Not-a-Bot: Improving Service Availability in the Face of Botnet Attacks

Ramakrishna Gummadi*, Hari Balakrishnan*, Petros Maniatis†, Sylvia Ratnasamy†
*MIT CSAIL, †Intel Research Berkeley

Abstract

A large fraction of email spam, distributed denial-of-service (DDoS) attacks, and click-fraud on web advertisements are caused by traffic sent from compromised machines that form botnets. This paper posits that by identifying human-generated traffic as such, one can service it with improved reliability or higher priority, mitigating the effects of botnet attacks.

The key challenge is to identify human-generated traffic in the absence of strong unique identities. We develop NAB ("Not-A-Bot"), a system to approximately identify and certify human-generated activity. NAB uses a small trusted software component called an attester, which runs on the client machine with an untrusted OS and applications. The attester tags each request with an attestation if the request is made within a small amount of time of legitimate keyboard or mouse activity. The remote entity serving the request sends the request and attestation to a verifier, which checks the attestation and implements an application-specific policy for attested requests.

Our implementation of the attester is within the Xen hypervisor. By analyzing traces of keyboard and mouse activity from 328 users at Intel, together with adversarial traces of spam, DDoS, and click-fraud activity, we estimate that NAB reduces the amount of spam that currently passes through a tuned spam filter by more than 92%, while not flagging any legitimate email as spam. NAB delivers similar benefits to legitimate requests under DDoS and click-fraud attacks.

1 Introduction

Botnets comprising compromised machines are the major originators of email spam, distributed denial-of-service (DDoS) attacks, and click-fraud on advertisement-based web sites today. By one measure, the current top six botnets alone are responsible for more than 85% of all spam mail [23], amounting to more than 120 billion messages per day that infest more than 95% of all inboxes [14, 24]. Botnet-generated DDoS attacks account for about five percent of all web traffic [9], occurring at a rate of more than 4000 distinct attacks per week on average [17]. A problem of a more recent vintage, click-fraud, is a growing threat to companies that draw revenue from web ad placements [26]; bots are said to generate 14–20% of all ad clicks today [8].

As a result, if it were possible to tag email or web requests as "human-generated," and therefore not "bot-generated," the problems of spam, DDoS, and click-fraud could be significantly mitigated. This observation is not new, but there is currently no good way to obtain such tags *automatically* without explicit human input. As explained in §4, requiring human input (say in the form of answering CAPTCHAs [30]) is either untenable (persuading users to answer a CAPTCHA before clicking on a web ad or link is unlikely to work well), or ineffective (e.g., because today the task of solving CAPTCHAs can be delegated to other machines and humans, and not inextricably linked to the request it is intended to validate).

The problem with obtaining this evidence automatically is that the client machine may have been compromised, so one cannot readily trust any information provided by software running on the compromised machine. To solve this problem, we observe that almost all commodity PCs hitting the market today are equipped with a Trusted Platform Module (TPM) [28]. We use this facility to build a trusted path between physical input devices (the keyboard and mouse, extensible in the future to devices like the microphone) and a *human activity attester*, which is a small piece of trusted software that runs isolated from the (untrusted) operating system.

The key challenge for the attester is to certify human-generated traffic without relying on strong unique identities. This paper describes NAB, a system that implements a general-purpose human activity attester (§4), and then shows how to use this attester for email to control spam, and for web requests to mitigate DDoS attacks and click fraud. Attestations are signed statements by the trusted attester, and are attached to application requests such as emails. Attestations are verified by a *verifier* module running at the server of an entity interested in knowing whether the incoming request traffic was sent as a result of human activity. If the attestation is valid (i.e., it is not forged or used before), that server can take suitable application-specific action—improving the "spam score" for an attested email message, increasing the priority of an attested web request, etc. NAB requires minor modifications to client and server applications to use attestations, and no application protocols such as SMTP or

HTTP need be modified.

NAB's philosophy is to do no harm to users who do not deploy NAB, while benefiting users who do. For example, email senders who use the attester decrease the likelihood that their emails are flagged as spam (that is, decrease the *false positives* of spam detectors), and email receivers that use the verifier see reduced spam in their inboxes. These improvements are preserved even under adversarial workloads. Further, since NAB does not use identity-based blacklisting or filtering, legitimate email from an infected machine can still be delivered with valid attestations.

The NAB approach can run on any platform that provides for the attested execution of trusted code, either directly or via a secure booting mechanism such as those supported by Intel's TXT and AMD's Pacifica architectures. We have constructed our prototype attester as host kernel model running under a trusted hypervisor. Other implementations, such as building the attester within the trusted hardware, or running it in software without virtualization (e.g., via Flicker [16]) are also possible.

Our prototype extends the Xen hypervisor [3], thus isolating itself from malicious code running within untrusted guest operating systems in a virtual machine. We stripped the host kernel and Xen Virtual Machine Monitor (VMM) down to fewer than 30,000 source lines, including the necessary device drivers, and built the attester as a 500-line kernel module. This code, together with the TPM and input devices forms the trusted computing base (TCB). Generating an attestation on a standard PC takes fewer than 10^7 CPU cycles, or less than 10 ms on a 2 GHz processor, making NAB practical for handling fine-grained attestation requests, such as individual web clicks or email messages.

We evaluate whether NAB can be applied to spam control, DDoS defense, and click-fraud detection, using a combination of datasets containing normal user activity and malicious bot activity. We used traces of keyboard and mouse activity from 328 PCs of volunteering users at Intel gathered over a one-month period in 2007 [11], packet-level traces of bot activity that we gathered from a small number of "honeypot" computers infected by malware at the same site, as well as publicly available traces of email spam and DDoS activity. On top of those traces, we constructed an adversarial workload that maximizes the attacker's benefit obtained under the constraints imposed by NAB. Our experimental study shows that:

1. With regards to spam mitigation, we reduced the volume of spam messages that evaded a traditional spam filter (what are called *false negatives* for the spam filter) by 92%. We reduced the volume of legitimate, non-spam messages that were misclassified by the spam filter (false positives) to 0.
2. With regards to web DDoS mitigation, we deprioritized 89% of bot-originated web activity without impacting human-generated web requests.
3. With regards to click-fraud mitigation, we detected bot-originating click-fraud activity with higher than 87% accuracy, without losing any human-generated web clicks.

Although our specific results correspond only to our particular traces, choice of applications, and threat model (e.g., NAB does nothing to mitigate the volume of evil traffic created manually by an evil human), we argue that they apply to a large class of on-line applications affected by bot traffic today. Those include games, brokerage, and single sign-on services. This suggests that a human activity attestation module might be a worthwhile addition to the TCB of commodity systems for the long term.

2 Threat Model and Goal

Threat model and assumptions. We assume that the OS and applications of a host cannot be trusted, and are susceptible to compromise. A host is equipped with a TPM, which boots the attester stack—this includes all software on which the attester implementation depends, such as the host kernel and VMM in our implementation (§4.4). This trust in the correct boot-up of the attester can be remotely verified, which is the standard practice for TPM-assisted secure booting today. We assume that the users of subverted hosts may be lax, but not malicious enough to mount hardware attacks against their own machine's hardware (such as shaving the protective coating off their TPM chip or building custom input hardware). We assume correct hardware, including the correct operation and protection of the TPM chip from software attacks, as per its specification [28]. We make no assumptions about what spammers do with their own hardware. Finally, we assume that the cryptographic primitives we use are secure, and that their implementations are correct.

Goal. NAB consists of an attester and a verifier. Our primary goal is to distinguish between bot and human-generated traffic at the verifier, so that the verifier can implement application-specific remedies, such as prioritizing or improving the delivery of human traffic over botnet traffic. We would like to do so without requiring any user input or imposing any cognitive burden on the user.

We aim to bound the final botnet traffic that manages to bypass any measures put up against it (spam and DDoS filters, click fraud detectors, etc.). We will consider our approach successful if we can reduce this botnet traffic that evades our best approaches today to a small fraction of its current levels (≈ 10%), even in the worst case for NAB (i.e., with adaptive bots that modulate their behavior to gain the maximum benefit allow-able by our mechanism), while still identifying all valid human-generated traffic correctly. We set this goal because we do not believe that purely technical approaches such as NAB will completely suppress attack traffic such as spam, since spam also relies on social engineering. We demonstrate that NAB achieves this goal with our realistic workloads and adaptive bots (§6).

3 NAB Architecture

We now present the requirements and constraints that drive the NAB architecture.

3.1 Requirements and Constraints

Requirements. There are four main requirements. First, attestations must be generated in response to human requests automatically. Second, such attestations must not be transferable from the client on which they are generated to attest traffic originating from another client. Third, NAB must benefit users that deploy it without hurting those that do not. Fourth, NAB must preserve the existing privacy and anonymity semantics of applications while delivering these benefits.

Constraints. NAB has two main constraints. First, the host's OS or applications cannot be trusted. In particular, a compromised machine can actively try to subvert the attester functionality. Second, the size of the attester TCB should be small, because it is a trusted component; the smaller a component is, the easier it is to validate it operates correctly, which makes it easier to trust.

Challenge. The key challenge is to meet these requirements without assuming the existence of globally unique identities. Even assuming a public-key infrastructure (PKI), deploying and managing large-scale identity systems that map certificates to users is a daunting problem [4].

Without such identities, the requirements are hard to meet, and, in some cases, even seemingly in conflict with each other. For example, generating attestations automatically without trusting the OS and applications is challenging. Further, there is tension between the requirement that NAB should benefit its users without hurting other users, and the requirement that NAB should preserve the existing anonymity and privacy semantics. NAB's attestations are anonymously signed certificates of requests, and the membership size of the signing keys is several million. We describe how NAB uses such attestations to overcome the absence of globally unique identities in §4.4.

TPM background. The TPM is a small chip specified by the Trusted Computing Group to strengthen the security of computer systems in general. A TPM provides many security services, among which the ability to measure and attest to the integrity of trusted software running on the computer at boot time. Since a TPM is too slow to be used routinely for cryptographic operations such as signing human activity, we use the TPM only for its secure bootstrap facilities, to load an *attester*, a small trusted software module that runs on the host processor and generates attestations (i.e., messages asserting human activity).

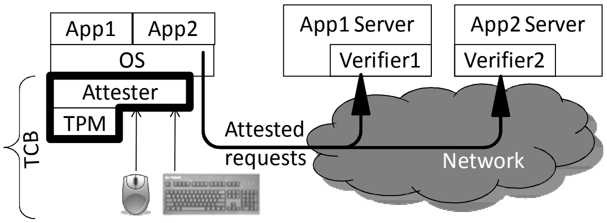

Figure 1: NAB architecture. The thick black line encloses the TCB.

The attester relies on two key primitives provided by TPMs. The first is called *direct anonymous attestation* (DAA), which allows the attester to sign messages anonymously. Each TPM has an *attestation identity key* (AIK), which is an anonymous key used to derive the attester's signing key. The second primitive is called *sealed storage*, which provides a secure location to store the attester's signing key until the attester is measured and launched correctly.

3.2 Architecture

NAB consists of an attester that runs locally at a host and generates attestations, as well as an external verifier that validates these attestations (running at a server expected to handle spam and DDoS requests, or checking for click fraud). The attester code hashes to a well-known SHA-1 value, which the TPM measures at launch. The attester then listens on the keyboard and mouse ports for human activity clicks, and decides whether an attestation should be granted to an application when the application requests one. If the attester decides to grant an attestation, the application can submit the attestation along with the application request to the verifier for human activity validation. The verifier can confirm human activity as long as it trusts the attestation TCB, which consists of the attester, the TPM, and input device hardware and drivers. This architecture is shown in Figure 1.

Attestations are signed messages with two key properties that enable the verifier to validate them correctly:

1. **Non-transferability.** An attestation generated on a machine is authenticated by a chain of signing keys that pass through that machine's TPM. Hence, a valid

attestation cannot be forged to appear as if it were issued by an attester other than its creator, and no valid attestation can be generated without the involvement of a valid attester and TPM chip.

2. **Binding to the content of a request.** An attestation contains the hash digest of the content of the request it is attesting to. Since an attester generates an attestation only in response to human activity, this binding ensures that the attestation corresponds to the content used to generate it. Binding thus allows a request to be tied as closely as practical to the user's intent to generate that request, greatly reducing opportunities for using human activity to justify unrelated requests.

4 Attester Design and Implementation

Our attester design assumes no special hardware support other than the availability of a TPM device. However, it is flexible enough to exploit the recent processor extensions for trusted computing such as AMD's Secure Virtual Machine (SVM) or Intel's Trusted Execution Technology (TXT) to provide additional features such as late launch (i.e., non boot-time launch), integration into the TCB of an OS, etc., in the future.

The attester's sole function is to generate an attestation when an application requests one. An attestation request contains only the application-specific content to attest to (e.g., the email message to send out). The attester may provide the attestation or refuse to provide an attestation at all. We discuss two important decisions: when to grant an attestation and what to attest.

4.1 When To Grant An Attestation

The key question in designing the attester is deciding under what conditions a valid attestation must be granted. The goal is to simultaneously ensure that human-generated traffic is attested, while all bot-generated traffic is denied attestation.

The attester's decision is one of guessing the human's presence and intent: was there a human operating the computer, and did she really intend to send the particular email for which the application is requesting an attestation? Since the attester lacks a direct link to the human's intentions, it must guess based on the trusted inputs available: the keyboard and mouse. We considered three key design points for such a guessing module.

The best-quality guess is not a guess at all: the attester could momentarily take over the keyboard, mouse, and display device, and prompt the user with a specific question to attest or not attest to a particular email. Since the OS and other applications are displaced in the process, only the human user can answer the question. From the interaction point of view, this approach is similar to the User Account Control (UAC) tool in Microsoft Windows Vista, in which the OS prompts the user for explicit approval before performing certain operations, although in our context it would be the much smaller and simpler attester that performs that function. While technically feasible to implement, users have traditionally found explicit prompts annoying in practice, as revealed by the negative feedback on UAC [29]. What is worse, user fatigue inevitably leads to an always-click-OK user behavior [32], which defeats the purpose of attestation.

So, we only consider guesses made automatically. In particular, we use implicit guessing of human intent, using *timing* as a good heuristic: how recently before a particular attestation request was the last keyboard or mouse activity observed? We call this a "$t - \delta$" attester, if δ_m denotes the time since the last mouse activity and δ_k denotes the time since the last keyboard activity. For example, the email application requests an attestation specifying that a keyboard or mouse click should have occurred within the last Δ_k or Δ_m milliseconds respectively, where the $\Delta_{\{k,m\}}$ represents the application-specified upper-bound. The attester generates attestations that indicate this time lag, or refuses if that lag is longer than $\Delta_{\{k,m\}}$ milliseconds.

This method is simpler and cheaper in terms of required resources than an alternative we carefully considered and eventually discarded. Using keyboard activity traces, we found that good-quality guesses can be extracted by trying to *support* the content of an attestation request using specific recent keyboard and mouse activity. For example, the attester can observe and remember a short sequential history of keystrokes and mouse clicks in order of observation. When a particular attestation request comes in, the attester searches for the longest subsequence of keyclicks that matches the content to attest. An attestation could be issued containing the quality of match (e.g., a percentage of content matched), only raising an explicit alarm and potential user prompting if that match is lower than a configurable threshold (say 60%). This design point would not attest to bot requests unless they contained significant content overlap with legitimate user traffic. Nevertheless this method raised great implementation complexity, given the typical multitasking behavior of modern users (switching between windows, interleaving keyboard and mouse activity, inserting, deleting, selecting and overwriting text, etc.). So, we ultimately discarded it in favor of the simpler $t - \delta$ attester, which allowed a simple implementation with a small TCB size.

One drawback of the $t - \delta$ attester is that it allows a bot to generate attestations for its own traffic by "harvesting" existing user activity. So, NAB could allow illegitimate traffic to receive attestations, though only at the rate of human activity.

NAB mitigates this situation in two ways. First, NAB ensures that two attestations are separated by at least the application-specific Δ milliseconds. For email, we find from the traces (§6) that $\Delta = 1$ second works well. Since key clicks cannot be captured or stored, we throttle a bot significantly in practice. Today's bots send several tens of thousands of spam within a few hours [14], so even an adaptive bot is constrained by this rate limit.

Second, if legitimate traffic fails to receive an attestation (e.g., because bot code attestation requests absorbed all recent user activity before the user's application had a chance to do so), a NAB-aware application alerts the user that it has not been able to acquire an attestation, possibly alerting the user that unwholesomeness is afoot at her computer. We note that this technique is not perfect, because a bot can hijack such prompts. In practice, we found that such feedback is useful, although we evaluate NAB assuming adversarial bots.

4.2 What To Attest

The second attester design decision is what to attest, i.e., how much to link a particular attestation to the issuer, the verifier, and the content.

Traditional human activity solutions such as CAPTCHAs [30] do not link to the actual request being satisfied. A CAPTCHA is a challenge that only a human is supposed to be able to respond to. A correct response to a CAPTCHA attests to the fact that a human was likely involved in answering the question, but it does not say where the human was or whether the answer came from the user of the service making the request. The problem is that human activity can be trafficked, as evidenced by spammers who route human activity challenges meant for account creation to sketchy web sites to have them solved by those sites' visitors in exchange for free content [25], or to sweatshops with dedicated CAPTCHA solvers. Thus, *a* human was involved in providing the activity, but not necessarily the human intended by the issuer of the challenge.

In contrast, NAB generates responder-specific, content-specific, and, where appropriate, challenger-specific attestations. Attestations are certificates of human activity that contain a signature over the entire request content. For example, an email attestation contains the signature over the entire email, including the "From:" address (i.e., the responder), the email body (i.e., the content), and the "To:" address (i.e., the challenger). Similarly, a web request attestation contains the URL, which provides both responder-specific and content-specific attestations.

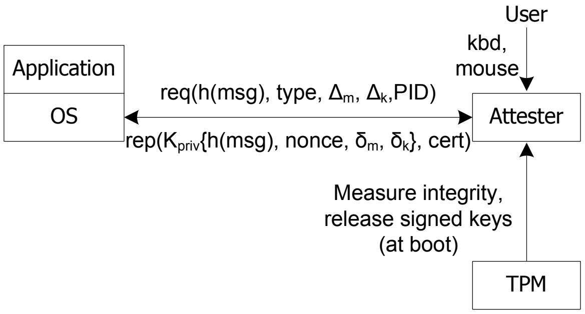
User
kbd, mouse
Application
OS
req(h(msg), type, Δm, Δk,PID)
Attester
rep(Kpriv{h(msg), nonce, δm, δk}, cert)
Measure integrity, release signed keys (at boot)
TPM

Figure 2: Attester interfaces.

Content-specific attestation is more subtle. Whereas CAPTCHAs are used today for coarse-grained actions such as email account creation, they are considered too intrusive to be used for finer granularity requests such as sending email or retrieving web URLs. So, in practice, the challenge response is "amortized" over multiple requests (i.e., all email sent from the CAPTCHA-created mail account). Even if an actual human created the account, nothing prevents the bots in that human's desktop from sending email indiscriminately using that account.

Finally, challenger-specific attestation helps in ensuring that unwitting, honest humans do not furnish attestations for bad purposes. A verifier expecting an attestation from human A's attester will reject an attestation from human B that might be provided instead. In the spam example, this is tantamount to explicit sender authentication.

Attestations with these three properties, together with application-specific verifier policies described in §5.2, meet our second and third requirements (§3.1).

4.3 Attester API

Figure 2 shows the relationship between the attester and other entities. The API is simple: there is only a single request/reply pair of calls between the OS and the attester. An application's attestation request contains the hash of the message to be attested (i.e., the contents of an email message or the URL of a browser click), the type of attestation requested, and the process id (PID) of the requesting process.

If the attester verifies that the type of attestation being requested is consistent with user activity seen on the keyboard/mouse channels, it signs the attestation and, depending on the attestation type, includes δ_m and δ_k, which indicate how long ago a mouse click and a keyboard click respectively were last seen. The attestation is an offline computation , and is thus an instance of a non-interactive proof of human activity.

The same API is used for all applications. The only customization allowed is whether to include the values of the δ_m or δ_k, depending on the attestation type. The attester uses a group signature scheme for anonymous attestations, extending the Direct Anonymous Attestation (DAA) service [7] provided by recent TPMs. Anonymous attestations preserve the current privacy semantics of web and email, thereby meeting our fourth and final requirement (§3.1).

We have currently defined and implemented two attestation types. Type 0 is for interactive applications such as all types of web requests. Type 1 is for delay-tolerant applications such as email. Type 0 attestations are generated only when there is either a mouse or keyboard click in the last one second, and do not include the δ_m or δ_k values. Type 0 attestations are offered as a privacy enhancement, to prevent verifiers from tracking at a fine temporal granularity a human user's activity or a particular request's specific source machine. We chose one second as the lag for Type 0 attestations since it is sufficient for local interactive applications; for example, this is ample time between a key or mouse click and a local action such as generating email or transmitting an HTTP GET request. Type 1 attestations can be used with all applications we have examined, when this finer privacy concern is unwarranted. To put the two types in perspective, a Type 0 attestation is roughly equivalent to a Type 1 attestation requested with $\Delta_m = \Delta_k = 1sec$ and in which the attested δ_m/δ_k values have been hidden.

Attestation structure. An attestation has the form $\langle d, n, \delta_m, \delta_k, \sigma, C\rangle$. It contains a cryptographic content digest d (e.g., a SHA-1 hash) of the application-specific payload being attested to; a nonce n used to maintain the freshness of the attestations and to disallow improper attestation reuse; the $\delta_{\{k,m\}}$ values (for type 1 attestations); the attestation signature $\sigma = sign(K_{priv}, \langle d, n, \delta_m, \delta_k\rangle)$; and a certificate C from the TPM guaranteeing the attester's integrity, the version of the attester being used, the attestation identity key of the TPM that measured the attester integrity, and the signed attester's public key K_{pub} (Figure 2). The certificate C is generated during booting of the attester and is stored and reused until reboot.

The mechanism for attesting to web requests is simple: when a user clicks on a URL that is either a normal link or an ad, the browser requests an attestation on the entire page URL. After the browser fetches the page content, it uses the same attestation to retrieve any included objects within the page. As explained in §5.2, the verifier accepts the attestation for all included objects.

The mechanism for sending email in the common case is also straightforward: the entire email message, including headers and attachments, constitutes the request. Interestingly, the same basic mechanism is extensible to other email usage scenarios, such as text or web-based email, email-over-ssh, batched and offline email, and script-generated email.

Email usage scenarios (mailing lists; remote, batched, offline, scripted or web mail). To send email to mailing lists, the attester attests to the email normally, except that the email destination address is the name of the target mailing list. Every recipient's verifier then checks that the recipient is subscribed to the mailing list, as described in §5.2. Also, a text-based email application running remotely over ssh can obtain attestations from the local machine with the help of the ssh client program executing locally. This procedure is similar to authentication credential forwarding implemented in ssh. Similarly, a graphical email client can obtain and store an attestation as soon as the "send" button is clicked, regardless of whether it has a working network connection, or if the email client is in an offline mode, or if the client uses an outbox to batch email instead of sending it immediately. In case of web mail, a browser can obtain an attestation on behalf of the web application.

Script-generated email is more complex. The PID argument in the attestation request (Figure 2) is used for deferred attestations, which are attestations approved ahead of time by the user. Such forms of attestation are not required normally, and are useful primarily for applications such as email-generating scripts, cron-jobs, etc. When an application requests a deferred attestation, the user approves the attestation explicitly through a reserved click sequence (currently "Ctl-Alt-F4", followed by number of deferred attestations). These attestations are stored in a simple PID-table in the attester, and released to the application in the future. Since the content of a deferred attestation is not typically known until later (such as when the body of an email is dynamically generated), it is dangerous to release an unbound attestation to the untrusted OS. Instead, the attester stores the deferred attestations in its own memory, and releases only bound attestations. Although the attester ensures that unbound attestations are not released to the untrusted OS, thereby limiting damage, there is no way to ensure that these attestations are not stolen by a bot faking the legitimate script's PID. However, the user is able to reliably learn about the missing attestations after this occurrence, which is helpful during troubleshooting.

4.4 Attester Implementation

The attester is a small module, currently at fewer than 500 source code lines. It requires a TPM chip conforming to any revision of the TPM v1.2 specification [28].

Attester installation. The attester is installed by binding its hash value to an internal TPM register called a Platform Configuration Register (PCR). We use PCR_{18}. Initially, the register value is -1. We extend it[1] with the attester through the TPM operation:

$$PCRExtend(18, H(ATT))$$

where $H(ATT)$ is the attester's hash. If the attester needs to be updated for some reason (which should be a rare event), PCR_{18} is reinitialized and extended with the new code value.

Key generation. At install time, the attester generates an anonymous signing key pair: $\{K_{pub}, K_{priv}\}$. This key pair is derived from the attestation identity key AIK of the TPM, and is an offline operation. K_{priv} allows the attester to sign requests anonymously. The attester then seals the private key K_{priv} to the TPM using the TPM's private storage root key K_{root}.

Assume that the system BIOS, which boots before the attester, extends PCR_{17}. Thus, the sealing operation renders K_{priv} inaccessible to everyone but the attester by executing the TPM call:

$$Seal((17, 18), K_{priv})$$

which returns the encrypted value C of K_{priv}. The TPM unseals and releases the key only to the attester, after the attester is booted correctly.

Until the TPM releases K_{priv} and the accompanying certificate to the attester, there is thus no way for the host to prove to an external verifier that a request is accompanied by human activity. Conversely, if the attester has a valid private key, the external verifier is assured that the attester is not tampered with.

Attester booting. The attester uses a static chain of trust rooted at the TPM and established at boot-time. It is booted as part of the secure boot loading operation before the untrusted OS itself is booted. After the BIOS is booted, it measures and launches the attester. After the attester is launched, it unseals the previously sealed K_{priv} by executing:

$$Unseal(C, MAC_{K_{root}}((17, PCR_{17}), (18, PCR_{18})))$$

The *Unseal* operation releases K_{priv} only if the PCR registers 17 and 18 after reboot contain the same hash values as the registers at the time of sealing K_{priv}. If the PCR values match, the TPM decrypts C and returns K_{priv} to the attester.

Thus, by sealing the anonymous signing key K_{priv} to the TPM and using secure boot loading to release the key to the attester, NAB meets the challenge of generating attestations without globally unique identities.

Attester execution. The attester waits passively for attestation requests from an application routed through the untrusted OS. A small untrusted stub is loaded into the OS in order to interact with the attester on behalf of the application.

With our current attester design and implementation, applications need to be modified in order to obtain attestations. We find the modifications to be fairly small and localized (§6). The only change as far as applications are concerned is to first obtain appropriate attestations and then include them as part of the requests they submit today. Protocols such as SMTP (mail) or HTTP (web) need not be modified in order to include this functionality. SMTP allows extensible message headers, while HTTP can include the attestation as part of the "user agent" browser string or as an extended header.

5 Verifier Design and Implementation

We now describe how verifiers use attestations to implement attack-specific countermeasures for spam, DDoS and click-fraud.

5.1 Verifier Design

The verifier is co-located with the server processing requests. We describe how the server invokes the verifier for each application in §5.2. When invoked, the verifier is passed both the attestation and the request. The attestation and request contain all the necessary information to validate the request.

The verifier first checks the validity of the attester public key used for signing the request, by traversing the public-key chain in the certificate C (Figure 2). If valid, it then recomputes the hash of the request's content and verifies whether the signed hash value in the attestation matches the request's contents. Further, for attestations that include the $\delta_{\{k,m\}}$ values, the verifier also checks whether $\delta_{\{k,m\}}$ are less than the application-specified $\Delta_{\{k,m\}}$. The verifier then checks to ensure that the attestation is not being double-spent, as described in § 5.3.

A bot running in an untrusted domain cannot masquerade as a trusted attester to the verifier because a TPM will not release the signed K_{pub} (Figure 2) to the bot without the correct code hash. Further, it derives no benefit from tampering with the δ values it specifies in its requests, because the verifier enforces the application-specified upper-limit on $\delta_{\{k,m\}}$.

The verifier then implements an application-specific policy as described next.

5.2 Application-specific Policies

Verifiers implement application-specific policies to deal with bot traffic. Spam can be more aggressively filtered using information in the attestations, legitimate email with attestations can be correctly classified, DDoS can be handled more effectively by prioritizing requests with attestations over traffic without attestations, and click-fraud can be reduced by only serving requests with valid attestations and ignoring other requests.

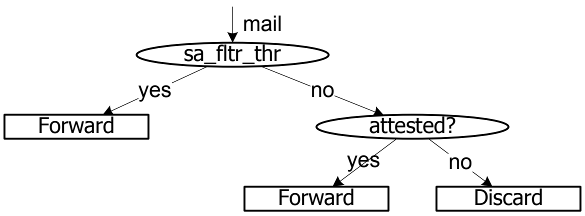

Figure 3: Sender ISP's verifier algorithm.

We now describe how the verifier implements such application-specific policies. Note that these are only example policies that we constructed for our three case studies, and many others are possible.

5.2.1 Spam policy

The biggest problem with Bayesian spam filters such as spamassassin today is that they either flag too much legitimate email as spam, or flag too little spam as such.

When all legitimate requests are expected to carry attestations, the verifier can set spam filters aggressively to flag questionable unattested messages as spam, but use positive evidence of human activity to "whitelist" questionable attested messages.

Sender ISP's email server. The verifier sits on the sender ISP's server alongside a Bayesian spam filter like spamassassin. The filter is configured at an aggressive, low threshold (e.g., -2 instead of the default 5 for spamassassin), because the ISP can force its users to send email with attestations, in exchange for relaying email through its own servers.

This low spamassassin "required score" threshold (or `sa_fltr_thr` in Figure 3) tags most unattested spam as unwanted. However, in the process, it might also tag some valid email as spam. In order to correct this mistake, the verifier "salvages" messages with a high spam filter score that carry a valid attestation, and relays them; high-score, unattested email is discarded as spam. This step ensures that legitimate human-generated email is forwarded unconditionally, even if the sender's machine is compromised. Thus, NAB guarantees that human-generated email from even a compromised machine is forwarded correctly (for example, in our trace study in §6, we did not find a single legitimate email that was ultimately rejected). Finally, while spam that steals attestations will also be forwarded, in our trace-based study this spam volume is 92% less than the spam forwarded today (§6). This reduction is because the attester limits the bot to acquiring attestations only when there is human activity, and even then at a rate limit of at most one attestation per Δ (one second for type 0 attestations).

Recipient's inbox. A second form of deploying the verifier is at the email recipient. This form can coexist with the verifier on the sender's side.

We observe that any email server classifying email as spam or not can ensure that a legitimate email is not misclassified by improving the spam score for email messages with attestations by a small number (=3, §6). This number should be high enough that all legitimate email is classified correctly, while spam with or without attestations is still caught.

The verifier improves the score for all attested emails by 3, thereby vastly improving the delivery of legitimate email. Additionally, in this deployment, the verifier also checks that the 'To:" or "Cc:" headers contain the recipient's email address or the address of a subscribed mailing list. If not (for example, in the case of "Bcc:"), it does not improve the spam score by 3 points.

Incentives. Email senders have an incentive to deploy NAB because it prevents their email from being misclassified as spam. Verifiers can be deployed either for reducing spam forwarded through mail relays or for ensuring that all legitimate email is classified and delivered correctly. Home ISPs, which see significant amount of compromised hosts on their networks, can benefit from the first deployment scenario, because, unlike other methods of content or IP-based filtering, attestations still allow all legitimate email from compromised hosts, while reducing spam significantly (§6). Also, web-based mail servers such as gmail have an incentive to deploy NAB so that they can avoid being blacklisted by other email relays by reducing the spam they forward today. Finally, email recipients have an incentive to deploy NAB because they will receive all legitimate email correctly, unlike today (§6).

5.2.2 DDoS policy

We consider scenarios where DDoS is effected by overloading servers, and not by flooding networks. The verifier resides in a firewall or load balancer, and observes the response time of the web server to determine whether the server is overloaded [31]. Here, unlike in spam, the verifier does not drop requests with invalid or missing attestations. Instead, it prioritizes requests with valid attestations over those that lack them. Prioritizing, rather than dropping, makes sense because some valid requests may actually be generated automatically by machines (for example, automatic page refreshes on news sites like `cnn.com`).

The verifier processes the web request in the following application-specific manner. If the request is for a page URL, the verifier treats it as a fresh request. It keeps a set of all valid attestations it has seen in the past 10 minutes, and adds the attestation and the requested page URL to the list. If the request is for an embedded object within a page URL, the verifier searches the attestation list to see if the attestation is present in the list. If the attestation is present in the list, and if the requested object belongs to the page URL recorded in the list for the attestation, the verifier treats the attestation as valid. Otherwise, it lowers the priority of the request. The verifier ages the stored attestation list every minute.

The priority policy serves all outstanding attested requests first, and uses any remaining capacity to serve all unattested requests in order.

Incentives. Overloaded web sites have a natural incentive to deploy verifiers. While users have an incentive to deploy attesters to receive priority treatment, the attester deployment barrier can be still high. However, since our attester is not application-specific, it is possible for the web browser to leverage the attester deployed for email or click-fraud.

5.2.3 Click-fraud Policy

Click-fraud occurs whenever an automated request is generated for a click, without any interest in the click target. For example, a botmaster puts up a web site to show ads from companies such as Google, and causes his bots to fetch ads served by Google through his web site. This action causes Google to pay money to the botmaster. Similarly, an ad target's competitor might generate invalid clicks in order to run up ad costs and bankrupt the ad purchaser. Further, the competitor might be able to purchase ad words for a smaller price, because the victim might no longer bid for the same ad word. Finally, companies like Google have a natural incentive to prove to their advertisers that ads displayed together with search results are clicked not by bots but by humans.

With NAB, a verifier such as Google can implement the verifier within its web servers, configured as a simple policy of not serving unattested requests. Also, it can log all attested requests to prove to the advertiser that the clicks Google is charging for are, in fact, human-generated.

Incentives. Companies like Google, Yahoo and Microsoft that profit from ad revenue have a good incentive to deploy verifiers internally. They also have an incentive to distribute the attester as part of browser toolbars. Such toolbars are either factory installed with new PCs, or the user can explicitly grant permission to install the attester. While the user may not benefit directly in this case, she benefits from spam and DDoS reduction, and from being made aware of potential problems when a bot steals key clicks.

5.3 Security guarantees

NAB provides two important security guarantees. First, it ensures that attestations cannot be double-spent. Second, it ensures that a bot cannot steal key clicks and accumulate attestations beyond a fixed time window, which reduces the aggregate volume and burstiness of bot traffic.

The verifier uses the nonce in the attestation (Figure 2) for these two guarantees. The verifier stores the nonces for a short period (10 minutes for web requests, one month for email). We find this nonce overhead to be small in practice (§6.3). If a bot recycles an attestation after one month, and the spam filter at the verifier flags the email as spam based on content analysis, the verifier uses the "Date:" field in the attested email to safely discard the request because the message is old.

The combination of application-specific verifier policy and content-bound attestations can also be used to mitigate bursty attacks. For example, a web URL can include an identifier that encodes the link freshness. Since attestations include the identifier, the verifier can discard out-of-date requests, even if they have valid signatures.

6 Evaluation

In this section, we evaluate NAB's two main components: a) our current attester prototype with respect to metrics such as TCB size, CPU requirements, and application changes; and b) our verifier prototype with respect to metrics such as the extent to which it mitigates attack-specific traffic such as spam, DDoS and click-fraud, and the rate at which it can verify attestations.

Our main experiments and their conclusions are shown in Table 1. We elaborate on each of them in turn.

6.1 Attester Evaluation

TCB size. We implemented the attester as a kernel module within Xen. Xen is well-suited because it provides a virtual machine environment with sufficient isolation between the attester and the untrusted OS. However, the chief difficulty was keeping the total TCB size small. Striving for a small TCB allows the attester to handle untrusted OSes with a higher assurance. While the Xen VM itself is small (about 30 times smaller than the Linux kernel), we have to factor the size of a privileged domain such as Domain-0 into the TCB code base. Unfortunately, this increases the size of the TCB to more than 5 million source lines of code (SLOC), the majority of which is device driver code.

Instead, we started with a minimal kernel that only includes the necessary drivers for our platform. We included the Xen VMM and built untrusted guest OSes us-

Experiment	Conclusion
TCB size	500 source lines of code (SLOC) for attester, 30K SLOC total
Attester CPU cost	$< 10^7$ instructions/attestation
Application changes	<250 SLOC for simple applications
Worst-case spam mitigation	$> 92\%$ spam suppressed; no human-sent email missed
Worst-case DDoS mitigation	$> 89\%$ non-human requests identified; no human requests demoted
Worst-case click-fraud mitigation	$> 87\%$ automated clicks denied; no human request denied
Verifier throughput	$> 1,000$ req/s. Scalable to withstand 100,000-bot DDoS

Table 1: Summary of key experiments and their results.

ing the mini-OS [19] domain building facility included in the Xen distribution. Mini-OS allows the user-space applications and libraries of the host VM to be untrusted, leaving us with a total codebase of around 30,000 source lines of code (SLOC) for the trusted kernel, VMM and attester. Our attester was less than 500 SLOC. While this approach produced a TCB that can be considered reasonably small, especially compared to the status quo, we are examining alternatives such as using Xen's driver domain facility that allows device drivers to run in unprivileged domains. We are also working on using the IOMMUs found on the newer Intel platforms, which enable drivers for devices other than keyboard and mouse to run in the untrusted OS, while ensuring that the attester cannot be corrupted due to malicious DMA requests. Such an approach makes the attester portable to any x86 platform.

Attester CPU cost. The attester uses RSA signatures with a 1024-bit modulus, enabling it to generate and return an attestation to the application with a worst-case latency of 10 ms on a 2 GHz Core 2 processor. This latency is usually negligible for email, ad click, or fetching web pages from a server under DDoS. Establishing an outgoing TCP connection to a remote server usually takes more than this time, and attestation generation is interleaved with connection establishment.

Application changes. We modified two command-line email and web programs to request and submit attestations: NET::SMTP, a Perl-based SMTP client, and cURL, an HTTP client written in C. Both modifications required changes or additions of less than 250 SLOC.

6.2 Verifier Evaluation

We used a trace study of detailed keyboard and mouse activity of 328 volunteering users at Intel to confirm the mitigation efficacy of our application-specific verifier policies. We find the following four main benefits with our approach:

1. If the sender's mail relay or the receiver's inbox uses NAB and checks for attestations, the amount of spam that passes through tuned spam filters (i.e., false negatives) reduces by more than 92%, while not flagging any legitimate email as spam (i.e., no false positives). The spam reduction occurs by setting the "scoring thresholds" aggressively; the presence of concomitant human activity greatly reduces the number of legitimate emails flagged as spam.
2. In addition to reduced spam users see in their inboxes, NAB also reduces the peak processing load seen at mail servers, because the amount of attested spam that can be sent even by an adaptive botnet is bounded by the number of human clicks that generate attestations. Hence, mail servers can prioritize attested requests potentially dropping low-priority ones, which improves the fraction of human-generated email processed during high-load periods.
3. NAB can filter out more than 89% of bot-mounted DDoS activity without misclassifying human-generated requests.
4. NAB can identify click-fraud activity generated by adware with more than 87% accuracy, without losing any human-generated web clicks.

Methodology. We use the keyboard and mouse click traces collected by Giroire et al. [11]; activity was recorded on participants' laptops at one-second granularity at all times, both at work and at home. Each user's trace is a sequence of records with the following relevant information: timestamp; number of keyboard clicks within the last second; number of mouse clicks within the last second; the foreground application that is receiving these clicks (such as "Firefox", "Outlook", etc.); and the user's network activity (i.e., the TCP flow records that were initiated in the last one second). Nearly 400 users participated in the trace study, but we use data from 328 users because some users left the study early. These 328 users provide traces continuously over a one-month period between Jan–Feb 2007, as long as their machines were powered on. While the user population size is moderate, the users and the workloads were diverse. For example, there were instances of significant input device activity corresponding to gaming activity outside regular work. So, we believe the traces are sufficiently representative of real-world activity.

Separately, we also collected malware traces from a honeypot. The malware whose traces we gathered included: a) the Storm Worm [13], which was until recently the largest botnet, generating several tens of billions of spam messages per day; and b) three adware bots called 180solutions, Nbcsearch and eXactSearch, which are known to perpetrate click-fraud against Yahoo/Overture. For spam, we also used a large spam corpus containing more than 100,000 spam messages and 50,000 valid messages [1]. Each message in the corpus is hand-classified as spam or non-spam, providing us with ground-truth. For DDoS, we use traffic traces from the Internet Traffic Archive [27], which contain flash-crowd scenarios. We assume that these flash crowds represent DDoS requests, because, as far as an overloaded server is concerned, the two scenarios are indistinguishable.

We overlay the user activity traces with the malware and DDoS traces for each user, and compare the results experienced by the user at the output of the verifier with and without attestations. We consider two strategies for overlaying requests: a normal bot and an adaptive bot. The adaptive bot represents the worst-case scenario for the verifier, because it monitors human activity and modulates its transmissions to collect attestations and masquerade as a user at the verifier.

We consider an adaptive adversary that buffers its requests until it sees valid human activity, and simulate the amount of benefit NAB can provide under such adversarial workloads.

Spam mitigation. The verifier can be used in two ways (§5.2). First, mail relays such as gmail or the SMTP server at the user's ISP can require attestations for outgoing email. In this case, the main benefit comes from filtering out all unattested spam and catching most attested spam, while allowing all legitimate email. So, the main metric here is how much attested spam is suppressed. Second, the inbox at the receiver can boost the "spam score" for all attested email, thereby improving the probability that a legitimate email is not misclassified. So, the main metric here is how much attested human-generated email is misclassified as spam.

Figure 4 shows the amount of spam, attested or not, that managed to sneak through spamassassin's Bayesian filter for a given spam threshold setting. By setting a spam threshold of -2 for an incoming message , and admitting messages that still cleared this threshold and carried valid attestations, we cut down the amount of spam forwarded by mail relays by more than 92% compared to the amount of spam forwarded currently.

From our traces, we also found that no attested human-generated email is misclassified as spam for a spam threshold setting of 5, as long as the spam score of attested messages is boosted by 3 points. On the other hand, spamassassin uses a threshold of 5 by default because, without attestations, a lot of valid email would be missed if it were to use a spam score of -2. Even so, about 0.08% of human-generated email is still misclassified as spam, which is a significant improvement of legitimate email reception.

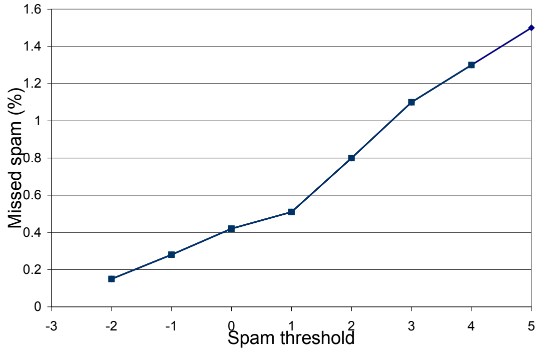

Figure 4: Missed spam percentage vs. spam threshold with attestations. By setting spam threshold to -2, spam cleared by spamassassin and received in inboxes today is reduced more than 92% even in worst case (i.e., adaptive bots), without missing any legitimate email.

There is another benefit that the verifier can derive by using attestations. It comes in the form of reduced peak load observed while processing spam. Today's email servers are taxed by ever-increasing spam requests [23]. At peak times, the mail server can prioritize messages carrying attestations over those that do not, and process the lower-priority messages later.

Figure 5 shows the CDF of the percentage of spam requests that the verifier must still service at a high priority because of stolen attestations. NAB demotes spam traffic without attestations by more than 91% in the worst case (equivalently, less than 7.5% of spam traffic is served at the high priority). At the same time, no human-generated requests are demoted. The mean of the admitted spam traffic is 2.7%, and the standard deviation is 1.3%. Thus, NAB reduces peak server load by more than 10×.

DDoS mitigation. The verifier uses the DDoS policy described in §5.2, by giving lower priority to requests without attestations. Figure 6 shows the CDF of the percentage of DDoS requests that the verifier still serves at a high priority because of stolen attestations. NAB demotes DDoS traffic by more than 89% in the worst case (equivalently, only 11% of DDoS traffic is served at the high priority). At the same time, no human-generated requests are demoted. The mean of the admitted DDoS traffic is 5.8%, and the standard deviation is 2.2%.

Click-fraud mitigation The verifier uses the Click-fraud policy described in §5.2. Figure 7 shows the amount of click-fraud requests that the verifier satisfies due to

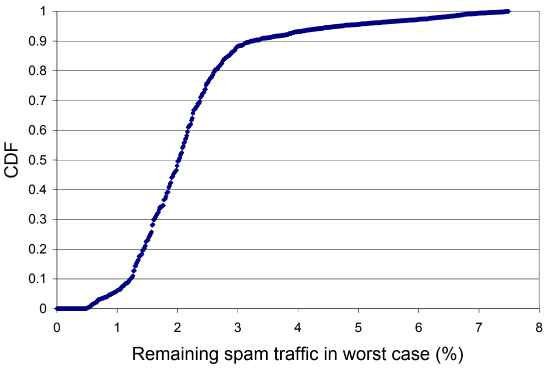

Figure 5: CDF of percentage of bots' spam requests serviced by an email server in the worst case. The mail server's peak spam processing load is reduced to less than 7.5% of its current levels.

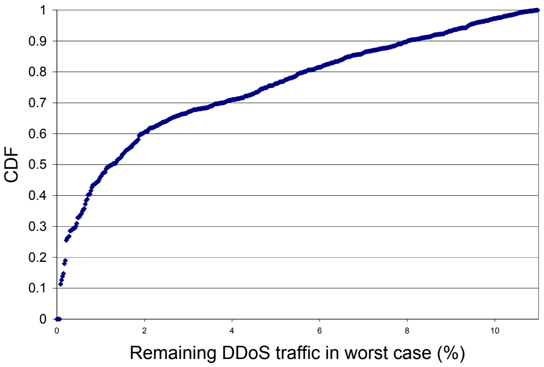

Figure 6: CDF of percentage of bots' DDoS requests serviced in the worst case. Allowed DDoS traffic is restricted to less than 11% of original levels.

valid attestations. NAB denies more than 87% of all in the worst case (equivalently, only 13% of all click-fraud requests is serviced). At the same time, no human-generated requests are denied service. The mean of the serviced click-fraud traffic is 7.1%, and the standard deviation is 3.1%.

6.3 Verifier Throughput

The verifier processes attestations, which are signed RSA messages, at a rate of more than 10,000 attestations per second on a 2 GHz Core 2 processor. It benefits from the fact that RSA verification is several times faster than signing. The verifier processes an attestation by consulting the data base of previously seen nonces within an application-specific period. The longest is email, with a duration of one month, while nonces of web requests are stored for 10 minutes, and fit in main memory. Even in the worst-case scenario of a verifier at an ISP's busy SMTP relay, the storage and lookup costs for the nonces are modest—for a server serving a million clients, each of which sends a thousand emails per day, the nonce storage overhead is around 600 GB, which can fit on a single disk and incur one lookup overhead. This overhead is modest compared to the processing and storage costs incurred for reliable email delivery.

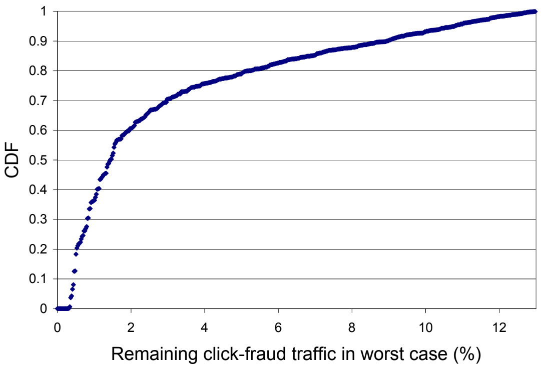

Figure 7: CDF of percentage of bots' click-fraud requests serviced in the worst case. Serviced click-fraud requests are restricted to less than 13% of original levels.

Another concern is that the verifier is itself susceptible to a DDoS attack. To understand how well our verifier can withstand DDoS attacks, we ran experiments on a cluster of 10 Emulab machines configured as distributed email verifiers. We launched a DDoS from bots with fake attestations. Each DDoS bot sent 1 req/s to one of the ten verifiers at random, in order to mimic the behavior of distributed low-rate bots forming a DDoS botnet. Our goal was to determine whether a botnet of 100,000 nodes (which is comparable to the median botnet size) can overwhelm this verifier infrastructure or not. Our bot implementation used 100 clients to simulate 1000 bots each, and attack the ten verifier machines. We assume network bandwidth is not a bottleneck, and that the bots are targeting the potential verification overhead bottleneck. A verifier queues incoming requests until it can attend to it, and has sufficient request buffers.

Figure 8 shows the latency increase (in ms) experienced by a normal client request. Normally, a user takes about 1 ms to get her attestation verified. With DDoS, we find that even a 100,000-node botnet degrades the performance of a normal request only by an additional 1.2 ms at most. Hence, normal request processing is not affected significantly. Thus, a cluster of 10 verifiers can withstand a 100,000-node botnet using fake attestations.

7 Related Work

We classify prior work into three main categories.

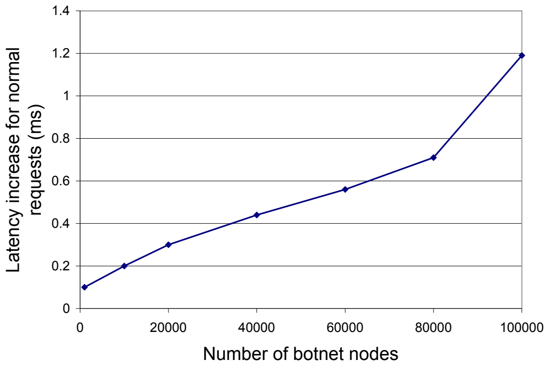

Figure 8: Request processing latency at the verifier.

Human activity detection. CAPTCHAs [30**?**] are the currently popular mechanism for proving human presence to remote verifiers. However, as described in §4, they suffer from four major drawbacks that render them less attractive for mitigating botnet attacks. First, CAPTCHAs as they are used today are transferable and not bound to the content they attest, and are hence vulnerable to man-in-the-middle attacks, although one could imagine designs to improve this shortcoming; second, they are semantically independent of the application (i.e., unbound to the user's intent), are hence exposed to human solver attacks; third, they are obtrusive, which restricts their use for fine-grained attestations (by definition, CAPTCHAs require manual human input), and hence cannot be automated, unlike NAB. Also, we are witnessing continued successes in breaking the CAPTCHA implementations of several sites such as Google, Yahoo, and MSN [12], leading some to question even their long-term viability [34], at least in their current form. By contrast, NAB's security relies on cryptographic protocols such as RSA that have been studied and used longer.

The recent work on the Nexus operating system [33] has developed support for application properties to be securely expressed using a trusted reference monitor mechanism. The Nexus reference monitor is more expressive than a TPM implementing a hash-based trusted boot. So, it allows policies restricting outgoing email only from registered email applications. In contrast, we assume commodity untrusted OS and applications.

The approach of using hardware to enable human activity detection has been described before in the context of on-line games, using untrusted hardware manageability engines (such as Intel's AMT features) [21].

Mitigating spam, DDoS and click-fraud. There is extensive literature related to mitigation techniques for Spam [2], DDoS [20, 35] and click-fraud [26]. There are still no satisfactory solutions, so application-specific defenses are continuously proposed. For example, Occam [10], SPF (Sender Policy Framework), DKIM (DomainKeys Identified Mail) and "bonded sender" [6] have been put forth recently as enhancements. Similarly, DDoS and click-fraud mitigation have each seen several radically different attack-specific proposals recently. These proposals include using bandwidth-as-payment [31], path validation [35], and computational proofs of work [20] for DDoS; and using syndicators, premium clicks, and clickable CAPTCHAs for click-fraud [26].

While all these proposals certainly have several merits, we propose that it is possible to mitigate a variety of botnet attacks using a uniform mechanism such as NAB's attestation-based human activity verification. Such a uniform attack mitigation mechanism amortizes its cost of deployment. Moreover, unlike some proposals, NAB does not rely on IP-address blacklisting, which is unlikely to work well because even legitimate requests from a blacklisted host are denied. Also, NAB can be implemented purely at the end hosts, and does not require Internet infrastructure modification.

Secure execution environments. The TPM specifications [28] defined by the Trusted Computing Group are aimed at providing primitives that can be used to provide security guarantees to commodity OSes. TPM-like services have been extended to OSes that cannot have exclusive access to a physical TPM device of their own, as with legacy and virtual machines. For example, Pioneer [22] provides an externally verifiable code execution environment for legacy devices similar to that provided by a hardware TPM, and vTPM [5] provides full TPM services to multiple virtualized OSes. NAB assumes a single OS and a hardware TPM, but can leverage this research in future.

XOM [15] and Flicker [16] provide trusted execution support even when physical devices such as DMA or, with XOM, even main memory are corrupted, while SpyProxy [18] blocks suspicious web content by executing the content in a virtual machine first. In contrast, NAB assumes compromised machines' hardware is functioning correctly, that the bot may generate diverse traffic such as spam and DDoS, and that owners do not mount hardware attacks against their own machines, which is realistic for botted machines.

8 Conclusions

This paper presented NAB, a system for mitigating network attacks by using automatically obtained evidence of human activity. NAB uses a simple mechanism centered around TPM-backed attestations of keyboard and mouse clicks. Such attestations are responder- and content-specific, and certify human activity even in the absence of globally unique identities. Application-specific verifiers use these attestations to implement various policies. Our implementation shows that it is feasible to provide such attestations at low TCB size and runtime cost. By evaluating NAB using trace analysis, we estimate that NAB can reduce the amount of spam evading tuned spam filters by more than 92% even with worst-case adversarial bots, while ensuring that no legitimate email is misclassified as spam. We realize similar benefits for DDoS and click-fraud. Our results suggest that the application-independent abstraction provided by NAB enables a range of verifier policies for applications that would like to separate human-generated requests from bot traffic.

Acknowledgments. We thank Nina Taft and Jaideep Chandrashekar for the click traces used in this paper, our shepherd Geoff Voelker, Bryan Parno, Frans Kaashoek and the anonymous reviewers for their helpful comments.

References

[1] 2005 TREC public spam corpus, http://plg.uwaterloo.ca/~gvcormac/treccorpus/.

[2] A plan for spam, http://www.paulgraham.com/spam.html.

[3] P. Barham, B. Dragovic et al. Xen and the art of virtualization. In *SOSP'03*.

[4] M. Bellare and P. Rogaway. Entity authentication and key distribution. In *CRYPTO'93*.

[5] S. Berger, R. Cáceres et al. vTPM: Virtualizing the Trusted Platform Module. In *USENIX-SS'06: Proceedings of the 15th conference on USENIX Security Symposium*.

[6] Bonded sender program, http://www.bondedsender.com.

[7] E. Brickell, J. Camenisch, and L. Chen. Direct anonymous attestation. In *CCS'04*.

[8] Click fraud rate rises to 14.1%, http://redmondmag.com/columns/print.asp?EditorialsID=1456.

[9] Five percent of Web traffic caused by DDoS attacks, http://www.builderau.com.au/news/soa/Five-percent-of-Web-traffic-caused-by-DDoS-attacks/0,339028227,339287902,00.htm.

[10] C. Fleizach, G. Voelker, and S. Savage. Slicing spam with occam's razor. In *CEAS'07*.

[11] F. Giroire, J. Chandrashekar et al. The Cubicle Vs. The Coffee Shop: Behavioral Modes in Enterprise End-Users. In *PAM'08*.

[12] Gmail CAPTCHA cracked, http://securitylabs.websense.com/content/Blogs/2919.aspx.

[13] T. Holz, M. Steiner, F. Dahl, E. Biersack, and F. Freiling. Measurements and mitigation of peer-to-peer-based botnets: A case study on Storm worm. In *Leet'08*.

[14] C. Kanich, C. Kreibich et al. Spamalytics: An empirical analysis of spam marketing conversion. In *CCS'08*.

[15] D. Lie, C. A. Thekkath, and M. Horowitz. Implementing an untrusted operating system on trusted hardware. In *SOSP'03*.

[16] J. M. McCune, B. J. Parno, A. Perrig, M. K. Reiter, and H. Isozaki. Flicker: An execution infrastructure for TCB minimization. In *EuroSys'08*.

[17] D. Moore, C. Shannon, D. J. Brown, G. M. Voelker, and S. Savage. Inferring Internet denial-of-service activity. *ACM Trans. Comput. Syst.*, 24(2), 2006.

[18] A. Moshchuk, T. Bragin, D. Deville, S. D. Gribble, and H. M. Levy. SpyProxy: Execution-based detection of malicious web content. In *USENIX'07*.

[19] D. G. Murray, G. Milos, and S. Hand. Improving xen security through disaggregation. In *VEE'08*.

[20] B. Parno, D. Wendlandt et al. Portcullis: Protecting connection setup from denial-of-capability attacks. In *SIGCOMM'07*.

[21] T. Schluessler, S. Goglin, and E. Johnson. Is a bot at the controls?: Detecting input data attacks. In *SIGCOMM workshop on Network and system support for games*, 2007.

[22] A. Seshadri, M. Luk et al. Pioneer: verifying code integrity and enforcing untampered code execution on legacy systems. In *SOSP'05*.

[23] Six botnets churning out 85% of all spam, http://arstechnica.com/news.ars/post/20080305-six-botnets-churning-out-85-percent-of-all-spam.html.

[24] Spam reaches all-time high of 95% of all email, http://www.net-security.org/secworld.php?id=5545.

[25] Spammers using porn to break CAPTCHAs, http://www.schneier.com/blog/archives/2007/11/spammers_using.html.

[26] The first AdFraud workshop, http://crypto.stanford.edu/adfraud/.

[27] Traces in the Internet Traffic Archive, http://ita.ee.lbl.gov/html/traces.html.

[28] Trusted Platform Module (TPM) specifications, https://www.trustedcomputinggroup.org/specs/TPM/.

[29] Vista's UAC security prompt was designed to annoy you, http://arstechnica.com/news.ars/post/20080411-vistas-uac-security-prompt-was-designed-to-annoy-you.html.

[30] L. von Ahn, M. Blum, N. Hopper, and J. Langford. CAPTCHA: Using Hard AI Problems for Security. In *Eurocrypt'03*.

[31] M. Walfish, M. Vutukuru, H. Balakrishnan, D. Karger, and S. Shenker. DDoS Defense by Offense. In *SIGCOMM'06*.

[32] A. Whitten and J. D. Tygar. Why Johnny can't encrypt: A usability evaluation of PGP 5.0. In *USENIX Security'99*.

[33] D. Williams, P. Reynolds, K. Walsh, E. G. Sirer, and F. B. Schneider. Device driver safety through a reference validation mechanism. In *OSDI'08*.

[34] Windows Live Hotmail CAPTCHA cracked, exploited, http://arstechnica.com/news.ars/post/20080415-gone-in-60-seconds-spambot-cracks-livehotmail-captcha.html.

[35] X. Yang, D. Wetherall, and T. Anderson. A DoS-limiting network architecture. In *SIGCOMM'05*.

Notes

[1] The TPM terminology uses the term *register extension* to imply appending a new value to the hash chain maintained by that register.