Enhancing Host Security Using External Environment Sensors

Ee-Chien Chang[1,⋆], Liming Lu[1], Yongzheng Wu[1,2,⋆⋆], Roland H.C. Yap[1], and Jie Yu[3,⋆⋆⋆]

[1] School of Computing, National University of Singapore, Singapore
{changec,luliming,wuyongzh,ryap}@comp.nus.edu.sg
[2] Temasek Laboratories, National University of Singapore, Singapore
[3] Department of Computer Science
National University of Defense Technology, China
yj@nudt.edu.cn

Abstract. We propose a framework that uses environment information to enhance computer security. We apply our framework to: enhance IDS performance; and to enrich the expressiveness of access/rate controls. The environment information is gathered by external (w.r.t the host) sensors, and transmitted via an out-of-band channel, and thus it is hard for adversaries not having physical access to compromise the system. The information gathered still remains intact even if malware use rootkit techniques to hide its activities. Due to requirements on user privacy, the information gathered could be coarse and simple. We show that such simple information is already useful in several experimental evaluations. For instance, binary user presence indicating at a workstation can help to detect DDoS zombie attacks and illegal email spam. Our framework takes advantage of the growing popularity of multimodal sensors and physical security information management systems. Trends in sensor costs suggest that it will be cost-effective in the near future.

Keywords: intrusion detection, spam, sensors, access control, host security.

1 Introduction

Securing computers against malware is increasingly difficult today. Anecdotes abound that the survival time of an unpatched PC running Windows XP connected to the Internet is in the order of minutes [1,2]. The recent Conficker worm [3] is estimated to have infected 6% of computers on the Internet.

Often the goal of the attackers is to infect a host to make it part of a botnet. The malware may be mostly dormant until it is activated, as such, it can be difficult to detect that the host is infected. The detection problem is made worse

⋆ Chang is supported by Grant R-252-000-413-232 from TDSI.
⋆⋆ Wu and Yap are supported by Grant R-394-000-037-422 from DRTech.
⋆⋆⋆ This work was done during internship at the National University of Singapore.

S. Jajodia and J. Zhou (Eds.): SecureComm 2010, LNICST 50, pp. 362–379, 2010.
© Institute for Computer Sciences, Social Informatics and Telecommunications Engineering 2010

since malware can exploit rootkit techniques to hide its presence and also any activity. For example, the Mebroot rootkit infects the master boot record of the hard disk allowing it to infect the Windows kernel during boot. After that it hides the changes to the master boot record to make it difficult for antivirus software to detect its presence.

Most security mechanisms tend to be host-based, and are often part of the operating system or interact with it. A primary exception is network-based security mechanisms which analyze network data and traffic. While there are some successes in detecting the presence or activity of malware by network-based security mechanisms, there are many other sources of information outside the host that are also useful for improving detection.

In this paper, we propose fusing environment information in decision making so as to enhance security. Our framework is designed to protect stationary machines (e.g. workstation) which users work on rather than servers controlled remotely or mobile laptops. We take advantage of the growth in pervasive computing and sensor technology providing relatively cheap sensors which can take a variety of physical measurements. We use the sensors in a variety of ways. As a measurement of how a resource is being used on the host, e.g. correlating CPU usage (the resource) with CPU temperature (the sensor measurement). The sensors allow us to determine the presence or absence of the user on the host and physical user activity such as keyboard usage. We remark that "user" means the human using the host and although there may be more than one user, we simply say user. Although it is possible to obtain comprehensive user activity information, e.g. user identity and key strokes from surveillance cameras, due to concerns of user privacy, we generally only consider sensors that only provide binary information like the presence of a user or keyboard activity or other coarse-grained indirect data. Another source of environment information could be from physical security information management systems. Such systems are already in-place in many organizations and could provide relevant environment information whereby user activities can be derived, e.g. the door entry control system gives evidence that a particular user is in the machine room.

One usage of environment information is in enhancing IDS performance by providing an external source that is difficult to be accessed or compromised by malwares or intruders. The results of an environment sensing based malware detector can be correlated with alerts from an IDS to reduce false positives. While existing IDS may incorporate network traffic information in gateways which are external to the host, the difference is that we make further use of other tamper-proof sensors and fuse it with user presence and activity. The following example shows the difference from the network intrusion problem. Consider the case of a single email being sent. At the network level, there is insufficient information to be able to identify whether it is a spam email generated by malware. Whereas, in our framework, suppose that the email is sent in the absence of the user and user activity, we can conclude that the email has been sent automatically. Furthermore, in the absence of any additional information, it is reasonable for a default rule to classify this email as spam activity. In this paper, we give some

application scenarios of malware detection – detecting when a botnet is using the host for email spam, distributed denial of service (DDoS) attack, and as a compute engine for offline dictionary attacks.

Environment information is also useful in other security applications. It increases the expressivity of access control and rate control policies by requiring privileged actions on a host to be correlated with physical user presence and physical activity. This prevents remote attacks which escalate privileges, e.g. enforce privileged actions to be only be performed if the administrator is present and using the console.

One advantage of our framework is that we are able to give reliable security guarantees even when the host is compromised. Without the fusion of sensor data from the environment surrounding the host, the attacker can simply hide inside the host or erase traces of an attack or intrusion. Some malware may even be able to shut down IDSs deployed in the host. A limitation of our framework is that the sensor data obtained is coarse grained and possibly noisy. Nevertheless, our evaluations show such coarse data is already useful in identifying certain mismatches between the host's and user's physical activities.

The rest of this paper is organized as follows. Sec. 2 introduces the framework of integrating the data from external sources to the access control or intrusion detection logic. We apply the framework to malware detection in Sec. 3 to demonstrate that information external to the host enhances the detection of malware activity. The framework is extended to rate and access control in Sec. 4. Related work is discussed in Sec. 5 and Sec. 6 concludes.

2 The Framework

Fig. 1 illustrates the relationship among different entities in our framework. The host considered in this paper is a stationary computer which typically includes keyboard, mouse, hard-disk, CPU, monitor, etc., and is operated through keyboard and mouse. Under this framework, *users* are persons who are directly accessing the computing resources. To access the computing resources, a user needs to be in the proximity of the host and interacting with it directly through the keyboard, mouse and display. Alternatively, users or attackers could be accessing the host remotely through a network connection. All network traffic in and out of the host is channeled through some routers. We consider all information processed and stored in the host as *internal* information. Potentially, internal information can be manipulated by an adversary if the host is compromised. We are more interested in the *external* information. In general, we call sensors installed outside the host *external* sensors, and the information gathered *external* information. External sensors could be an infrared sensor detecting whether a user is sitting in-front of the host's display, or a sensor installed in the router logging traffic information. The infra-red sensor is an example of an *environment sensor* gathering information from the physical environment, instead of the computed data from the host or routers. Information gathered can be classified into two types:

1. Information obtained from measuring computing resource usages, e.g. a temperature sensor measuring temperature near the motherboard (corresponding to CPU load), and a microphone to listen to sounds from the disk.
2. Information measuring user's activities, e.g. infrared sensors to detect user presence, pressure sensor to measure keyboard typing, etc. One method uses infra-red in a similar way as the common shop entrance alarm system. A reliable method is to derive the user presence from video captured by camera [14,15] but that may raise privacy concerns. However, video privacy was found to be an acceptable tradeoff to users [15].

Although we do not exclude the use of surveillance cameras as the environment sensors in our framework, the issue of user privacy must be taken into consideration in an implementation of the framework. A microphone not only can detect keyboard activity, but it can also record conversation among the users. A camera recording the user or display can also violate workplace privacy policies. Hence, we mainly consider binary information on user activities, such as whether a user is present, or detecting keyboard activities. Such information can be captured by sensors that are designed to give binary output or other coarse information, which alleviates privacy concerns, e.g. an infra-red sensor that detects user presence, or a camera that only outputs the detection outcomes.

All information gathered is channeled to a monitor which makes decisions. External sensors should communicate to the monitor securely to ensure the host is unable to compromise its integrity, authenticity and confidentiality. One solution is to have a separate private network for the sensors, e.g. a wireless sensor network with the external sensors as the nodes. Alternatively, the communication can still be tunneled through the host using cryptographic means. The privacy requirements and the need for a separate private network fit well with wireless sensor networks equipped with multi-modality sensors. There are many commercial wireless multi-sensor boards which fit our purposes, e.g. SBT80 from the EasySen [4] contains a number of sensors including infra-red, temperature and acoustic.

Besides wireless sensor networks, many physical security systems can provide relevant information for our monitor. For example, the door entry control system

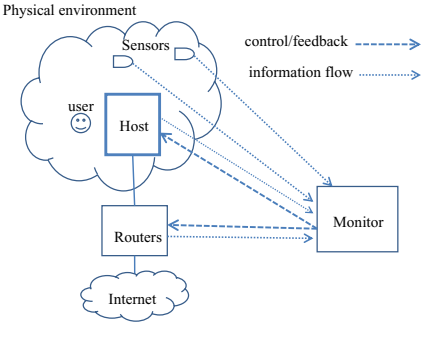


Fig. 1. The components of the framework

can give information about who gained access to a machine room, while surveillance cameras could reliably detect the presence of users near a console or other devices like scanner and printer. Although physical security systems are costly, some organizations could already have such systems in-place, together with a management system that is able to collect and process the data.

It is possible for an attacker to gain information similar to the sensory data, if the host is compromised, e.g. CPU temperature can be collected by the host and user presence can be inferred from keyboard or mouse events. Nevertheless, the CPU temperature and user presence information collected by sensors are still authentic and not subject to tampering.

3 Applying to Malware Detection

We now apply our framework to malware detection and give three malware detection implementations which detect email spamming, DDoS attack and password cracking.

A simple setup is to use two kinds of environment sensors. One detects whether the user is using the host, i.e. sitting at the machine. This can be done in a variety of ways, ranging from motion sensors to infrared sensors to video cameras. Another sensor records the temperature near the CPU. The first class of sensor records user activities while the second class measures the usage of computing resources. Network traffic of the host is also monitored at the router.

In our experiments, the malicious activities are carried out by a modified Agobot worm (also known as Gaobot). The worm sends spam email to other email accounts using the SMTP protocol, carries out a DDoS attack by flooding a target with UDP packets, and consumes CPU resources on the host to perform password cracking by hashing a dictionary of possible passwords.

The basic idea behind our detection rule is simple: the patterns of legitimate resource usage when the user is interacting with the host, is different from that when no user is present. If malware does not have the user presence information, its behavior will not be correlated with the user presence. We divide the time into intervals, in each interval, the user is either present or absent. The changepoint detection algorithm [5,6] is then applied to each interval to detect malicious activity. Table 1 gives an overview of the detection rules.

Table 1. Overview of malware detection rules

Malware Threat	Rules for triggering alarms
Email Spammer	(i) No user is present, and at least one email is sent; or (ii) A user is present, and changepoint detection decides that the cumulative sum of email sent exceeds a threshold.
DDoS Zombie	Changepoint detection detects the cumulative sum of the net outgoing packet rate exceeds thresholds for user present and absent
Password Cracker	Changepoint detection detects the cumulative sum of CPU temperature exceeds a certain threshold when user is absent

Table 2. Detection time of different spam worms. (Detection threshold $N = 120$ emails in $t = 6$ hours at user presence, and $N = 1$ during user absence.)

Spam worm	User present (min)	User absent (sec)
Storm	6.1	4
Rustock	3.6	3
Srizbi	0.07	< 1

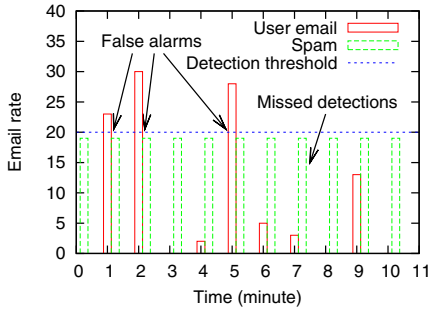


Fig. 2. False detections caused by email rate based spam detection

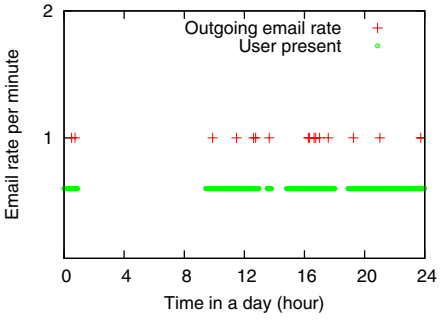


Fig. 3. Samples of user email rate

3.1 Changepoint Detection

The problem of changepoint detection [6] is to detect changes from normal behavior. We use a customized version of CuSum [7] to do such detection. In our model, behaviors are described as a real-valued time series of x_n. Normal behavior is estimated to have a value a or smaller. To detect an increase from the normal behavior, the following cumulative sum S_n is computed.

$$\begin{cases} S_0 = 0, \\ S_n = S_{n-1} + \max(0, x_n - a), \quad n \geq 1, \end{cases}$$

where x_n is the observation at time n from the sampling process. The value of a can be determined by the system administrator, or by observing samples of normal behaviors to select the smallest acceptable normal upper bound of x_n. If S_n exceeds the threshold N, a change point is detected at time n and the alarm will be triggered. Note that a can be exceeded when normal behavior is violated, so a is a lower detection threshold of abnormal behavior, S_n is the actual signal and N is the changepoint detection threshold. CuSum can be shown to be optimal under certain conditions [6]. As it is simple and efficient, it is suitable for real-time processing of streaming sensor data.

We use spam detection to show the difference between rate based detection and changepoint detection. Fig. 2 illustrates situations where rate based detection causes false positives and false negatives. Occasionally, a user email rate

exceeds the spam detection threshold, causing false alarms, even though the high rate is a transient. On the other hand, spam malware can control its email rate to just below the threshold, by spreading the operation over a long period of time. Rate based spam detection do not report such cases correctly, while changepoint detection can. With changepoint detection, we can set the upper bound of normal email rate as a baseline, e.g. one email per minute, then accumulate the excess amount of emails over a time period. Thus, it computes within a time interval, the area in the graph enclosed below by the baseline and bounded above by the email rate curve. With changepoint detection, occasional high email rates do not cause false alarms, and constant medium email rates cannot evade detection.

Changepoint detection relies on empirical or administrative thresholds, advanced attackers may gain information on user presence and adapt their behavior accordingly. Our mechanism cannot fully stop such malware, but it effectively mitigates the malicious activity.

3.2 Experimental Setup

Our experimental setup consists of one router and several hosts connected by 100Mbps connections to the Internet through the router. First, we gather normal data from three uncompromised hosts for a period of 12 days. Next, we "compromise" one host by installing the modified Agobot on it and control it from another machine. The compromised host carries out different activities on demand: generate spam email, start a DDoS attack and perform password cracking. More experimental details are described in the relevant section.

Three types of sensor data are collected on the hosts: (*i*) user presence; (*ii*) CPU temperature; and (*iii*) network traffic. User presence is represented by a binary-valued time series, indicating whether the user is sitting at the host. We use infrared sensors to detect user presence. This is more reliable compared to using acoustic sensors to detect keyboard typing or mouse clicks. One reason is that a user can be simply looking at the monitor. Another reason is that malware can fool the sensor by playing back clicks. The CPU temperature is represented by a real-valued time series measuring temperature at the processor, which correlates to CPU load. In principle, the CPU temperature should be measured with a sensor between the CPU heat sink and fan, but we had difficulty in doing so with our sensor. To prove the concept, we used the CPU temperature obtained from the host's operating system as a proxy.[1] We remark that the sensor data should be measured and sent to the monitor via a secure channel, and thus our temperature readings is only a simulation to simplify the experiments. For network traffic, headers of all packets and partial payload passing through the router are logged.

3.3 Spam Detection

Spam can be sent out at different rates. Some worms like Storm and Rustock have been reported to send at lowish rates of 20 and 33 emails/min while others

[1] We use the system call *IDebugDataSpaces–>ReadMsr*() in Windows.

like Srizbi's rate is 1800 emails/min [12]. While 20 emails/min is probably too high for humans, malware could also send at lower rates. We remark that humans might also send email at high rate, e.g. when using a script to send emails to a group of recipients.

Our experiment is meant to show reliable spam detection when data from external sensors is included. We also experiment on different modes of sending email, namely, SMTP-based clients and also webmail clients.

Spam Detection Using the Framework. To detect bots that send spam, we use the following rules: (i) when the user is absent, any outgoing email flags a spamming activity; and (ii) when the user is present, changepoint detection is applied on the number of emails sent. Essentially, if over N emails are sent within a time interval of length t, the algorithm flags it as spamming activity, where N and t are the two tunable parameters.

The first rule relies on the fact that emails are usually directly sent to the mail servers when the user hits send. Any scheduled delay in delivery is based on the server itself. The actual value of N and t in the second rule can be learned from the normal traffic for each host. Although the rules are simple, no matter how slow the spam rate is, we can detect spam when the user is absent. So if a stealthy spam program sends out only a single email at night, this could slip out unnoticed by normal email rate based spam detection while we would detect it.

We apply CuSum to detect changes in the amount of emails sent when the user is present. We incorporate a limit on the accumulation time t, and set $a = 0$, $N = 120$ and $t = 6$ hours. The effect is that whenever there is an outgoing email, we accumulate the count, and no more than $N = 120$ emails can be sent in 6 hours. If the accumulated sum exceeds 120, we raise a spam alarm for the host. The allowable average email rate is three minutes per email. It is high for a human user, since users do not consistently send an email every 3 minutes for 6 hours. When user is absent, we set $N = 1$, meaning any outgoing email indicates a spam.

Experimental Results on Spammer Detection. The detection of outgoing email is done by matching packets with a list of signatures. Table 3 shows the signatures of email sent using SMTP and webmail. Email sent using SMTP protocol is relatively easy to detect, since an SMTP command `MAIL` indicates the user is submitting email. Webmail interfaces are more complex – Table 3 summarises how we identify emails sent using Hotmail, Gmail, NetEase, and SquirrelMail. The destination IP is first matched with a set of possible known servers. Next, we examine the first few bytes of packet payload to look for HTTP request method `POST`. Existence of such a method indicates that the client could be submitting email, logging into the mail server, making a request to retrieve email, requesting for the email listing, or requesting housekeeping operations. To determine whether the client is sending an email, different checks are carried out for different mail services. For Hotmail, we look for URI that starts with `mail/SendMessageLight.aspx?`. For NetEase, the request URI ends with `&func=mbox:compose`. For Gmail, the request URI contains `&view=up&act=sm`.[2]

[2] Gmail uses HTTPS from late Jan. 2010, but our experiments were performed earlier.

Table 3. Rules for email detection

SMTP	the SMTP request command is `MAIL`.
`hotmail.com`	The destination IP is in the set of `hotmail.com` server IPs, the protocol is HTTP, the HTTP request method is `POST`, and the HTTP request URI starts with `/mail/SendMessageLight.aspx?`
`netease.com`	The destination IP is in the set of `netease.com` server IPs, the protocol is HTTP, the HTTP request method is `POST`, and the HTTP request URI ends with `&func=mbox:compose`
`gmail.com`	The destination IP is in the set of `gmail.com` server IPs, the protocol is HTTP, the HTTP request method is `POST`, and the HTTP request URI contains `&view=up&act=sm`
SquirrelMail	A particular HTTP request immediately after an email is sent.

To perform these checks, we need to read 33 bytes into the HTTP message for Hotmail, 64 bytes for NetEase and 80 bytes for Gmail. Hence during packet captures, the packet payload needs to be logged partially. Detecting SquirrelMail is less straightforward as the payload is encrypted. We found that immediately after an email is sent, there is an HTTP Get request in plain text of a large size to fetch the listing of the current folder. The signatures enable us to identify the sending of email reliably. Compared to a typical spam filter that inspects the email content, our method has less privacy concerns.

Fig. 3 shows the typical email activity of a user in a day. About 10 to 20 emails are sent daily. It shows that our hypothesis that all emails are sent with the user present is reasonable.

In our experiment, spam is sent from the monitored host using the modified Agobot. We tested the detection of spam at rates corresponding to Srizbi, Rustock and Storm worms. Table 2 shows the detection time when the user is present and absent. Note that threshold N is set on the parameter of the accumulated email amount. It is easy to see the detection time is inversely proportional to the spam rate. Since the spam rate of Srizbi is much higher than Rustock or Storm, it takes least time to detect. When the user is present, the detection time of Srizbi is less than 0.1 minute and about 4 and 6 minutes for Rustock and Storm respectively. When the user is absent, all three spam worms are detected instantly, because the detection threshold $N = 1$ at user absence.

3.4 Detecting DDoS Zombie Attacks

This experiment deals with detecting zombies which carry out UDP packet flooding for a DDos attack. In this attack, the compromised host sends out UDP packets to a victim to consume the victim's network bandwidth. When the user is absent, legitimate network traffic rate is low and thus we can potentially detect the malicious UDP packet flood by observing the traffic rate. However, there could be background processes that generate network traffic, e.g. automated updates. Another scenario in our experiment is a legitimate P2P program. The P2P program constantly generates network traffic even when the user is absent.

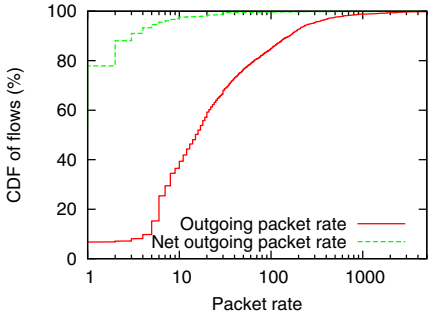


Fig. 4. Difference in the outgoing packet rate and the net outgoing packet rate of 2351 active TCP flows during user presence

Hence, it is desirable to derive another feature, instead of the overall rate, to distinguish UDP flood.

We observe that the UDP flood generates one-way traffic, whereas typical legitimate processes generate two-way traffic. This motivates us to consider the net outgoing packet rate p_{net},

$$p_{net} = \max(p_{out} - p_{in}, 0),$$

where p_{out} and p_{in} are the outgoing and incoming packet rate respectively. We apply CuSum on p_{net}, but with a different threshold a when the user is absent and present.

In addition to p_{net}, we also monitor the ratio r of outgoing packets that are not responded to,

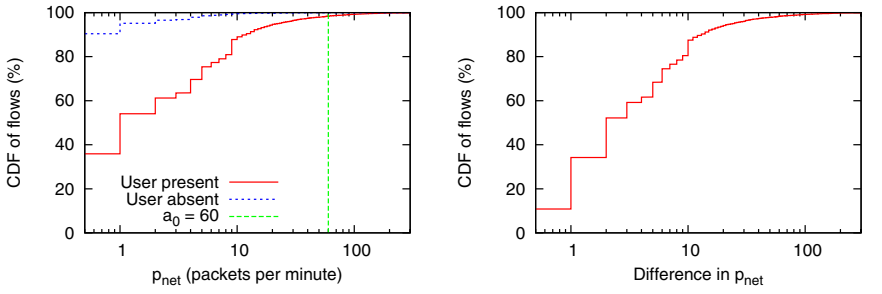
$$r = p_{net}/p_{out}.$$

We have $0 \leq r \leq 1$ where $r = 1$ if the flow has only outgoing packets; and $r = 0$ if $p_{out} \leq p_{in}$. The excess of p_{net} over a is accumulated only if $r \approx 1$.

Fig. 4 compares the maximum net outgoing packet rate p_{net} and the outgoing packet rate p_{out} of 2,351 non-attack TCP flows, observed in 10 minutes. A flow is identified by (local IP, remote IP, transport protocol) – port numbers are not differentiated as attackers may open multiple ports to flood the same victim. Over 60% of the flows have the maximum $p_{out} > 10$ packets a minute; whereas less than 5% of the flows have the maximum $p_{net} > 10$. In some flow, maximum p_{out} is 2,443 packets per minute, but the maximum p_{net} is as low as 10 packets.

Fig. 5 shows the distribution of p_{net} for 13,620 flows. Fig. 5(a) shows the net outgoing packet rate is close to 0 for most flows. When the user is present, 35% of the flows have $p_{net} = 0$; and 80% flows have p_{net} less than 10 packets a minute. When the user is absent, 90% of the flows have $p_{net} = 0$. Fig. 5(b) shows that the difference in p_{net} when user is present and absent, it can be as large as 600 for some flows, so different upper bounds of normal p_{net} should be used for user presence and absence and for each flow.

From Fig. 5(a), initializing the upper bound of normal p_{net} to be $a = 60$ packets per minute is sufficient for most flows. Parameter a can be lowered by



(a) Distribution of flow net outgoing packet rates

(b) Difference in the flow rates when user is present and absent

Fig. 5. Distribution of the maximum net outgoing packet rate p_{net} with 13,620 TCP and UDP flows, each flow is observed for 10 minutes during user presence and absence

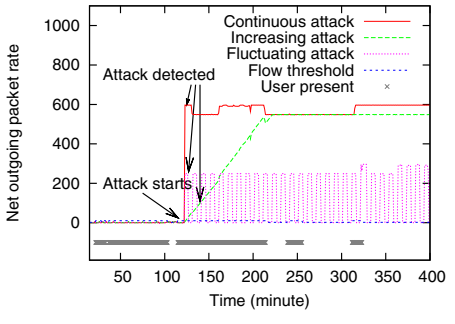


Fig. 6. Net outgoing packet rate of the DDoS attack flow in different attack patterns

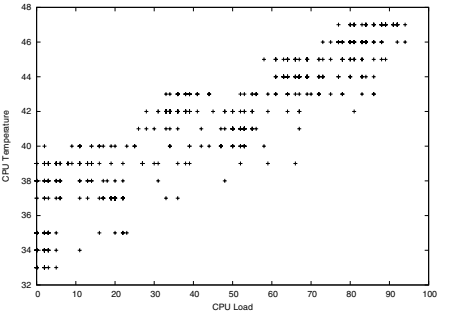


Fig. 7. Correlation of CPU load and CPU temperature

examining the online traffic. To accumulate the excess of p_{net} over a, r must be close to 1 and we set it at 0.95. The threshold to trigger an alarm is $N = 800$ packets, accumulated over 20 minutes. It ensures an attack flow can be detected if at least 2 UDP packets are sent a second on average.

Some non-attack flows have large net outgoing packet rate. However, they do not cause false alarms, because the outgoing packets are responded to, or r is not close to 1. In terms of absolute value, $p_{net} = 1278$ is high, but its $p_{out} = 2,812$, making $r = 0.45$, so there is 1 response in about 2 packets, the ratio is reasonable and hence the high net outgoing packet rate is also accepted.

Fig. 6 shows the net outgoing packet rate p_{net} of the attack flow under 3 different attack scenarios. The upper bound a of normal p_{net} is 10 and 2 packets a minute for user present and absent respectively. These bounds are determined from analyzing the training data of the flow. The attack starts at the 125th minute. For the continuous attack, p_{net} sharply increases to around 600. After

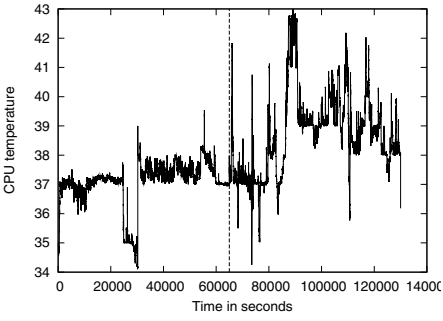


Fig. 8. CPU temperature variation during user absence and presence. User is absent from 0 to 64,000 second; and present from 64,000 second onwards. The vertical line separates user absence from presence.

1.36 minutes, the alarm is triggered. The increasing rate attack increases its attack intensity very slowly to delay detection. This attack might not be detected if the detector adapts to the flow rate in near real time. Our approach detects the attack in about 15 minutes, long before it reaches its peak rate. The fluctuating attack in Fig. 6 constrains the attack intensity, and releases attack traffic in pulses. This is to avoid detection if average traffic rate or peak rate is used by the detector. Our approach detects the fluctuating attack within 3 minutes.

3.5 Detecting Misuse of Compute Resources

Bots are often recruited to invert cryptographic functions to break passwords. Inverting such functions requires extensive computations which are distributed to the bots. To detect such activities, we look for increases in the CPU load after user has been absent for a while. As CPU load is internal to the host, we rely on CPU temperature as a measure of the CPU load.

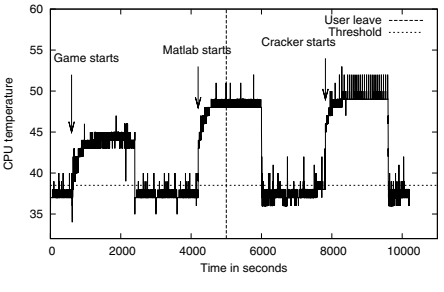


Fig. 9. CPU temperature variation during various activities

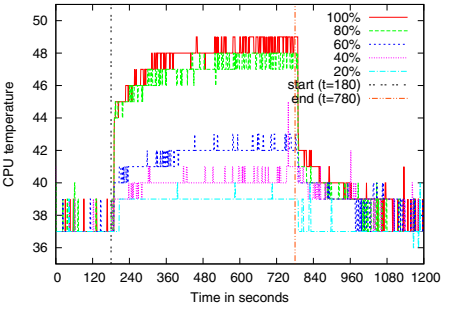


Fig. 10. Correlation of attack intensity and CPU temperature

Fig. 7 shows the correlation between CPU load and temperature. The correlation is good but it includes fluctuation and noise. Fig. 8 shows the temperature when a user is present and absent. Note when the user is absent, the temperature is around 37°C, and varies less. We measured the temperature during automated software update and found the temperature increase of 2-3°.

We employ CuSum to detect the increase in CPU temperature. Fig. 8 shows normal temperature variations during user presence and absence. From the observation of Fig. 8, we set the upper bound of system idle temperature to be $a = 38.5$°C. We set the threshold for the accumulated increase in temperature as $N = 2400$ in 30 minutes. This threshold N is chosen to allow software update that increases the CPU temperature by 2°and lasts for 20 minutes. We set a grace period of $g = 10$ minutes at the transition from user presence to absence, that is, if the changepoint is detected within 10 minutes after a user leaves, the alarm is not activated.

Fig. 9 shows the temperature during various computation tasks. The changepoints in temperature for playing games and running matlab are detected 10.4 minutes and 5.2 minutes respectively after they start. Since the user is present or just left when the changepoint occurs, the activities are considered initiated by the user and accepted as normal. The password cracker activity triggers an alarm 4.5 minutes after it starts. Since the changepoint is detected when user is absent, it is considered as malicious computation. If the malicious computation executes during user presence, it can be noted by the user.

We also conducted an experiment which varies the intensity of the CPU usage. This investigates if the CPU usage can be detected when the malware throttles its computation at a lower rate. CPU usage is controlled as follows: for each one-second interval, the bot computes at full speed for x seconds and then sleep for $1 - x$ seconds, where $x < 1$. We denote the CPU usage to be $100x\%$ in Fig. 10. We carry out the attack for 600 seconds (starts at 180 seconds and ends at 780 seconds). Fig. 9 shows the temperature under intensity variation. Note that after the attacks stop, the temperature gradually returns to normal. For the password computation running at 100% usage, we detect the change after about 4.5 minutes. For the computation at 20% usage, we can also detect the change if the computation lasts for more than 20 minutes. To evade detection, the malicious computation can only further reduce its CPU usage or shorten its execution time. Thus, our system effectively limits the amount of malicious computation.

We need to handle carefully the non-malicious programs that commonly run when no user is present. Besides system auto-updates, these programs include nightly backup, nightly virus scan, remote desktop and user scheduled computation. Backup and virus scan are regular tasks. Their resource consumption can be profiled, in terms of the start time, duration, and the increase in CPU temperature. Based on our measurement, system backup only increases the CPU temperature by 1-3°. Virus scan usually increases the CPU temperature by 2-3°. These host profiles are basis of a whitelist kept by the monitor to suppress false alarms. Even if attacker tries to hide the malware execution by running it at

the same time as these tasks, the excess increase in temperature or duration will still trigger the alarm.

User scheduled computations may cause irregular behavior. As the execution of scheduled computation can be planned ahead, the user simply declares in advance to the monitor the estimated process start time, duration and CPU load (converted to temperature increases by the monitor). The monitor suppresses any false alarm within the specified resource consumption. The user is advised to give a conservative estimation, for it is better for the user to intervene if excess resources are consumed, than allowing the attacker to free ride.

4 Application to Access Control and Rate Control

Here we investigate how environment information can be useful in controlling and allocating resources. A security policy may require administrators to be physically inside the machine room to access server consoles and administrative tools. In order to mitigate malware from sending spam email, we could implement rate control in the router or the mail server, and the rate limit is based on user presence. In both cases, environment information is used as a condition to access certain resources. Note that the first case is on access control, while the second is on rate control.

4.1 Access Control

Our framework can be used to implement location-based access control. One location-based access control scheme was proposed by Ardagna et al. [8] to restrict access to certain resources based on physical location of the user. In their work, the physical location is obtained from mobile devices, such as mobile phones carried by users. Our framework also implements location-based access control, but in a different way. Their work adopts a user-centric approach where the device is attached to the user and user's location is measured in order to figure out which resources can be accessed. We adopt a resource-centric approach where the device is attached to the resource and user presence is observed near the resource in order to decide whether to grant the access. Both approaches have advantages and disadvantages.

Our access control policy not only incorporates user presence information but also user activity information. For example, the user activity information can be "the user has typed some keys". Enforcement is implemented both on the host and router depending on the type of the resource. Enforcement implemented on the host implicitly assumes that malware is not in control of the host and the host decides the access based on environmental information gathered by the monitor, but the malware cannot affect access control policies at the environment level. Here we give two access control policies as examples.

1. *The user can execute the* `/usr/bin/sudo` *program only if he is sitting at the host.* This policy is used to mitigate the problem of remote attacks - it requires that there is a human present before the sudo operation is allowed.

This policy has to be enforced on the host. A remote attacker who does not yet have control of the host would be prevented from performing actions which could be used to infect the operating system.

2. *The user can send email only if he is sitting at the computer and has mouse or keyboard activity.* This policy is used to prevent malware from sending email while the user is away. Since user activity is enforced in the policy, it also prevents sending email while the user is idle, e.g. watching a movie. The intuition is that the user must have performed some typing or mouse action in order to send an email. Unlike user presence, which can be thought of as being a continuous signal, mouse and keyboard input are discrete events which may happen at time points which do not overlap with the email sending time interval. Thus, the precise meaning of the policy is "An email can be sent at time t if user presence is observed at that t and there is mouse or keyboard input between $[t - \Delta, t + \Delta]$". This policy is enforced in the router or the mail server instead of the host and thus provides enforcement even when the host is compromised.

4.2 Rate Control

Previously we showed that abnormal resource usage when the user is absent can be easily detected under several scenarios. This can be easily extended to controlling the resource usage rate in the case of activities involving external resources. Two natural scenarios of external rate control are:

- **Shaping Network Traffic**
  To mitigate computers being used as bots to perform DDoS attacks, router can shape network traffic based on user presence information. By limiting the traffic on flows, this becomes a form of inverse quality of service, providing reduced quality when the user is not present. If P2P programs are being used, this would save some network bandwidth, e.g. if the host uses Skype and becomes a Skype supernode, it could lose its supernode status under the reduced network flow.
- **Sending Email**
  Similarly, the emails could be simply denied when the user is absent. If the threshold is above zero, then for both webmail and SMTP, the router can simply rate limit the protocol. Alternatively for an SMTP server, it could quarantine email beyond the threshold. In both cases, outgoing spam emails are rate limited possibly to zero.

  The email rate can also be rate limited when the user is present. The idea is that email is based on typing and/or mouse clicks. This data can also be recorded using environmental sensors. The email rate can then be based on a function of the detected keystrokes and/or mouse clicks.

The framework can make resource rate limiting policies more flexible and usable. The idea is to have feedback if a usage is too high for a resource. External sensors can then be used as a secure channel to request a higher resource rate.

For example, a monitor on the host can display the resource usage and warn the user if he is reaching the limit. The user can have something as simple as a button which is pressed to function as the secure request channel to obtain more network bandwidth or more emails. Notice that such a policy cannot be done securely without the help of external environment information.

In the access and rate control application discussed above, to control the resource utilization, essentially we need to verify whether an entity is "human". Instead of using external environment to infer user presence, alternatively, CAPTCHA [9] or graphical Turing test could be implemented to verify that the user is human. Using the environment information has the advantage that the users are not interrupted by the challenges issued by the graphical Turing test.

5 Related Work

If we consider a computer host to include the host, the user channel and the network channel, then host security can be divided into: (*i*) software security, which ensures the software running in the host is authentic, e.g. antivirus [19], system call filtering [16] and binary authentication [17]; (*ii*) user security, which ensures the user is authentic, e.g. password/biometric authentication, physical perimeters and surveillance camera monitoring; and (*iii*) network security, which ensures the network communication is authentic, e.g. personal firewalls [18]. Our approach to enhance host security is substantially different from the existing designs in three aspects. Firstly, the model we propose fuses data from a few channels of external environment sensors to monitor the host activity. Secondly, as our model does not require controlling or modifying the host OS or software, it is able to provide some security even when the host is compromised. Thirdly, our system detects outbound or on-host malware execution, which complements intrusion detection.

There are a few works which correlates information from different channels to improve host security. The system BINDER [10] correlates user events (user input), process events (process creation and process termination), and network events (connection request, data arrival and domain name lookup) to detect malware. Both our malware detection system and BINDER correlate user presence information and system behaviour to detect malware. BINDER uses information collected by the user's machine, which potentially could be manipulated by the compromised host.

The systems proposed by Gu et al. [11] and Yen et al. [13] detect botnets by analyzing and correlating network traces. The two systems and our malware detection system use information from the network router rather than the host in question so as to be able to deal with the case when the host is compromised and gives false information. The difference with our malware detection system is that we have user presence and activity information in addition to network information.

Kumar et al. proposed a system [14] which continuously monitors user's biometric identity and locks up the computer if it cannot detect the correct user.

Both their system and our access control system use physical user information to provide additional factor for authentication. There are two main differences between the two. Firstly, our system only detects user presence information, while their system detects user's biometric information which is much stronger but gives less privacy. Secondly, their system runs entirely in the user's machine, thus it cannot guarantee the authentication once the machine has been compromised.

Location-based access control (LBAC) has been actively researched on wireless networks, e.g. [20]. The model in LBAC assumes user devices are mobile, their locations are tracked for service continuity, or verified before granting access. The problem we address is different in that we are not dealing with mobile devices, instead our focus is on utilizing a combination of environment data to enhance security, such as to detect malware on the host and to regulate the usage of resources by the host.

6 Conclusion

In this paper, we proposed a framework that incorporates environment information in securing host computers in a few ways: by using the environment information as an additional source of information for malware detection, or by integrating the environment information with existing conditions in rate-control mechanisms and access control policies. We argued that, since the sensors are "external" with respect to the host, they are difficult to be accessed and tampered by a compromised host. Furthermore, by investigating several applications, we showed that the simple and coarse information on user activities and resource usages is sufficient to provide good performance in malware detection, and is useful in expressing certain rate-control and access control policies. We have also identified a few important requirements of the sensors, in particular, the concerns of user-privacy and the need of a secure channel. Thus, we have proposed a simple and effective framework for security enhancement which is arguably safe against compromise by attackers.

The framework also takes advantage of the growing popularity of pervasive computing and sensor networks. As the trend in cost of wireless multi-modality sensors is decreasing, applications of our framework are feasible for cost-effective deployment in the near future.

References

1. The Myth of The Four-minute Windows Survival Time, `http://www.edbott.com/weblog/?p=2071`
2. Unpatched PC 'Survival Time' Just 16 Minutes, `http://www.informationweek.com/news/showArticle.jhtml?articleID=29106061`
3. Conficker, `http://en.wikipedia.org/wiki/Conficker`
4. EasySen SBT80 Product Page, `http://www.easysen.com/SBT80.htm`

5. Wang, H., Zhang, D., Shin, K.G.: Detecting SYN Flooding Attacks. In: IEEE InfoCom (2002)
6. Basseville, M., Nikiforov, I.V.: Detection of Abrupt Changes: Theory and Application. Prentice-Hall, Englewood Cliffs (1993)
7. Page, E.S.: Continuous Inspection Schemes. Biometrika (1954)
8. Ardagna, C.A., Cremonini, M., Damiani, E., di Vimercati, S.D.C., Samarati, P.: Supporting Location-Based Conditions in Access Control Policies. In: ACSAC (2006)
9. Von Ahn, L., Blum, M., Hopper, N.J., Langford, J.: CAPTCHA: Using hard AI problems for security. In: Biham, E. (ed.) EUROCRYPT 2003. LNCS, vol. 2656. Springer, Heidelberg (2003)
10. Cui, W., Katz, R.H., Tan, W-.T.: Design and Implementation of an Extrusion-based Break-In Detector for Personal Computers. In: ACSAC (2005)
11. Gu, G., Porras, P., Yegneswaran, V., Fong, M., Lee, W.: BotHunter: Detecting Malware Infection Through IDS-Driven Dialog Correlation. In: USENIX Security (2005)
12. John, J.P., Moshchuk, A., Gribble, S.D., Krishnamurthy, A.: Studying Spamming Botnets Using Botlab. In: NSDI (2009)
13. Yen, T.-.F., Reiter, M.K.: Traffic Aggregation for Malware Detection. In: GI Intl. Conf. on Detection of Intrusions and Malware, and Vulnerability Assessment (2008)
14. Kumar, S., Sim, T., Janakiraman, R., Zhang, S.: Using Continuous Biometric Verification to Protect Interactive Login Sessions. In: ACSAC (2005)
15. Kwang, G.K., Yap, R.H.C., Sim, T., Ramnath, R.: An Usability Study of Continous Biometrics Authentication. In: IAPR/IEEE Intl. Conf. on Biometrics (2009)
16. Provos, N.: Improving Host Security with System Call Policies. In: USENIX Security (2003)
17. Halim, F., Ramnath, R., Sufatrio Wu, Y., Yap, R.H.C.: A Lightweight Binary Authentication System for Windows. In: IFIPTM (2008)
18. Ingham, K., Forrest, S.: A History and Survey of Network Firewalls. Tech. Rep. TR-CS-2002-37, University of New Mexico Computer Science Department (2002)
19. Post, G., Kagan, A.: The Use and Effectiveness of Anti-Virus Software. Computers & Security 17(7) (1998)
20. Ardagna, C.A., Cremonini, M., Damiani, E., di Vimercati, S.D.C., Samarati, P.: Supporting Location-Based Conditions in Access Control Policies. In: ASIACCS (2006)