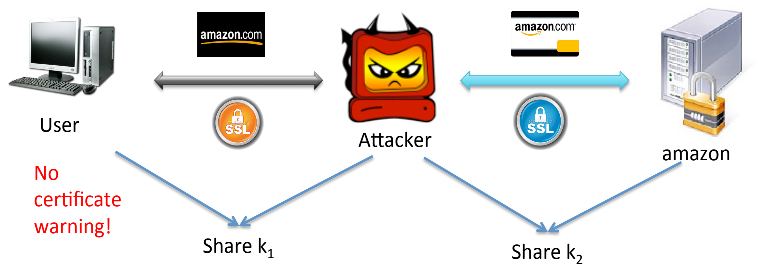

Fig. 1. Illustration of a compelled MITM attack between the User and Amazon using false certificates

An adversary is able to intercept traffic between two communicating entities, redirect the traffic to its machines, and is able to read the transmissions through deep packet inspection and transmit them in a reasonable amount of time. The adversary needs to be able to respond in a timely manner, otherwise the connection will be dropped from the client or the server side due to timeouts. If the traffic is unencrypted, this becomes a trivial matter for an adversary with the abilities mentioned above.

However, if the attacker obtains access to verified certificates, he can just as easily read the contents of an encrypted connection. To do so, he must intercept incoming packets, redirect them to his machines using DNS/IP hijacking, send back the verified certificate through DPI, and simultaneously create a connection with the intended server. As such, the adversery merely needs to act as a conduit for the communication between the client and the server, and he can easily snoop on private data since he is the one who has an established shared secret with both the user and the server. The client remains unaware of the attacker because it received a verified certificate. The server, having received valid credentials, also has no reason to suspect foul play. Also, this attack can easily be carried out through automated scripts. The attacker simply directs all relevant traffic to its machines, performs the aforementioned procedure and dumps all intercepted data without having to maintain a human presence. Fig. 1 explains this attack.

Traditional CAPTCHAs, widely used to distinguish bots and human users, also fail in detecting this attack. Instead of solving the CAPTCHAs, the adversary can simply redirect the CAPTCHA to the user and ask him/her to solve it. There have been reports of certain adult websites being used by attackers to make users solve CAPTCHAs for them. Researchers in [17] address that problem by using URL/IP binding on CAPTCHA content to ascertain the server address where the CAPTCHA was created.

4 Design Considerations

The nature of the attack compels us to make a few choices. These choices drive the design and implementation of an OBC.

4.1 Why CAPTCHAs?

Since their introduction in 2003, CAPTCHAs have become the tool of choice for e-commerce websites to facilitate user authentication and defend against DoS attacks. According to an estimate [7], more than 614,000 websites use various kinds of CAPTCHAs. We present two main arguments while making a case for CAPTCHAs as a tool to defend against compelled certificate attacks.

Familiarity. The extensive use of CAPTCHAs on the Internet means that users now possess a sense of familiarity with the software. Users have grown accustomed to solving CAPTCHAs in order to prove to the server that they are humans and not bots. Therefore, it would therefore be easier for them to adopt a similar solution to test the legitimacy of their connection rather than employ a new scheme. CAPTCHAs also offer an attractive range of possibilities such as text-based CAPTCHAs and audio/video CAPTCHAs, which improve both usability and accessibility of our scheme.

Security. CAPTCHAs have been proven to provide reasonable security against automated attacks. In the past, various attacks (especially ML-based) have been demonstrated against CAPTCHAs but have always been thwarted by trivial tweaking of the CAPTCHA modules. The current *state-of-the-art* are complex modules such as reCAPTCHA, that balance usability and resilience, using intelligent image distortion techniques to make it really hard for ML-based algorithms to crack them. Hence, the attacker must employ a human workforce or conscript unaware users in order to solve the CAPTCHAs. In short, the battle between CAPTCHA generators and solvers is a perennial arms race that we expect to continue for the forseeable future [11].

Limitation. People find it quite cumbersome to solve CAPTCHAs and it could be a major deterrent for some of them. However, we believe potential advantages gained through privacy preservation outweigh the trouble that solving CAPTCHAs would create. Also, newer proposals such as [13] illustrate ways to make it easier for people to solve CAPTCHAs without compromising the security of a CAPTCHA and can easily be used with OBC.

4.2 Proposed Modifications

Our work does not make changes to the TLS protocol and instead relies on simple modifications at the servers. The only modification required is that the server concurrently transmits an OBC with its certificate upon the initiation of a TLS-handshake and asserts the solution provided by the client before establishing the connection. At the client end, an installed browser plug-in intercepts the OBC and prompts the user to solve it. If the verification steps related to the Origin Bound CAPTCHA (mentioned in the next section) are successful, the client can authenticate the legitimacy of the connection and send his/her sensitive information securely.

We believe that to successfully avert an ISP-level attacker from snooping, there has to be some support from the server side. Our scheme would increase client trust on the servers, much like other schemes such as Multi-Factor Authentication.

5 Our Approach

An OBC is a dual-CAPTCHA test with the purpose of server validation, at the client side, in order to prevent compelled certificate attacks. The server is responsible for creating an OBC that it transmits back to the client as part of the SSL-handshake response. The user solves the CAPTCHAs and verifies it against the certificate being provided to authenticate the remote server. If the legitimacy is established, the user sends back the solution and, if not, terminates the connection and restarts the process. From here onward we refer to a legitimate SSL-enabled webserver as a *server* and a legitimate end-host as a *user*.

Our approach is tailored to fit the following assumptions:

1. We employ a ***passive detection*** technique in which we test the legitimacy of the connection only when transmitting sensitive information, e.g., username/passwords etc. and expect cryptographic protocols to take in afterwards.
2. We assume the attacker is not able to obtain the secret key corresponding to the public key of the server by breaking the cryptosystem. However, the user is able to obtain a verified certificate for any site that the user desires although the certificates will not be the same as those of the server that creates the OBC.

Intuition. The basic insight of our approach is to bind the content of the CAPTCHA to the identity of the server, so that the user solving the CAPTCHA is able to verify that the certificate that the user received during the SSL handshake indeed belongs to the server the user is communicating with. The CAPTCHA becomes non-transferable in some sense because the attacker risks being detected if he simply forwards the CAPTCHA to the user to solve, but would need to deploy a human being to solve by himself. While older CAPTCHAs can be solved by ML-based attacks [11], there is newer research that suggests new CAPTCHAs will be designed to resist existing attacks [19], including those that resist human-solver relay attacks. With hundreds of thousands of such CAPTCHAs needed to be manually solved in a short time (for many users), this becomes a non-trivial task for the attacker as we explain in Section 6.3.

Construction. Suppose the authentic certificate of the server endorses the public key PK. An OBC consists of two traditional CAPTCHAs A and B whose contents are derived from H(PK || s) and s, in which 'H' is a hash function, s is a short random salt and || is concatenation function. In this scheme, A is a CAPTCHA for s, displayed as characters in hexadecimal format. For usability reasons, we choose s to be about 20 bits (5 characters shown in hexadecimal format). On the other hand, B is a CAPTCHA for the value of H($PK \mid\mid s$) and only the first 6 characters in hexadecimal, i.e., 24 bits, of the hash value, are displayed.

We assume a browser plug-in is installed on the user's machine. The plug-in records the public key PK' endorsed by the certificate that is verified during the SSL handshake process. Before the user is asked to input his login credential, the server sends the CAPTCHA to the user to solve. The add-on obtains the solution of the two CAPTCHAs, which we denote by the two strings as a and b, and verifies if the first six characters of H($PK' \mid\mid a$) equal b. If this condition is not satisfied, it raises an alarm to the user and closes the connection. If the attacker used a PK' different from PK during the SSL handshake there is high probability the hash value would not verify. This would occur if the attacker intercepted the SSL-handshake request and replied with his/her own certificate without changing the CAPTCHAs sent by the server.

Step-by-Step. We now describe the above construction as a series of communication events between a user Alice and a service provider (server) Bob. An enumerated description of the aforementioned construction is shown below.

1. Alice attempts to initiate a SSL-handshake with Bob.
2. Bob acknowledges the handshake and returns his certificate with an OBC.
3. The browser plugin stores the CAPTCHAs A and B from the accompanying OBC.
4. First, Alice is prompted to solve CAPTCHA A.
5. Alice solves A, and the resulting value a is stored by the installed plug-in.
6. Now, Alice is prompted to solve B by the plugin.
7. Alice solves B, resulting in b, and the following relation is compared:

$$\textit{First 6 digits of H(PK} \mid\mid \textit{a)} == b \tag{1}$$

8. If the relation holds, the credentials are sent forward alongwith the solution to the OBC. Otherwise, the user is notified his connection might not be secure.

In the next few sections, we provide a security and usability analysis of this approach.

6 Security Analysis

In this section, we explain in detail how the MITM attack is carried out and how our work counters it. We also explain why a manual attack on a large number of people is almost impossible if OBC is employed, the semantics of the hash function we use, and the probability of encountering a collision.

6.1 Detecting an Attack

Consider the attack mentioned in Section. 3, i.e. an attacker tries to hijack a connection between a client and a server by presenting parallel realities to each of them. The capabilities of the attacker are that it controls a router along the pathway, can do arbitrary DNS/IP hijacking and has a false but verified certificate for the site that the user wants to initiate an SSL connection with. However, using OBCs, it becomes non-trivial to fool the client to think that you are the valid server and not the attacker. We provide an illustration of this in Figure 2.

Suppose an attacker wants to hijack communication between User and Amazon. Using the router he has on the pathway between User and Amazon, he finds out the User is trying to contact Amazon. Therefore, he creates a connection with the User saying he is Amazon and he creates a connection with Amazon claiming to be the User. Since Amazon is using our technique, it sends the attacker an OBC that the attacker dutifully relays to the User. The User solves the OBC and verifies the relation provided by Equation. 1. However, the certificate the attacker sent to the User during the initial handshake will have a different public key PK', and the relation will not be satisfied. Therefore, the browser plug-in will realize there is something wrong with the connection and will terminate the connection with the User's consent.

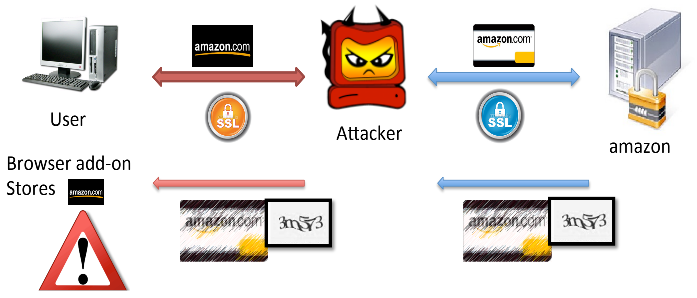

Fig. 2. Workflow illustrating how the MITM attack between the User and Amazon is detected with OBCs

6.2 Cryptoanalysis and Implementation Considerations

We use the SHA-256 hash function for the computation of the 6-digit hash h of the salt s and public key PK, i.e., $h = \mathrm{H}(PK \mid\mid s)$. We first calculate the probability of a hash collision such that the public key provided by a malicious attacker has a combined hash with the salt s, h', that collides with the hash

h provided by the server. The probability $q(n)$ that the attacker generates a random public key that hashes s to $h' = h$ is given by, where $n = 1$ and $d = 16^6$, we obtain $q = 5.96 \times 10^{-8}$.

More concerning, however, is the probability that the server issues the same OBC to multiple clients. For any hash function–assuming random hash values with a uniform distribution, a collection of n different data blocks and a hash function that generates b bits–the probability p that there will be one or more collisions is bounded by the number of pairs of blocks multiplied by the probability that a given pair will collide. For example, setting $b = 24$, we observe that to maintain a probability lower than 2^{-10} that there is a collision between two hashes, with the currently implemented 300-second expiry interval for each hash pair, the server should open up new connections with clients at a rate no faster than 109 connections per second.

If we use 5 and 6 Base-64 digits for the salt and hash, respectively, in place of the hexadecimal digits currently implemented, the rate for new connections to maintain the same probability goes up to 134217728. To avoid server-side storage overhead when dealing with multiple connections, we employ action tokens as a method of providing end users a reasonable time window to solve the OBC. However, if we keep track of issued and unsolved-yet-currently valid CAPTCHAs on the server, it is possible to reduce the probability of collision to 0 while enforcing a limit on the number active OBCs served.

6.3 Employing a Human Workforce

As described obove, OBC provides reasonable security against attacks on its design. Therefore, the only way that the attacker can get away with the attack is by:

- Initiating the attack as mentioned in section 3.
- Creating an OBC with his own public key PK' and sending it to the user.
- Solving the OBC that will be sent by the legitimate server.

This is non-trivial because even a medium-level ISP has to solve literally thousands of SSL-handshake requests in a very small time. The exact number of people that an attacker will have to hire in order to hijack all SSL communication within his sphere depends upon the average number of active users, average number of SSL connections initiated in a time interval, the average rate of solving CAPTCHAs, and the error rate of solving CAPTCHAs. Studies [30] have shown it takes about $10s$ on average for a person to solve a CAPTCHA. Another factor that needs to be accounted for is the connection timeout limit, which is normally set to $1\,minute$ in a popular browser such as Google Chrome. For a 2-minute interval, neglecting the exact start and end time for the SSL connections, the relation given below holds.

$$N_{people} = \frac{N_{avg} \times N_{users} \times N_{ssl} \times 2}{12 \times r_{error}^2} \tag{2}$$

In the equation, N_{avg} represents the average percentage of people active, N_{users} represents the total number of users registered with the ISP, N_{ssl} represents the average number of SSL connections, and r_{error} represents the error rate (which is a value between 0 and 1). We assume that takes about 10 seconds for a person to solve a CAPTCHA and if he/she works tirelessly for 2 minutes, about 12 CAPTCHAs can be solved. The error rate is squared because both the attacker and the user can make a mistake while solving CAPTCHAs. Here, we assume that all SSL connections are OBC-enabled.

To understand the equation, consider a 2-minute interval in which the first minute is spent requesting SSL handshakes which must be solved within the next minute otherwise the connection will be terminated by the browser. So, any connection created between T = 0 and T = 60 will have to be responded to by T = 60 to T = 120. To simplify the equation, we neglect the exact start times and apply the worse case, which means that any connection created between T = 0 and T = 60 has to be responded to by T = 120.

Let us consider the case of a small ISP with about 100,000 registered users and also suppose each user uses only a single device at a time and the percentage of active users is 5%. Furthermore, suppose each active user initiates only a single SSL connection in the interval. Without considering any error, more than 800 people will be required to work tirelessly to hijack all the encrypted communication within an isolated 2-minute interval for a small ISP. The number will rise exponentially as the number of users and/or the error rate increases. Hence, we believe employing a human workforce is a non-trivial barrier for medium-to-large large ISP environments.

7 Usability Analysis

To better understand the usability of OBCs, we conducted a two-week user study in our institution (Lahore University of Management Sciences) that involved 150 subjects. Our study was approved by our institute's Institutional Review Board (IRB). All participants consented to their participation in the study and data was sanitized to ensure that personal private information would not be disclosed to the public. We did not mass-distribute the study and limited it to the confines of our academic institution.

Our user study was conducted online using a website that was only accessible within the institute's network. All the participants were consenting adults between the ages of 18-35 years old and *non-native* English speakers. One caveat of our study is that it was fully conducted in a university environment and thus the results may be less indicative of a wider userbase (since younger students may be more technology savvy than the general public).

To participate in the study, users were first instructed to visit the OBC user study website link and provided with a survey that must be completed without pauses to ensure the consistency of our results. We conducted the study with two main CAPTCHA libraries, `captchas.net` [14] and `Securimage PhP`

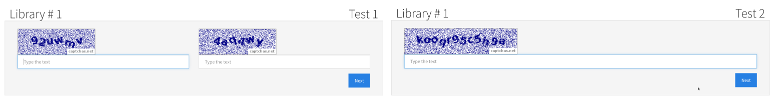

Fig. 3. User study depictions for design 1 (left) and design 2 (right)

`Captcha` [2]. Both these libraries are popular, open-source, and have continued support from the developers.

We outline below the specific goals of our user study:

- To measure out how long it takes for users to solve an OBC.
- To measure the error rate for users in solving an OBC.
- To evaluate various design choices and ascertain user preferences.

7.1 Introductory Message

The user were provided the following information at the start page:

"This is a study on the efficiency of CAPTCHAs. CAPTCHAs are commonly used to prevent automated attacks on webservers. This study will take 5-10 minutes. Please complete this study in a single pass and complete the survey provided at the end."

7.2 CAPTCHA Tests

We considered two main design choices for an OBC:

1. Two small CAPTCHAs of 6 characters, i.e., Captchas A and B.
2. One large CAPTCHA of 12 characters (formed by joining both CAPTCHAs A and B together).

Our main goal was to measure the time taken, error rate, and user preference for each design. For design 1, we simply showed the participants two CAPTCHAs side by side as in Figure 3. To calculate the time taken, we started a timer in the background as soon as the user started typing on the provided box. We set timers for individual CAPTCHAs as well as a timer that stopped when both boxes were filled and the user moved onto the next test. Figure 3 (right) depicts design 2, in which the user was asked to fill the 12 character CAPTCHA to ascertain the same insights we wanted from design 1. Here, it should also be mentioned that we used both libraries for each design, and users solved five OBCs for each library and for each design, which totals to 30 distinct (twenty 6-character + ten 12-character) CAPTCHAs per-user.

7.3 Survey

After completing the CAPTCHA tests, the users were asked to complete a simple survey that posed some simple questions about which design they preferred and which library of CAPTCHAs was easier for them.

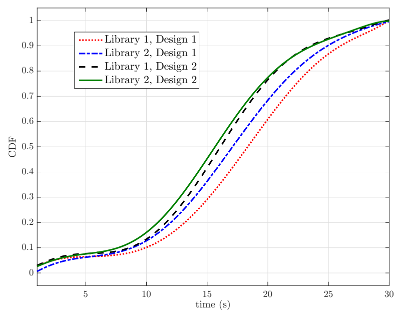

Fig. 4. CDF graphs for completion times for CAPTCHAs using design 1 and design 2

Table 1. Statistics from user study

	Design 1		Design 2	
	Lib 1	Lib 2	Lib 1	Lib 2
Avg. Time (s)	18	17.5	16	15.3
Err. Rate (%)	31.7	25.1	37.56	33.52

7.4 Results

Figure 4 shows the cumulative distribution function (CDF) o f completion times with designs 1 and 2. The CDFs were quite similar to each other with design 2 performing slightly better, which is as expected since the users don't have to switch between solving two CAPTCHAs and are simply solving a larger CAPTCHA. Table 1 provides a summary of the average completion times for users with all designs and the error rates that we encountered with the individual designs.

Our results indicate solving CAPTCHAs incurs a high error rate; the accuracy of our tests, which converged at a mean of 69%, is similar to previous studies [12] involving multiple CAPTCHAs and non-native English speakers. The accuracy is consistently within the range of 65-75%, which indicates there is still a fair amount of work to be done to make CAPTCHAs more solvable for humans.

Our results also show the vast majority i.e., almost 75%, of our subjects prefer design 2 (one long CAPTCHA) over design 1 (two smaller CAPTCHAs). Furthermore, the error rate does not necessarily increase by a significant margin when we move from design 1 to design 2, which makes design 2 seem even more practical for large-scale deployment.

8 Implementation

To evaluate our technique, we developed a reference client-server module using Node.js [6], a popular server-side JavaScript environment. Node.js was chosen for rapid prototyping since it allows us to run a webserver without using Apache, and both server and client-side code can be written in JavaScript. The Node.js module was used for stress testing our technique against vanilla-TLS, the results of which are presented in the next section. Node.js also relies on event-driven programming and non-blocking, is an ideal framework for real-time applications and provides numerous APIs for the programmer to use. Also, the V8 JavaScript runtime engine, which Node.js uses, has been optimized to provide performance similar to that of low-level languages. Another benefit is that Node.js works with a variety of modern browsers such as Google Chrome and Firefox.

8.1 The OBC Stack

The OBC implementation stack consists of two main modules: the browser plugin and an OBC-enabled server.

OBC-enabled Server. We have used the term “OBC-enabled Server” to simply distinguish servers that have an OBC-patch [5] installed from the ones that do not. The difference in the working of an OBC-enabled server and a non-OBC server is that the latter sends an OBC to the client to solve and waits for the solution to the OBC before confirming the TLS/SSL handshake. The server tags the outgoing webpage with an OBC tag to signal to the client plugin that an OBC-enabled connection is about to be initiated.

Browser Plugin. The plugin is a non-intrusive add-on to the client’s browser and is similar to various common extensions such as AdBlock etc. It serves the following functions:

- Checks TLS-response to ascertain an OBC-enabled connection
- Redirects an OBC onto a seperate page and asks the user to solve it
- Captures user’s response and recreates the relationship proven by Equation. 1.
- Informs the user about the legitimacy of the connection

In our implementation, we used a Firefox plugin, but it can be easily implemented in any modern browser. The plugin’s main role is to hide the semantics of the operations from the User and simply present the him/her with a ready-to-solve page embedded with an OBC and an MITM alert, in case the relationship does not hold.

[5] An “OBC-patch” could mean a change to the TLS protocol or installation of a shim layer that works in tandem with the TLS-protocol and is responsible for dealing with the OBC. We are in favor of the latter approach.

8.2 OBC Workflow

The workflow of the OBC reference implementation is as follows. The user attempts to initiate a TLS connection with an OBC-enabled server. The server replies with the certificate and an OBC. The client plugin observes that an OBC-enabled connection is underway, prepares itself by storing the certificate, and consequently the public key (PK) of the contacting server.

The server sends a page on which an 'OBC' is embedded and the plugin simply presents that to the user. The user solves the OBC and the solution is extracted by the browser plugin. The plugin ascertains the conditions established in equation (1). The only way those relations will not be fulfilled are if:

- There is a certificate-attack being carried out.
- The user solved the CAPTCHAs incorrectly.

Newer forms of CAPTCHAs have been shown to have solution error rates less than 10% [13] and therefore we have opted not to recheck the solutions. In case of an error, we simply assume the worst-case and reset the connection. A new OBC is then requested from the server and the whole process is repeated.

9 Performance Analysis

In this section, we compare the performance of an OBC-enabled webserver with a traditional SSL webserver. The webserver used for the comparison is a Core i7 PC with 16GB RAM and internal SSD memory. The following graph depicts the time taken in case of varying number of connections. We have ignored the network latency and focused solely on the stress it induces on the PC.

As shown in Figure 5, the overhead of using an OBC increases as the number of concurrent connections established increase. With 10 concurrent connections, the performance of OBC TLS and vanilla TLS are comparable. As the number of concurrent connections increases to 100, the response time of the OBC TLS server increases to 4.5s (this is a $3\times$ overhead in comparison with the vanilla TLS server response time of 1.5s). The overhead can be attributed to increased computations the server has to perform to create the CAPTCHAs to send to the clients. However, a few additional details need to be considered.

- The prototype implementation is unoptimized. In these experiments, the CAPTCHAs are created each time a connection is attempted. The time it takes to create a CAPTCHA is significant when compared to other trivial computations of the TLS protocol. If we can have a pre-computed local database of CAPTCHAs from which servers can simply extract, the overhead will reduce significantly. This is a straightforward extension we will pursue in future work.
- In most datacenters [8], effective load-balancing ensures that load is distributed amongst servers in such a manner that the overall instantaneous stress on a single server system is minimized.
- Finally, the benefits of using an OBC in guaranteeing reasonable security far surpass the induced overhead in many situations.

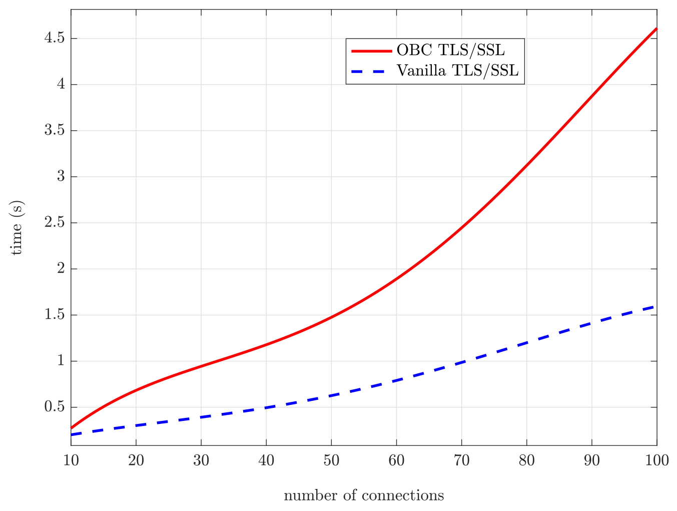

Fig. 5. Comparison between OBC and vanilla TLS

10 Limitations and Future Work

There is one scenario in which OBCs would fail. Suppose an attacker tries to simply get username/passwords for users connecting to a social networking site like Facebook or Twitter using fake certificates. As soon as the username and passwords are harvested, the attacker can simply end the transmission and allow the retry connection to go through unsnooped. In such instances, the plugin should be redesigned to warn users when there is a certificate mismatch between the original and retried connection. However, we believe such an attack is infeasible for mass surveillance on the internet, which is the main focus of this work.

CAPTCHAs are a necessary drawback of our system to make mass surveillance infeasible for ISP-level attackers. We hope that future research in improvements to CAPTCHA schemes will benefit our system and make it more accessible to normal users. We aim to further investigate how the OBC approach could be deployed on the Internet with the least number of modifications. We would like to improve the OBC stack by introducing stored databases of CAPTCHAs that would raise the performance of an OBC-enabled TLS/SSL quite close to that of vanilla TLS/SSL connections. We would also add support for other kinds of CAPTCHAs and also investigate if a modification of OBCs can be used to prevent other MITM attacks.

11 Conclusion

This paper presented the design and implementation of OBCs, a browser plugin-based solution against compelled MITM attacks. The simple and promising solution relies on CAPTCHAs, but requires no certificate pinning, bootstrapping, trusted notaries or third parties. We presented a detailed description of our design and conducted a user study to assess design tradeoffs. To validate our approach, we developed a simple prototype implementation that demonstrates how OBCs can deployed without prohibitive performance penalties. We believe if OBCs are deployed ubiquitously, they can render compelled MITM attacks ineffective, at least temporarily, and raise the bar by forcing adversaries to invest in human resources for CAPTCHA solving. We hope our approach can benefit from improvements in CAPTCHA design and inspire the development of alternate solutions against compelled MITM attacks, that do not rely on CAPTCHAs.

12 Acknowledgements

We would like to thank all the anonymous reviewers of the program committee for their valuable insights on the paper. This work was partially funded by the National Science Foundation (NSF) under Grant No. CNS-1514503. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of NSF.

References

1. Certlock - securew2, `https://www.securew2.com/products/certlock/`
2. Securimage php captcha, `https://www.phpcaptcha.org/`
3. Heise ssl guardian: Protection against unsafe ssl certificates. www.h-online.com/security/features/Heise-SSL-Guardian- 746213.html. (2008)
4. Comodo report of incident. https://www.comodo.com/ComodoFraud-Incident-2011-03-23.html. (2011)
5. Google 2-step verification. https://www.google.com/landing/2step/ (sep 2016)
6. Node.js. https://www.nodejs.org/en/ (jul 2016)
7. Sites using captchas. https://wappalyzer.com/categories/captchas (jul 2016)
8. Abts, D., Felderman, B.: A guided tour of data-center networking. Communications of the ACM **55**(6), 44–51 (2012)
9. Alicherry, M., Keromytis, A.D.: Doublecheck: Multi-path verification against man-in-the-middle attacks. In: Computers and communications, 2009. iscc 2009. ieee symposium on. pp. 557–563. IEEE (2009)
10. Balfanz, D., Hamilton, R.: Transport layer security (tls) channel ids. IETF Draft (2013)
11. Bursztein, E., Aigrain, J., Moscicki, A., Mitchell, J.C.: The end is nigh: generic solving of text-based captchas. In: 8th USENIX Workshop on Offensive Technologies (WOOT 14) (2014)

12. Bursztein, E., Bethard, S., Fabry, C., Mitchell, J.C., Jurafsky, D.: How good are humans at solving captchas? a large scale evaluation. In: IEEE Symposium on Security and Privacy. pp. 399–413 (2010)
13. Bursztein, E., Moscicki, A., Fabry, C., Bethard, S., Mitchell, J.C., Jurafsky, D.: Easy does it: more usable captchas. In: Proceedings of the 32nd annual ACM conference on Human factors in computing systems. pp. 2637–2646. ACM (2014)
14. captchas.net: Free captcha-service, `http://captchas.net/`
15. Dietz, M., Czeskis, A., Balfanz, D., Wallach, D.S.: Origin-bound certificates: a fresh approach to strong client authentication for the web. In: Presented as part of the 21st USENIX Security Symposium (USENIX Security 12). pp. 317–331 (2012)
16. Evans, C., Palmer, C., Sleevi, R.: Public key pinning extension for http. Tech. rep. (2015)
17. Ferraro Petrillo, U., Mastroianni, G., Visconti, I.: The design and implementation of a secure captcha against man-in-the-middle attacks. Security and Communication Networks **7**(8), 1199–1209 (2014)
18. Gao, H., Wang, X., Cao, F., Zhang, Z., Lei, L., Qi, J., Liu, X.: Robustness of text-based completely automated public turing test to tell computers and humans apart. IET Information Security **10**(1), 45–52 (2016)
19. Gao, S., Mohamed, M., Saxena, N., Zhang, C.: Emerging image game captchas for resisting automated and human-solver relay attacks. In: Proceedings of the 31st Annual Computer Security Applications Conference. ACSAC (2015)
20. Karapanos, N., Capkun, S.: On the effective prevention of tls man-in-the-middle attacks in web applications. In: 23rd USENIX Security Symposium (USENIX Security 14). pp. 671–686 (2014)
21. Karapanos, N., Marforio, C., Soriente, C., Capkun, S.: Sound-proof: Usable two-factor authentication based on ambient sound. In: 24th USENIX Security Symposium (USENIX Security 15). pp. 483–498 (2015)
22. Karlof, C., Shankar, U., Tygar, J.D., Wagner, D.: Dynamic pharming attacks and locked same-origin policies for web browsers. In: Proceedings of the 14th ACM conference on Computer and communications security. pp. 58–71. ACM (2007)
23. Kim, T.H.J., Huang, L.S., Perring, A., Jackson, C., Gligor, V.: Accountable key infrastructure (aki): a proposal for a public-key validation infrastructure. In: Proceedings of the 22nd international conference on World Wide Web. pp. 679–690. ACM (2013)
24. Osadchy, M., Hernandez-Castro, J., Hernandez, J., Gibson, S., Dunkelman, O., Pérez-Cabo, D.: No bot expects the deepcaptcha!
25. Roosa, S.B., Schultze, S.: Trust darknet: Control and compromise in the internet's certificate authority model. IEEE Internet Computing **17**(3), 18–25 (2013)
26. Shultze, S.: Diginotar hack highlights critical failures of our ssl web security model. https://freedom-to-tinker.com/blog/sjs/diginotar-hack-highlights-critical-failures-our-ssl-web-security-model (sep 2011)
27. Soghoian, C., Stamm, S.: Certified lies: Detecting and defeating government interception attacks against ssl (short paper). In: International Conference on Financial Cryptography and Data Security. pp. 250–259. Springer (2011)
28. Syta, E., Tamas, I., Visher, D., Wolinsky, D.I., Jovanovic, P., Gasser, L., Gailly, N., Khoffi, I., Ford, B.: Keeping authorities" honest or bust" with decentralized witness cosigning. arXiv preprint arXiv:1503.08768 (2015)
29. Szalachowski, P., Matsumoto, S., Perrig, A.: Policert: Secure and flexible tls certificate management. In: Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security. pp. 406–417. ACM (2014)

30. Von Ahn, L., Blum, M., Hopper, N.J., Langford, J.: Captcha: Using hard ai problems for security. In: International Conference on the Theory and Applications of Cryptographic Techniques. pp. 294–311. Springer (2003)
31. Wendlandt, D., Andersen, D.G., Perrig, A.: Perspectives: Improving ssh-style host authentication with multi-path probing. In: USENIX Annual Technical Conference. vol. 200 (2008)