ABSTRACT
We argue that the CAPTCHA in its current incarnation may be near the end of its useful life, and propose an alternative throttling mechanism to control access to web resources. We analyze our proposed solution against a collection of realistic adversaries and conclude that it is a viable approach.

As a result of potential independent value, we describe heuristic tools to identify cookie theft, machine cloning attacks, and DNS poisoning attacks.

Categories and Subject Descriptors

General Terms
Security

Keywords
access, CAPTCHA, cloning, cookie theft, DNS poisoning, malware, scripting, throttle, usability.

1. INTRODUCTION
The emergence and spectacular rise of unwanted bots on the Internet has turned CAPTCHAs [?, ?] into an almost ubiquitous component of the web. However, as CAPTCHAs are strengthened to withstand increasingly powerful automated attacks, they are also becoming harder and harder for people to solve. If this trend were to continue as it has for the last few years, we will soon reach a point at which the CAPTCHA as we know it will have outlived its productive life: It will simply become too hard for humans to solve. The increased use of I/O constrained devices (such as smartphones) may aggravate the problem and speed up these developments by hurting consumer tolerance when it comes to having to read and enter data.

However, there is reason for concern even if we were to witness the development of a truly ideal CAPTCHA – one that cannot be beaten using automated methods and which does not pose any usability problems. For example, CAPTCHAs from legitimate sites are commonly re-posted on other sites – porn sites, typically [?] – and their solutions are fed back by the attacker to the original sites. Moreover, CAPTCHA-solving sweatshops have proliferated [?] in developing nations during the last few years. While currently, these human-aided attacks may only contribute marginally to the total number of CAPTCHAs broken, it still begs the question of whether distinguishing between man and machine is really the best approach to control access.

We believe that CAPTCHAs are being assaulted from three directions: automated attacks, human-aided attacks, and degraded usability. The two first place an upper limit on the value of the resources CAPTCHAs can be used to protect1, while the third corresponds to a threshold of perceived user benefit, below which people will not be bothered with solving a CAPTCHA. Together, this poses a serious threat to CAPTCHAs, and we argue that there is the need for a viable option to the current approach. In this paper, we describe and analyze one possible CAPTCHA alternative.

Before we describe our solution, we need to detail what properties are necessary to successfully address the underlying access control problem. It is important to understand that we do not aim to distinguish between humans and machines – that is a simple artifact of the current approach to throttle access to online resources. On the other hand, we do need assurance that the adversary’s cost to create multiple valid requests during a specified time interval will be significant, and that the cost will increase with the number of requests. Moreover, we want the throttling system to exhibit a maximal degree of usability; ultimately, it should be entirely transparent to users. The system should be accessible to everybody using the web, and if possible, should not involve the installation of any client-side software. Finally, we want to minimize the degree of trust that has to be placed in the various operators, while taking advantage of the new structure to piggyback desirable system information.

We propose a solution that does not require any user involvement after an initial setup phase (except for in rare special cases.) Our solution relies on cookies and cookie alternatives, the latter which contributes an increased robustness against loss and theft of cookie information. We analyze our solution in the context of a collection of reason-

1The market price for breaking CAPTCHAs, as of November 2008, is $2 per 1000 CAPTCHAs [?].
able adversaries, many of which are influenced by currently known attacks on the advertising infrastructure [?, ?]. Our adversarial model includes human adversaries, script-based adversaries, and malware-powered adversaries.

While we are no strong believers of CAPTCHAs, we do believe that the best protection may be obtained by the use of a solution like ours in occasional combination with a traditional CAPTCHA. We show how such a hybrid approach can be used to drastically limit the number of successful automated accesses that an adversary can obtain from accounts he has registered or otherwise obtained access to.

As a building block of potential independent interest, we detail a technique that can be used to identify cookie theft, machine cloning attacks, and DNS poisoning attacks. Our technique is based on cookies and cache cookies, configured in a way that allows various anomalies to be detected.

**Outline.**

We begin with a brief survey of the related work (section 2). We then detail our adversarial model and describe the general requirements on an access throttle (section 3). We present our solution (section 4), including a hybrid approach that relies on the occasional invocation of a traditional CAPTCHA (section 4.3). We analyze the properties of our proposal (section 5), then finally, briefly discuss privacy issues.

2. RELATED WORK

The general notion of a proof of being human was proposed by Naor in 1996 [?]. Inspired by this general idea, and prompted by Manber [?], von Ahn et al. proposed practical instantiations in 2003, giving birth to the CAPTCHA [?] – or Completely Automated Public Turing test to tell Computers and Humans Apart. Three other early and independent implementations of the same concept were AltaVista’s bot detection technique [?] in 2001; the so-called Gausebeck-Levchin test, which was used by PayPal in the early 2000’s for similar purposes; and a 2001 proposal by Coates et al. [?].

The core idea behind all of these approaches is to leverage on problems that are known in the artificial intelligence community to be difficult for computers to solve – yet easy for humans – and build CAPTCHAs based on such problems. The most common type of underlying problem involves the recognition of partially obfuscated words or letter sequences, using obfuscation methods against which known Optical Character Recognition (OCR) methods fail.

The research on CAPTCHAs has been intense, both in terms of coming up with new methods (see e.g., [?, ?, ?, ?, ?]) and breaking existing CAPTCHAs (see e.g., [?, ?, ?, ?, ?]). At the same time as research has produced more robust CAPTCHAs, we have also seen the development of special-purpose CAPTCHAs – such as audio CAPTCHAs (see e.g., [?, ?]) and clickable CAPTCHAs (see e.g., [?, ?, ?]). A very appealing new type of CAPTCHA is the so-called reCAPTCHA [?]; this outsources the interpretation of words that automated methods failed to identify when library books are digitized. This approach combines known hardness guarantees with a utility that previous CAPTCHAs did not have: To derive benefit from the responses received. This is akin to the outsourcing of computational tasks (such as SETI@Home[?]) to solve large problems – although using the human as the harnessed resource.

To address the problems with CAPTCHAs, as outlined in the introduction, we propose an account-based throttling system where individual computers are associated with accounts, and their transactions logged and counted by a trusted third party. Fraud would be detected using heuristics very similar to those used to detect click-fraud [?, ?, ?]. The resemblances between advertising and throttling go deeper than in terms of the adversarial model and the detection methods: The structure we propose has notable similarities to the structure that underlies the DoubleClick system, an advertising system acquired and operated by Google. Given that our approach relies on authenticating registered accounts, it is valuable to keep in mind how these may be threatened by phishing and malware attacks (see [?, ?] for an overview of these types of attacks.) In particular, account takeovers and additions of machines controlled by adversaries to “honest” accounts may be realistic threats.

The notion of accounts was also studied by Shi et al. [?] in the context of DDoS protection, although using computational puzzles to control account creation. This is not a suitable approach to address the problem we study.

Since our approach utilizes cookies, it is important to recognize their limitations and vulnerabilities. In particular, our solution uses cache cookies [?] as a backup mechanism to obtain robustness against the loss of cookies. We propose and describe mechanisms to identify cookie theft [?]; machine cloning and DNS poisoning attacks.

Our solution involves a “translation component”, similar to that proposed in [?], to automatically convert webpages to make them comply with our proposed protocol. We also rely on techniques to establish access rights to accounts, such as bank accounts. In the early days of online payments, PayPal introduced a method by which users could prove that they had access rights to a given bank account by reporting to PayPal on the sizes of a collection of small payments made to their account by PayPal – this effectively outsourced the account verification to the banks, and was one of the enabling techniques behind PayPal’s now pervasive payment scheme. We propose a more secure and less costly variant of this mechanism, and describe how it can be used for bootstrapping purposes.

3. ADVERSARY AND REQUIREMENTS

Our goal is to construct a throttle that can be used to control access to web resources. It is the goal of the adversary to gain access to as much of these resources as possible, at the lowest possible cost. To this end, the adversary may carry out human-aided and automated attacks against the system, and combinations of these. We assume that the adversary does not control the communication infrastructure, or that the client and server have already built up a secure connection for their other communication preceding the throttle stage. (Therefore, we do not consider attacks in which the adversary uses eavesdropping to gain access to information that allows him to pass a throttle request.)

A human-aided attack involves a collection of users who are coerced to perform throttle tasks (and potentially other tasks as well) requested by the adversary. In addition, it may involve a network of proxies that forward requests to hide the IP address of the coerced human.

An automated attack may consist of one or more components: a computational attack; a Sybil attack; a scripting attack; and a malware attack.
A computational attack involves performing some computation to complete throttle tasks. This has been and still is the main threat against existing CAPTCHAs.

A Sybil attack is a technique by which the adversary creates a large number of user accounts, and with this, a perception of a large number of users – all of which the attacker controls and assigns throttle tasks.

A scripting attack is one in which a user visits a webpage controlled by the adversary, and is sent a script (such as a JavaScript snippet) that – by being evaluated on the user’s machine – attempts to perform a throttle task on behalf of the adversary.

A malware attack, finally, may involve an attempt to clone the contents of the victim’s machine in order to have the clone carry out throttle tasks at the whims of the adversary. It may also involve an attempt to gain long-term control over the victim’s machine and use this machine as a zombie to perform throttle tasks.

We must also worry about an attack on the throttle infrastructure. An adversary can simply initiate a large enough number of throttle requests that the trusted third party used in our approach becomes overwhelmed. With a powerful trusted third party – run by a large organization, such as Google, Yahoo or Akamai – this concern is not so great as they are already facing the risk of DoS attacks. If run by a smaller organization, the risk is real, but can be addressed using a fallback onto traditional CAPTCHAs for requests that time out. In many situations, another request can be performed later during the session to prevent that denial of service attacks are successfully used to circumvent the protection.

Formalizing the adversary.

We let \( n_a \) be the number of machine-specific accounts that the adversary controls – whether by having created them himself (a Sybil attack); by coercing the account owners to act on his behalf; by gaining complete control over user machines; or a combination of these.

Furthermore, we let \( n_c \) be the number of machine-specific accounts that the adversary gained access to, but to which the legitimate users still also have access to. We assume that these users are not coerced, and that their actions cannot be controlled by the adversary. This corresponds to an attack in which user machines are (at least partially) cloned by the adversary, whether by stealing human-machine credentials (such as account passwords) or by stealing machine-machine credentials (such as cookies).

We let \( T \) be the threshold of requests, above which an account would be flagged for a given time interval. This is admittedly a very rough characterization of the fraud-detection heuristics used, but allows a simple and still meaningful security statement. With a more complex policy, the threshold would be replaced by a function. (With a secret or randomized policy, it is harder for the adversary to maximize his success, assuming that he loses access to accounts that are flagged as being corrupted.)

Furthermore, we let \( t \) be the average number of accesses possible to perform by an adversary using an account to which he does not have sole access to, before the account owner attempts to access the account or the account is otherwise flagged. Clearly, we have that \( t \leq T \).

The maximum number of throttle tasks that can be solved by the adversary within the considered time interval is:

\[
 n_a T + n_c t
\]

This corresponds to making \( T \) accesses from each one of the \( n_a \) accounts the adversary controls, and \( t \) accesses (on average) from each account to which he has access but which he does not control.

Requirement.

It is our goal to provide a solution that implements the policies reflected by the above access thresholds. In other words, our goal is to guarantee that an adversary controlling \( n_a \) accounts, and with access to \( n_c \) accounts, cannot succeed with more than an expected \( n_a T + n_c t \) throttle requests in one time period, with \( T \) and \( t \) defined as above.

4. SOLUTION

4.1 Overview

Our solution involves three types of parties – users, sites and one or more trusted third parties. The users visit sites; the sites wish to throttle access to resources; the trusted third parties keep track of accesses on a per-user basis and identify potential abuses and account takeovers using back-end heuristics.

Our solution has four principal components: setup, throttle, recovery, and detection. These are overviewed below:

Setup (Overview).

In the setup phase, the user first creates a “throttle account” with a trusted third party, and links it to one or more restricted resources to which the user has access – these resources have the simple property of being hard to obtain in large numbers. He then ties the throttle account to one or more computers (this can be done over a longer period of time.) This is done marking these machines (also referred to as tagging). The traditional approach to mark machines is to place cookies on them; we will go beyond this, though, for practical purposes associated with the acceptability and survivability of cookies. The marking of machines allow the trusted third party later to identify machines and determine what account they correspond to.

Throttle (Overview).

In the throttle phase, the user is visiting a site that wishes to throttle access. (This may be a site that the user has never before accessed.) The site then causes the user’s machine to be identified by a trusted third party. If the trusted third party fails to identify the machine, the recovery step is performed. On the other hand, if the machine is positively

\(^3\) Examples of restricted resources include bank accounts and other money transfer accounts, phone numbers and residential mailing addresses. While a person can have three or four financial accounts, phone numbers or mailing addresses, he will typically not have thousands – at least not without unusual efforts and investments.

\(^2\) One throttle account may have several user machines associated with it; we count each such corrupted machine.
identified, then the trusted third party increases a count\(^4\) determining the number of accesses made during the associated time period. If this is a reasonable number (different sites may have different thresholds) then the trusted third party signals success to the site (using the user machine as the messenger). If not, then the trusted third party may communicate failure to the querying site and flag the user account.

**Recovery (Overview).**

The recovery step is invoked when the trusted third party fails to identify a user account. This might be due to one of several reasons: (1) The user has an account with another trusted third party only; (2) The user has an account with the current trusted third party, but his machine has lost the cookie and cache cookies used to identify it; or (3) the user has no account with a trusted third party.

To address the first possibility, one can determine whether the user has an account with another authority simply using a traffic redirect. (For simplicity, we will assume that there is only one trusted third party onwards.)

To deal with the second alternative, which is a rare special case, the user may be prompted to supply a user name or other account identifier. He may then either be prompted for a password or other credential, or otherwise be challenged to prove access to a resource (such as an email account) already associated with his account. Once he has established his identity with the trusted third party, the cookies and cache cookies will be re-established, and the recovery phase is ended.

To recover from the third possibility, one may either ask the user to establish an account, or may demand that he solves a traditional CAPTCHAs. If the latter approach is taken, then the trusted third party could communicate this to the calling site, so that the site can determine (based on some risk assessment policy) how to treat the user’s access request. We note that users who do not wish to opt in to our proposed solution would have to solve traditional CAPTCHAs in order to get access to a site. These may be harder than today’s CAPTCHAs, and the resulting sessions may receive closer scrutiny than the sessions initiated using more secure solutions.

Due to lack of space and the relative straightforwardness of the recovery process, we do not provide details of this step in this paper.

**Detection – Overview.**

The detection step is a heuristic back-end operation run by the trusted third party, with the goal of identifying, flag and block suspect accounts and throttle requests. This step involves efforts to detect cookie theft, machine cloning and DNS poisoning, in addition to the detection of excessive access from a small number of accounts.

### 4.2 Detailed Solution

We will now offer more details of our solution; we note that there are many possible variants, and we will not describe all. Rather, we will focus on the main structure, and only provide a few example instantiations, where applicable. Developing the most suitable and usable instantiations is far from straightforward, though, and we want to emphasize the need for further research in this area.

**Setup.**

The setup can be done in many ways; in essence, a user establishes access rights to a resource (such as a bank account), and ties this resource to a “throttle account”. The user can then tie one or more machines to the throttle account; as this is done, each machine is marked in a manner that allows it to be identified by the trusted third party later on.

- **Bootstrapping to a Restricted Resource\(^5\).** To prove access rights to a bank account, the user can provide the account number to the trusted third party, who then sends a one-cent payment to the account, along with a message\(^6\). The message will be a pseudo-random alphanumeric string that cannot be guessed with a non-negligible probability by an attacker. To complete registration, the user would provide the trusted third party with this string (e.g., using cut-and-paste from his online banking account), which effectively proves that he has access to the account. An analogous approach can be taken using SMS messages sent to phones, or PayPal payments with associated messages – here, the restricted resources are phones and PayPal accounts – both of which have already been tied to some financial resource.

- **Marking a Machine.** It is straightforward to mark a machine if we assume that users may download plugins or other executables. However, this is likely to decrease the the opt-in rate, require larger development and maintenance efforts, and elevate the exposure to malware in society.

As a better alternative, cookies are a simple and efficient way of marking machines. Cookies, however, are commonly cleared by users – and some types of software. Cache cookies\(^7\) are a good backup measure, and may use the browser history and various types of browser caches (e.g., images, flash objects, etc.) to store data. Cache cookies, however, are slower to read and write than regular cookies. The best approach, we believe, is a new and combined structure – we refer to this as a “reinforced cookie”. Such a reinforced cookie can be written by encoding an identifier using each one of the underlying cookie types, writing each one of these to the target machine. To read a reinforced cookie, all the various types of cookies can be attempted to be read at once\(^8\); if any attempt is successful, then we perform a complete rewrite to maximize the survivability of the reinforced cookie.

To allow detection of inconsistencies, the identifier is replaced with a new unique identifier each time it is

\(^4\)More detailed logs can also be kept. This information can be used to identify potential abuse.

\(^5\)We do not attempt to describe all possible bootstrapping methods, as it is enough to provide evidence that there are enough meaningful resources to use that everybody can find a way of completing this step.

\(^6\)All current bank payments can be accompanied by messages; these are typically used to signal the purpose of the payment.

\(^7\)The traditional cookie is fastest to read, followed by the browser cache cookie, in turn followed by the browser history cookie\(^?\).
rewritten. To simplify distributed record keeping, the identifier could be of a format \((id, \text{count}, \text{type}, \text{auth})\). Here, \(id\) is a unique and static identifier of the throttle account, \(\text{count}\) is a monotonically increasing counter associated with the throttle account, \(\text{type}\) identifies the type of cookie (e.g., HTML cookie, HTML SSL cookie, cache cookie, or browser history cookie.) Finally, \(\text{auth}\) is a MAC of \((id, \text{count}, \text{type})\) using a key known to the trusted third party only.

In particular, we may use a marking technique that sets one HTML or HTML/SSL cookie, one cache cookie and one browser history cookie for each IP address\(^8\) used by the trusted third party, in addition to one HTML cookie, one cache cookie and one browser history cookie associated with the domain of the trusted third party. Each one of these is considered a separate \(\text{type}\), and therefore, obtains a different encoding of the identifier. This redundant marking technique is used to help identify cookie theft, machine cloning attacks, and DNS poisoning attacks.

**Throttle.**

The throttle step has three components: a translation component, a verification component, and an access component:

**Throttle: Translation Component.**

The translation component takes a collection of standard webpages\(^9\) as input, along with a description of what portions and operations should be protected by the throttle, and outputs a new collection of webpages that has the same apparent functionality as the input set – to a user whose machine passes the throttle.

Let \(W = \{W_i\}\) be a set of input webpages. We assume that \(W_1\) is the landing page\(^10\), and that pages \(W_i, i > k\) for some value \(k\), must only be accessed by a user whose machine passes the throttle test. The translation component produces a set of webpages \(W' = \{W'_i\}\) such that \(W'_i\) and \(W_i\) have the same appearance and functionality when rendered on a machine that passes the throttle evaluation.

This translation is run for each session. (For simplicity of exposition, we also produce a new instance in the case where a user reloads the landing page without having ended the session before.)

1. **Session identifier**: A session identifier \(id\) is selected, as described in the “machine marking” paragraph above.

2. **Landing page**: An iframe is added to \(W_1\) to create \(W'_1\); this iframe references content from a URL \(T/id/req\), where \(T\) is a landing page of the trusted third party, \(id\) is the session identifier, and \(req\) may contain special requests in terms of throttling policies. The iframe will also contain HTML code \(\text{CC}\) that causes a release\(^11\) of cache and history cookies to the trusted third party – if available on the user machine. (See \([?]\) for details.)

3. **All pages in \(W\)**: Any reference to a page \(W_i\) for \(i > k\) is replaced by a reference to \(W_i/id\), e.g., the URL address is augmented with the session identifier.

**Throttle: Verification Component.**

The communication component of the throttle can be designed as follows:

1. **Visit to landing page**. The user loads a webpage \(W_1\), managed by a site \(S\).

2. **Connecting to the trusted third party**. The user machine, prompted by \(W_1\), loads \(T/id/req\), thereby transmitting \((id, req)\) to the trusted third party. At the same time, any cookies set by the trusted third party will be transmitted. The user machine also attempts to render \(\text{CC}\), which causes cache cookies and browser history cookies to be sent to the trusted third party, if present. (Due to the delay of rendering, these strings will be communicate with a lag after the page request and the standard cookie.) If no cookies, cache cookies or browser history cookies are received by the trusted third party, then the recovery will be initiated, and the throttle decision set according to the outcome of this step.

3. **Response generated**. The trusted third party computes a response \(R\) containing \(id\); optional auxiliary information to be transmitted to \(S\); and a throttle decision that is relative to \(req\), the success of the read of the cookie (or other mark) from the user machine, and the outcome of the recovery phase (when applicable). The trusted third party computes \(\mu = MAC_{K_S}(R)\), where \(K_S\) is a key known\(^12\) only by \(S\) and the trusted third party. The user machine is redirected to a page\(^13\) \(W_1/R/\mu\) managed by \(S\). This effectively transmits \((R, \mu)\) to \(S\), using the user machine as an intermediary.

4. **Interpreting responses**. \(S\) verifies \(\mu = MAC_{K_S}(R)\).

   If not, then the throttle request is considered to fail. \(S\) determines what action to take based on the response \(R\); if and only if the throttle request succeeded, then any page request containing \(id\) will be accepted for the duration of the session. To achieve this, \(id\) is entered in a database of accepted identifiers – this database is distributed among all the servers of \(S\). (Contents of

\(^8\)Setting the HTML cookies for all the IP addresses used may be a rather cumbersome activity; in order not to have to perform this for each rewrite operation, these cookies can refreshed less frequently than the other types. Cache cookies and browser history cookies can be set for a large number of IP addresses by one and the same entity, though. We refer to \([?]\) for a detailed description of how to do this.

\(^9\)For the sake of simplicity, we assume that all of them are hosted on the same domain, although we do permit load balancing between different servers.

\(^10\)For simplicity, but without loss of generality, we assume there is only one landing page.

\(^11\)If a very large number of IP addresses are used by the trusted third party, then this is not a practical approach, and it may be better for the selected server, when contacted, to request the appropriate IP-based cookie objects.

\(^12\)We assume that this key has been exchanged between the parties in a secure manner when \(S\) established an account with the trusted third party.

\(^13\)It does not matter that a page with this address does not exist; the purpose of the request is to communicate information. If the iframe added to \(W_1\) is zero-sized, the user would not even know how the request was resolved – or that it even took place.
Throttle: Access Component.

Whenever a webpage \( W^i_j \), \( i > k \) is requested (using a URL \( W^i_j/id \)), it is determined whether \( id \) is an accepted verifier; if it is, then the page is served; if not, then the user may be served an error message.

A remark about delays.

We note that the access can often be made to appear instantaneous, even if the verification causes a slight delay. For example, if a user has not yet been granted access to a search interface, he may still be presented with the search bar and allow to enter his search phrase; as long as the verification completes before the user finished the text entry, the user will not perceive any delay at all.

Detection.

Our solution incorporates detection mechanisms to identify cookie theft, session replay, malware-based cloning attacks, and excessive access (which is a symptom of account takeover).

- Identifying Cookie theft. When considering this attack, we assume that the attacker does not have full read access to the victim machine, and that he cannot access to the communication between the victim machine and the trusted third party. (The former rules out malware corruption, and the latter rules out an active man-in-the-middle attack.)

Unless DNS has been poisoned, cache cookies and browser history cookies can only be read by a party who knows the name of the cached item resp. URL used to encode the information stored (see [?]). We assume that the attacker has not corrupted the DNS of the victim (that case is dealt with separately below). Thus, the attacker may be able to obtain the domain-based HTML cookie from the victim machine, but no other cookie elements. Consequently, if the trusted third party receives only a domain-based HTML cookie from a client, that is a sign\(^{14}\) of cookie theft.

- Identifying DNS poisoning. As above, we assume that the attacker does not have full read access to the victim machine, and that he cannot access the communication between the victim machine and the trusted third party.

If the victim machine’s DNS has been poisoned, then all domain-based cookies will be readable to the attacker; however, and based on the same argument as above, he will not have read access to any of the IP-based cookie objects. Thus, if the trusted third party receives domain based cookie objects but not IP based objects from a client machine, that is a sign of likely DNS poisoning.

14Another possible explanation is that the user has cleared his cache and browser history, and erased the IP-based HTML cookie – but not the domain-based HTML cookie. This appears rather unlikely.

- Identifying Machine cloning. By cloning a victim machine, the attacker will have access to all the cookie elements, and will be able to produce a perfect set of cookie objects to an interrogating server of the trusted third party. However, as the attacker’s machine will receive a rewritten set of updated cookie objects, the victim machine will have a now-outdated set of cookie elements. When he contacts a service provider that initiates a throttle mechanism, then the trusted third party will receive the outdated cookie elements\(^{15}\). This is a clear sign of previous cloning. Whereas it may not be possible to identify who the rightful account owner is, the account can quite simply be flagged.

- Identifying excessive access. To detect excessive access, we need to correctly identify the machines used to access a site. This is done by verifying consistency, freshness and the number of cookie elements submitted, along with the associated originating IP addresses, request types, and the nature and count of the previous access requests from the associated account\(^{16}\).

To verify consistency, the trusted third party checks that the identifier of the cookie elements received for a given session all encode the same identifier, but for the differing values of the type variable. Freshness is determined by verifying that the received cookie elements encode an identifier that has not expired. An expired identifier is one that corresponds to an old value of count for the throttle account. These values of count are synchronized between servers of the trusted authority in batches; whereas servers are not guaranteed to have the most recent value at any time, it is very unlikely that a server will fail to detect the replay of a cookie that is more than one or two replacements old. Finally, the trusted third party determines whether the number of throttle requests (and the associated contexts of these) suggests fraud. We do not detail this step in the paper, but note that the mechanisms are very similar to current mechanisms used to detect click-fraud, credit card fraud, and phone fraud.

4.3 Integrating Traditional CAPTCHAs

It is possible to strengthen our approach using the occasional use of a traditional CAPTCHA. Here, it is important to note that the traditional CAPTCHA can be relatively weak, and therefore, reasonably easy to solve. This would avoid that the difficulty of passing the throttle task is escalated over time, as is the problem with traditional CAPTCHAs in isolation.

Human failure rates of about 15% are commercially tolerable [?], and bot success rates of up to 45% are reported [?] for well designed CAPTCHA systems. For the sake of an argument, and erring on the safe side, we may assume worst-case human failure rates of 25% and bot success rates of 50%. A human will fail 375 out of 1000 consecutive tests.

\(^{15}\)If the throttle authority receives the same mark twice in a row, this could be a sign of machine cloning, but may also be due to a failed rewrite of cookie elements to an honest machine. Old marks can only be sent as a result of attack, or the rather uncommon rewinding of state associated with the use of a backed-up state, e.g., due to a crash.

\(^{16}\)This is a simplified description of the data to be analyzed; a real implementation would use other information as well, identifying links between corrupted accounts, etc.
with a probability of less than $2^{-45}$, whereas the adversary will pass the same tests with that same probability.

Therefore, in the account model that we propose, it is possible to detect abuse over time due to the vastly differing cumulative probabilities of success between humans and bots. In contrast, the use of such statistical success models is not meaningful in an account-less model where the adversary has access to a tremendous number of machines and it can simply ditch blacklisted machines. This is the current situation for traditional CAPTCHAs.

We may therefore note that it is necessary to involve a human in the loop for an attack of our system to be successful. However, it is not sufficient, as the attacker also has to establish or take over accounts, both of which has a substantial cost.

5. ANALYSIS

Recall that our goal is to guarantee that an adversary controlling $n_a$ accounts, and with access to $n_a$ accounts, cannot succeed with more than an expected $n_aT + n_aT$ throttle requests in one time period, with $T$ is a threshold set by the trusted third party and $t$ is the average number of throttle requests the adversary can make from an account he has gained access to before the legitimate account owner produces a throttle request. In order to reach this goal, the following conditions have to be met:

1. The adversary cannot output a valid mark of a computer he has not successfully attacked, but with a negligible probability.

2. The adversary cannot read the complete mark from an uncorrupted machine, and can only read portions of the mark from a machine that he has successfully attacked using cookie theft or DNS poisoning. (We assume that the adversary does not have read access to the communication path between a user machine and the trusted third party.)

3. The mark of a machine will cease to be valid after the machine has generated a number of valid throttle requests (associated with this mark) exceeding a threshold set by the trusted third party. (This threshold, which can depend on the commercial utility associated with a given machine, will be set high enough not to lock out legitimate users, whose access patterns are vastly different from that of typical attackers.)

4. The mark of a cloned machine will cease to be valid as soon as throttle access requests representing the cloned machine have been made by both the cloned machine and the adversary.

The first condition is met due to the believed computational impossibility to forge valid MACs without knowledge of the associated secret key, and our assumption that the attacker cannot eavesdrop on the communication infrastructure.

The second condition is met due to use of IP-bound cookie elements, which cannot be stolen or read without having corrupted the machine on which they are stored. Together, these two conditions imply that the adversary can only produce complete marks from computers he fully corrupts; there are $n_a$ of these.

The third condition is met based on the assumption that different servers of the trusted third party will have a synchronized view of the counter associated with a given throttle account (relaxing the assumption only slightly affects the bound; this is not done for the sake of simplicity). Thus, the adversary can only make $T$ accesses from each of the $n_a$ machine-specific throttle accounts that he has complete control of.

The fourth condition is met since the mark of a machine is updated as a result of a successful throttle request, and the reuse of expired marks will cause an account to be flagged. This contributes $t$ accesses for each of the $n_a$ machine-specific accounts that the adversary has cloned.

We therefore believe that our solution offers a good alternative to the current types of CAPTCHAs, and hope that our effort will be the first in a series of steps to address the problem of vulnerable CAPTCHAs by approaching the problem from a new direction.

6. PRIVACY CONSIDERATIONS

On one hand, one may argue that the proposed solution and associated infrastructure is no greater threat to a user’s privacy than a search engine, ISP gateway, or DNS server – to name a few – and that the benefits associated with our proposal would outweigh the drawbacks. On the other hand, it is also of value to consider how the privacy implications can be limited to an absolute minimum. Instead of machines identifying themselves using cookies and related mechanisms, one could imagine an alternative approach in which a the client machine would generate a digital signature whose corresponding public key has been certified by the trustee using a blind signature protocol, and where a new public key is generated and certified with some preset frequency. This simple construction would allow the trustee to determine how many access attempts a given client machine have initiated within the applicable time frame. This allows the trustee to turn down a request if too many have already been made by a given party. We note that, in contrast to anonymous e-payment schemes, there is no need to unblind the signature or attempt to trace the signer in our context. Camenisch et al. [7] describe a more intricate solution that allows a trustee to verify that no user makes more than some $k$ accesses, but without being able to link legitimate accesses.