Countering Automated Exploits with System Security CAPTCHAS

Dinan Gunawardena[1], Jacob Scott[2], Alf Zugenmaier[3], and Austin Donnelly[1]

[1] Microsoft Research Cambridge, UK { dinang | austind } @microsoft.com
[2] UC Berkley, CA, USA jhs@ocf.berkley.edu
[3] DoCoMo Euro-Labs, Munich, Germany zugenmaier@docomolab-euro.com *

Abstract. Many users routinely log in to their system with system administrator privileges. This is especially true of home users. The advantage of this setup is that these users can do everything necessary to fulfil their tasks with the computer. The disadvantage is that every program running in the users context can make arbitrary modifications to the system. Malicious programs and scripts often take advantage of this and silently change important parameters. We propose to verify that these changes were initiated by a human by a ceremony making use of a CAPTCHA (Completely Automated Public Turing Test to Tell Computers and Humans Apart). We compare this approach with other methods of achieving the same goal, i.e. passwords, secure path and access control based on zone of origin of the code.

1 Introduction

Most users need to fulfil at least two roles when dealing with their computer. In one role they do their day to day activities with the system. In the other they perform systems administration. Most users are usually logged in with the maximum privileges that they ever need, which often are local system administrator privileges. It has often been proposed to give users minimum privilege, however in many cases that is not feasible as users will on occasion have to change crucial system parameters. Spyware, scripts, and viruses are usually crafted in such a manner that they initially are executed in the users context. They then change system parameters that allow them to do their evil deed, such as spy on the users network activity, change the dial-up phonebook entry to a premium rate number, or ensure that they are executed every time the computer is booted up.

We try to find a user friendly way in which the user can be asked for permission to change important parameters. This should happen in a way that cannot be subverted programmatically. Ellison [1] is the first to describe ceremonies as an extension of network security protocols. Ceremonies try to include the actual humans sitting in front of the computer as an important active entity that is vital to achieving security properties. We propose a ceremony ensuring a human authorises changes to protect important configuration parameters of a computer

* This work was done while the authors were at Microsoft Research Cambridge.

system from inadvertent change through the user or a program running in the users context. The concept is based on the system issuing a challenge to the user that can only be solved by a human. This challenge acts as a warning that something crucial is being altered and also as a barrier for automatic exploits.

This article is structured as follows: First we describe the concept of system security CAPTCHAS, the ceremony that proves that a human was involved in the protocol and not just a program that has taken over input and output. We then present a preliminary evaluation of the idea, based on a small prototype. Potential alternatives are discussed in Section 4. The paper finishes with conclusions.

2 System Security CAPTCHAs (SSCs)

In a Turing test a person tries to determine whether the entity responding to his or her inputs is a human or a computer [2]. Completely automated public Turing tests (CAPTCHAS) [3, 4] are in some respect a reverse Turing test – a computer is trying to determine that an entity responding to its inputs is a human. When combined with access control, they represent a way of ensuring that a human was involved in the decision to access a specific resource. The response to a CAPTCHA cannot be scripted. Web based email providers already successfully employ this technique to mitigate creation of a large number of email addresses that could be used to send unsolicited bulk email. Usually CAPTCHAS are based on a hard AI problem, such as recognition of severely distorted characters or audio streams.

These CAPTCHAS can now be used in a discretionary override of access control. The concept of discretionary override of access control [5] is based on the observation that currently deployed access control policies only have the option to allow or disallow an action. Sometimes an action should be disallowed at most times, but there should be a way to override this decision. System parameters (e.g. the Windows registry) are subject to access control, such as NTSEC. With System Security CAPTCHAs (SSC), the security policy can specify not only allow or deny, it could also require a user to correctly solve an SSC. The configuration change is either allowed or disallowed subject to the validation of the SSC. The system has to store some additional context or message to present to the user related to the parameter that some application has attempted to change (e.g. if this were an attempt to disable the firewall, an appropriate message to present to the user could be retrieved from a system string table).

To decide which parameters could be protected protect using SSC we did a cursory analysis of malware, and some brainstorming. The complete obviously depends on the system to be protected.

Some examples of parameters and API calls thet lend themselves to be protected with SSCs:

- Enabling or disabling the SSC configuration
- Automatic startup extensibility points [6]
- Firewall reconfiguration

- Virus protection configuration
- System software installation
- API to change security permissions
- API to access non-sandboxed resources
- Attempt to change telephone number for the system telephone dialler application (i.e. block rogue diallers)
- Maximum number of incomplete TCP connections allowed (Windows XPSP2 scanning worm propagation mitigation technique) configuration changes.
- Attempt to change registry settings regarding the network stack layers which is a favourite place to install Spyware
- Attempt to change the Start-up sequence (Autostart folder etc.)
- Enabling the exporting of a file system e.g. sharing C$ on the network should test whether a human is really attempting to perform the action.
- Verifying system activation and similar system licensing related API calls.

Because changes to the parameters often come in bulk, it makes sense to wrap the changes into a transaction that either commits all changes made by one process or discards all changes. The following list gives some examples of parameters and API calls that are unpracitcal to protect with SSCs.

- Every disk access
- Installation of signed/trusted software

2.1 A credit based extension for reducing CAPTCHA usability overhead

To reduce the overhead of asking the user for a CAPTCHA for every single security critical interaction, a scheme is possible whereby each security critical parameter has a virtual cost associated with its alteration (with that cost being proportional to the severity of the security impact of the parameter being changed). Users start with a credit threshold above which they must answer a CAPTCHA. If the CAPTCHA challenge is failed, the configuration changes are undone (back to the state at the last successful challenge or some well defined snapshot from a system point). The scheme is as follows:

1. System starts up, all security critical parameters have been audited and assigned a credit cost P_i for their reconfiguration by the user. Any parameter without an associated credit cost may be (optionally) given a nominal cost P_{default} as set by system policy. The system associates a SSC subsystem callback with each of the security critical parameters identified in the audit, the system startup parameter values are snapshot and stored in secure storage by the SSC subsystem.
2. When the user logs on, her SSC parameter modification cost U_p is set to 0. The SSC system retrieves the users associated SSC credit threshold U_t and stores it (with U_p) in secure storage. In addition, the SSC subsystem stores an empty list L_p in the secure storage (this will be used to store parameter change tuples).

3. When a user modifies system parameters that the SSC is monitoring, P_i is added to U_p.
   (a) If $U_p < U_t$, the parameter changed is added to a list of parameter tuples changed by the user (L_p) , along with the old value of the parameter i.e. ¡parameter identifier, old parameter value¿.
   (b) If $U_p \geq U_t$, the user is issued with a SSC challenge.
      i. If the challenge is passed, U_p is set to zero and the parameter change is allowed and the list of parameters changed is set to the empty list.
      ii. If the challenge fails, U_p is set to zero, all the parameters in Lp are reverted to the old parameter value and Lp is set to the empty list. The snapshot value for the last 'good' configuration parameter is updated to reflect the current parameter value.

2.2 Optional extensions of the credit based scheme

The following extensions are independent of each other. 1. Time period linked credit: The credit may be allowed to build up using some standard leaky bucket based scheme so that the value U_t is automatically and dynamically calculated based on the token bucket parameters associated with the U_t update as set by system policy. Obviously there is the danger of a script delaying its actions sufficiently that U_t is always 0.

2. Automatic and distributed calculation of P_i and U_t: To avoid a full audit of security parameters, $P_{default}$ can be set to some unit value. Every time a parameter is updated, Pi can be incremented by $P_{default}$. If any P_i exceeds some threshold P_g set by group policy, the parameter identifier can be forwarded by the SSC subsystem to a group administrator, following a local SSC challenge. The administrator may then decide whether this parameter is security critical or not and push down a fixed P_i to all users in the administrative group, by group policy. The same mechanism can be used to set U_t dynamically in this case, P_i values may be dynamically calculated or fixed. If U_p exceeds some threshold U_g set by group policy, the parameter value list L_p can be forwarded by the SSC subsystem to a group administrator following a local SSC challenge. The administrator may then decide whether the L_p chain is too long or too short for the level of security required and push down a fixed U_t to all users in the administrative group by group policy.

Either of the above schemes could be used whilst a system is under test (prior to deployment) to calculate reasonable static values for per configuration parameter credit costs or thresholds.

2.3 Possible drawbacks of the credit based scheme

If unit cost is known to attacker, changes can be aimed to stay below threshold. Accurately determining the criticality value of a system parameter is difficult. In addition, criticality of parameters may not be additive. Many of these problems are mitigated if commit happens on exit of program and SSC will be raised in all cases if $U_t > 0$

2.4 Non security centric applications for SSC

Some ideas for applications that go beyond the protection of API calls or the Trusted Computing Base (non-User/Kernel e.g. Master Boot Record, BIOS, firmware, microcode etc.) that have already been suggested include:

Human in the loop check for *safety* critical systems: There are numerous computer systems / processes that are critical to safety e.g. nuclear power plant monitoring, railway/flight traffic control systems, security monitoring of premises where it is important to ensure that the system is being actively monitored by a human. In these scenarios, if a malicious script or automated process were to be able to subvert the 'human in the loop' test then there might be dire consequences for safety. It is possible to use a SSC to verify the continued presence of a human being to avoid the threat. This somewhat akin to a 'dead man's hand' switch concept.

Service still in use (by a human) for billing purposes: With the migration to software as a service, one possible billing model is usage (time) based billing. Current billing models for software are usually based on upfront license fees and the use of some explicit of implicit license server. This model is suboptimal when the user can rent the use of an application for shorter periods on an ad-hoc basis. One problem with distributed application renting is it is hard to distinguish user inactivity from user session completed but not logged out correctly. This has a ramification for billing and charging. I.e. the billing system will bill the user from the last time the user interacted with the system plus some inactivity time-out (in the case where the user did not correctly terminate the session e.g. by logging out). One could suggest a simple 'this application has been idle for some time, would you like to keep this application open YES/NO' type query. This could be subverted by malicious automated software on the users system (the incentive for the malicious software is to keep a usage based billing channel open to unduly load/deny service at the server or to increase the revenue for a disreputable service provider). It is possible to use a SSC instead of this simple 'YES/NO' dialog to avert this problem. Consequently, there might be fewer billing queries from the user.

3 Evaluation

This evaluation is based on a small prototype that protects a chosen registry key using a SSC.

Security: Even though the program being run in the users context has full permission of the user, it cannot make full use of them. In our example human intervention is necessary to alter the protected registry key. This prevents automatic exploits from changing that registry key.

Possible attacks: For a program it may be possible to copy the SSC into different context, i.e. into a window claiming to change a different parameter than what is actually changed. But then, as soon as the SSC window pops up user should become suspicious anyways. In addition, responding to an SSC requires more effort than just clicking o.K. on a popup dialog box. In addition, just clicking o.K. could be scripted by a program taking over the user input.

The malicious program could try to proxy the SSC to some real person over the Internet if necessary, this may be mitigated by not permitting network I/O while a SSC is being presented. This also requires the SSC to time out after a short period, so that network I/O can resume. An alternative approach so that unwitting users are not tricked into solving CAPTCHAs would be to bind the content of the CAPTCHA somehow to the request that triggered it.

Furthermore, a program that can access the security function directly (e.g. has kernel access) can circumvent the protection.

Usability: We have not done a proper usability study, yet. The usability will very likely depend on how often the SSCs get triggered. If it happens rarely (less than once a day), we expect the SSCs to be a real help. This depends very much on careful selection on the parameters to be protected. Software installation may require the installation of a large number of services and/or configuration changes to the system. If a CAPTCHA were requested for each of these changes, the user would quickly disable the SSC protection owing to intrusiveness and lack of usability.

Choice of CAPTCHA content: Whilst the exact content of the CAPTCHA presented to the user is outside the scope of this document, the recommendation is the CAPTCHA present a challenge that is contextually relevant to the security configuration change or user operation requested. The rationale behind this is to bind (mentally) the SSC solved with the action the user wants and expects to execute. In the absence of such a binding, the user may not consider the importance of solving the SSC. User complacency in this regard will reduce the value of the SSC subsystem mechanism as a whole.

4 Comparison with other Approaches

Passwords: The alternative of having users remember a password for logging in with minimal privilege and using that for day to day business and an alternative password to that gives them system level privilege may be the right thing to do from the security standpoint, but has the drawback of too much cognitive burden. In addition, once the password is compromised, it can be used for all kinds of automated exploits.

User switching: Consciously switching users from normal to superuser requires foresight on part of the user. Often the user does not know in advance that the

program he or she is about to execute requires system level privileges. This is often found out by the user at a later stage, the user then has to rerun the program to get it to execute in the correct context.

Trusted path: Proving that human intervention is involved in a specific process could also be done by using trusted path for input and output (e.g. CTRL-ALT-DEL). All programs possibly interacting with the user interface would have to be disabled for that period of time. Therefore, using a SSC has the advantage of being less intrusive. It also has the additional feature of requiring more human interaction than just clicking O.K. in a window, possibly leading to greater reflection of the person doing the SSC about what he/she is authorizing.

SSCs can also be used to lower the bar on trusted path. If the trusted path only provides secure output (i.e. it can be made sure that no other application can overlap or grab the content of the SSC window), but not secure input, the randomization of the required input ensures that the user input is useless for all other applications that may divert the input.[4]

Code origin: This is the approach taken by e.g. Internet Explorer that authorises code depending on where code that is to be executed comes from (e.g. which security zone). The drawback with this solution is that it is too easy to convince the computer that the code comes from a trusted zone (e.g. cross site scripting attacks or saving the code first on local disk).

5 Conclusion

The proposed ceremonies involving system security CAPTCHAS seems to be less secure than passwords, user switching, and secure path, but more useable. Code origin seems more usable but has been shown to be brittle. The SSC ceremonies try to explore further the possibilities of making automatic exploits more difficult while keeping the user in control. It tries to give the user a second chance with the first immutable law of computer security "once the bad guy has convinced the user to run his code, game over [7].

References

1. Carl Ellison. UPnP Security Ceremonies Design Document. October 2003. www.upnp.org/download/standardizeddcps/UPnPSecurityCeremonies_1_0secure.pdf
2. Turing, A.M. Computing machinery and intelligence. Mind 59, 236 (1950), 433-460
3. Ahn, L. von, Blum, M., Hopper, N.J., and Langford, J. CAPTCHA: Telling humans and computers apart. In *Advances in Cryptology, Eurocrypt 03*. Lecture Notes in Computer Science, Vol. 2656, (2003), 294-311

[4] With secure output the challenge does not even have to be a hard problem, randomization is sufficient.

4. Ahn, L. von, Blum, M., Hopper, N.J., and Langford, J. The CAPTCHA Web page; www.captcha.net
5. Erik Rissanen, Babak Sadighi Firozabadi, Marek Sergot. Towards A Mechanism for Discretionary Overriding of Access Control. In *World Computer Congress*, 2004.
6. Yi-Min Wang, Roussi Roussev, Chad Verbowski, Aaron Johnson, Ming-Wei Wu, Yennun Huang, and Sy-Yen Kuo. Gatekeeper: Monitoring Auto-Start Extensibility Points (ASEPs) for Spyware Management. In *Proc. Usenix LISA*, Nov. 2004
7. Microsoft Corporation. 10 Immutable Laws of Security. Technet www.microsoft.com/technet/archive/community/columns/security/essays/10imlaws.mspx