Windows Security and the MKS Toolkit®

Summary for Management

Windows provides a strong security model that is flexible for the policy implementer and yet fairly transparent to the application programmer. An application cannot circumvent any of the defined policies and is forced to live within the boundaries of the rules laid out by the IT implementer.

In particular

- Policies such as automated local administrator name renaming and password aging policies do not affect the application.
- Domain structures such as departmental boundaries (Domains in NT4 and Active Directory Organizational Units in Windows 2000) might cause applications to fail to run for particular users who do not have appropriate permissions, and cannot be circumvented by applications even when such boundaries are inconvenient to the programmer.

Windows security has become easier to administer with the advent of Active Directory, but the basic security model has remained unchanged since Windows NT was first released in the early 90's. The model is one of machines, domains and trusts (and now with Active Directory more complex trust relationships and delegation).

The Win32 API, upon which applications are written, provides a library of security related capabilities, which can be tapped by applications. An application trying to violate the defined policies will see failures returned from these APIs. As long as the user using the application is following the rules, the applications will work. Any attempt to circumvent the rules will result both in a failure of that application to run and an auditable event that can later be tracked by administrators.

The MKS Toolkit commands and utilities are a suite of Win32 applications. Many of these applications are designed to use the Administration APIs (which have built-in security protection) within the Win32 framework. As such it is understandable that you will question if having access to these tools will in some way make your environment less secure or allow employees to bypass the procedures you have worked so hard to implement. In no way is this possible. If your environment is as secure as you think it is, these tools will fail to run for users not holding the right permissions and if you are auditing for such failures the attempts will show up in the Windows Event Log.

For example, `JoeUser` decides to use the MKS Toolkit secure shell client installed on his desktop to try to gain access to the local file server because he wishes to see the performance review of `PaulUser` who has recently annoyed him. He types "**`ssh server`**" and the secure shell service on the server prompts him for his password to log in. He types it in and is denied logon because his account is not permitted to log on to that server by the policies laid down by the administrator. Better still that administrator had set up auditing to show logon failures and is able to talk to JoeUser's manager about the attempt.

Now the MKS Toolkit commands and utilities and MKS AlertCentre can help the administrator monitor the event log for such events and alert them in such a manner that they do not have to perform manual searches. These tools work because the Administrator using them has the right to view the event log, not because any tool bypasses Windows security.

The MKS Toolkit products give you the power of UNIX style scripting on Windows. As such you must be cognizant of file level permissions on these scripts (i.e., permit write access only to Administrators) to ensure that non-privileged users do not modify these scripts to perform actions the script designer did not intend. However, even if a non-privileged user were to execute these scripts, nothing bad would occur as Windows has no concept of "`setuid`" and all the atomic parts of the script would fail with "permission denied".

Introduction

Security is a big question in global enterprises today and how applications interact or integrate with existing security policies can be a key factor for implementing one solution over another. This document will show that the MKS Toolkit product line is Security Policy neutral. It does not add additional security nor does it compromise existing policies.

In fact, MKS products will only serve to strengthen any security policies you have in place by applying automated testing, monitoring, alerting and corrective action. Your administrators cannot be omni-present, so you need a way to gain peace of mind that your stated policies are in fact implemented satisfactorily. With the MKS Toolkit product line and MKS AlertCentre you can dramatically increase the likelihood that these repetitive and otherwise error prone tasks are occurring on schedule and that you will be proactively notified of issues such as intrusion attempts and willful policy violation by non-privileged users.

MKS Toolkit commands and utilities can help automate many of the security related policies you may have in place already. For example, if you manually rotate your local Administrator password and the name of the account on a regular basis you might consider creating a KornShell, C Shell, or Perl script to automate it and firing that script on a predefined schedule with **`cron`** or **`wts`**, two of the scheduling utilities within the product.

Rather than visiting a workstation to resolve a problem, you may wish to connect using secure shell to install software or repair a problem. While you cannot gain access to the graphical desktop with such a tool, you can fix many common problems such as account lockout, registry mis-configurations, etc. Again, the MKS commands and utilities will work within the stated boundaries of your existing security policies; crossing the boundaries will need an account with higher privilege.

MKS Toolkit commands and utilities offer the same capabilities as can be found in the built-in Windows GUI tools, but with the following advantages:

1. The command line interface is more familiar and natural for UNIX administrators finding their way on Windows for the first time, increasing their productivity and reducing their learning curve; and
2. The scripting capabilities of the command line interface can be used to automate repetitive tasks, thereby improving administrator productivity, and dramatically reducing the time, tedium and error prone nature of performing these tasks on single machines or even across multiple machine boundaries.

Workstations and Stand-alone Servers

Each machine, be it a workstation, an NT4 Domain Controller, or a server that is not acting to control a domain (a stand-alone server), has a Security Access Manager (SAM) account database and one or more authentication packages accessed through the Local Security Authority (LSA). The authentication packages hand out tokens to be used by processes to access protected resources (such as files). A process can ask for a token using the Win32 API **`LogonUser()`** and use this token to access any resource that is accessible to that user as defined by the implementation of the security policy.

Log on credentials do not have to take the form of Username/Password pairs. There are other forms of authentication such as smart cards, optical scanners, certificates, and so on, as well as technologies we have not even considered today. Each of these technologies needs a new Authentication Package (a trusted library which together with the local SAM database is used to produce logon tokens). None of the MKS Toolkit products install an Authentication Package.

So when you log on to the console of a workstation, you enter a username and password and the logon engine creates a desktop for you to work on, impersonates you and launches an explorer. All children of the explorer inherit your identity and so the rights to access protected resources. You are “logged on” to a workstation.

The truth is that having the OS grant you access to the desktop on the console of that machine is just one way to log in – but console logon is the most common. If the local security policy disallows a user to log on then Ctrl-Alt-Del on the console or using **`LogonUser()`** will both fail because Win32 does not allow it.

Domains

The SAM (NT 4 terminology) or the Active Directory (2K and later) on a server (called a domain controller) describe a security boundary within which users have a single sign-on and as such can transcend machine boundaries within that domain such that a domain user on one machine may access, over the network, a resource on another machine within that or one of its trusted domains. This is accomplished using a “trust”. A machine is “joined” to a single domain and as such machine1 can send credentials to machine2 without the user having to re-enter username and password.

In fact the whole single sign-on technology is really quite interesting. When a user needs to log on to a domain account on a workstation, they enter the username, password and domain. The domain controller for the specified domain is contacted and a challenge occurs (like a public/private key pair exchange) such that the LSA on the local machine can be told that the password is valid without ever sending a password over the wire to the domain controller. When access is requested from a network resource, there is a similar logon on the target machine using the “cached credentials” from the interactive logon session (rather than a cached password) sent to that server over the wire. The default-cached credentials can be overridden for a network link from machine to machine, but no logon session can access another machine with multiple credentials.

When a machine is joined to a domain, the “Domain Administrators” is added to the local Administrators group on the machine SAM database such that any member of that group now has permission to administer the machine. It is possible to remove this membership and so deny a domain administrator rights on a machine, but is not common to see this done in practice.

So within the boundaries of a domain (or perhaps Organizational Unit, or OU, is the preferred term when discussing Active Directory) there is administrative authority, users and rights to access resources. All of these are shared by all members of the domain, users and machines alike.

Trusts

Just as there is a trust relationship between an individual machine and the domain that it belongs to, domains can trust each other. In the old NT4 days there were only one and two way trusts. A trusts B and/or B trusts A. With Active directory this trusting became significantly more complex while reducing the administration effort on a well designed system.

But even with Active Directory, none of the facts stated about machines and Domains are affected in any way and in fact applications written in the NT4 model continue to function without change in the Active Directory model. When trusts exist an account from a foreign domain may be used (where permitted by local policy) to log on to a machine, then all the rest of the rules remain unchanged.

Access Control

All resources can be protected with the use of an Access Control List (ACL). This list contains Access Control Entries (ACE) each of which specify if a user or group is granted or denied various accesses, such as read and write, to a resource. When a process attempts to access a resource, there is an ACL on the token that is matched to the ACL on the resource and access is either denied or granted.

Applications

So in fact the domain model in use is not relevant to most applications. They don’t care if the machine is NT4, Windows 2000, Windows XP or Windows Server 2003, a member of a domain or not or even if there are trusts between domains. Applications have the same set of rules governing them. When they try to access a protected resource, the process token is used by the Operating System to decide if the process may access the resource and is granted that access or not granted it. This decision is made by the Operating System, based on the access permissions set on resources by the administrator, not by the application.

Impersonation

There are times when an application designer knows that access to resources may require "becoming another user" for a period of time. For example, a database server application must run as a privileged user in order to read and write the actual database files, but must impersonate a client in order to access tables and other resources within the database. The impersonation allows the database server to rely on the Operating System to limit resource access and so free itself to focus on what it does best. At first glance you may be tempted to say "aha – I knew there were ways around this security stuff" but indeed all forms of impersonation are both required and strictly managed. Here are the mechanisms for impersonation:

- If you have a username and password you can call the Win32 APIs **`LogonUser()`** and **`ImpersonateLoggedOnUser()`**. At that point the "thread token" is adjusted to override the process token and that thread of execution may now access resources of the user specified. Note this requires a password (or one of a well defined list of credentials – but only password authentication is used by the various components of the MKS Toolkit products). There is no way around the need for such a credential.
- If you connect to another machine, that "network logon" has a form of impersonation. You may connect to a machine as another user if and only if you provide a username and password. After successful authentication, you may access resources on that machine as if you were the specified user. Note again that this requires a password.
- There are a series of **`ImpersonateXXX()`** APIs which allow a server application to impersonate the connected user. These impersonations allow you to become another user without the need for a password, but the application itself controls whether to allow the impersonation. This is accomplished by setting the Security Quality of Service (SQOS) parameters in Win32 APIs such as **`CreateFile()`** or on the binding strings for RPC clients. The following is a list of the SQOS options:
    - None – cannot even determine who is at the other end.
    - Identification – the server may learn the identity of the client
    - Impersonation – the server may call **`ImpersonateNamedPipeClient()`** and operate on the client's behalf on the server's machine
    - Delegation – the server application may become the client user for the purpose of access to local and network resources. The server is indistinguishable from the actual logged on user.
- The following is a list of Win32 APIs which permit impersonation on the server side of a connection.
    - **`ImpersonateNamedPipeClient()`** –When a client application (who's process token represents a logon) connects through a named pipe to a server application, part of the connection process is to tell Windows how the server may impersonate the client. .
    - **`RPCImpersonateClient()`** – similar to named pipes, and has the same set of rules, put over a DCE RPC connection.
    - **`ImpersonateDDEClient()`** – impersonation over a Dynamic Data Exchange session. Pretty much an obsolete transport.
    - **`ImpersonateSecurityContext()`** – a generalized from of the above three impersonation mechanisms such that the client can build a data blob to be sent to a server and hands it over an arbitrary transport and the server reassembles the data blob and may impersonate the client. E.g. for use over TCP sockets or any arbitrary transport.
    - **`CoImpersonateClient()`** – a COM API wrapper around **`ImpersonateRPCClient()`**.

LSA secrets

Unlike UNIX, Windows provides no way for the Administrator (root) to become another user without a recognized credential such as a password. Windows Task Scheduler, services, and so on, all need to run as specific users. In UNIX you would mark the executable as "`setuid`" to the account you want to become. When it is run, it is run in the security context of that user. This of course opens up a number of interesting security discussions none of which is relevant to the Windows discussion where such capabilities are not present, not even when MKS Toolkit is present.

So a service needs to start as `domain\joeuser` and it has to start when joeuser is not logged in at system startup. And for the service control manager to become joeuser it needs a password. Seems like an impossible problem. So Windows designers created a way to cache passwords in the LSA database (the same place that the local user database, SAM, resides in the registry under `HKEY_LOCAL_MACHINE/Security`). Now only the system account and the authentication package have access to that hive of the registry (which is stored on disk in `$WINDIR/System32/Config/Security`).

LSA secrets may be machine specific or accessible over the network. In either case, only the person creating the secret or member of the local Administrators group may access the secret. But this means that if joeuser stores his password as an LSA secret that sally administrator can read it if she knows the path to the secret.

LSA secrets are stored encrypted in the LSA database but there is a Win32 API to set and retrieve them: **`LsaRetrievePrivateData()`** and **`LsaStorePrivateData()`**.

Covert Channels

A smart developer (or in fact a naive one who creates the situation inadvertently) can easily create a covert channel to "bypass" Windows Security. It is not exactly bypassing as at some point the system of programs needs to have Administrative access to a machine. Consider the following system:

- An installation program that installs a service to run either as local system, or prompts the unsuspecting Administrator installer to give a domain password.
- This service acts as an RPC server.
- RPC clients are set up from non-privileged users.
- An RPC client connects to an RPC server and this server does not impersonate (either by design or forgetfulness) the client and now the client is executing privileged code.

Now this kind of thing happens all the time when a non-privileged client makes a call into the Operating System that has access to all system resources (like in the database server in the example at the beginning of the Impersonation section above), it is not necessarily a bad thing. It is up to the RPC service or the Operating System to either impersonate, or simply not access protected resources while in the context of a privileged user. It is safer to impersonate and let the Operating System refuse the calls; but it is not required. The server application may do its own resource protection.

MKS Toolkit

So finally, what is it that the MKS Toolkit commands and utilities do and why is it secure?

Generally

There is nothing an application can do to bypass Windows security. All of the logon technology requires that the user present a username and password before any of it works. Nothing in the MKS Toolkit products or in MKS AlertCentre uses any of the **`ImpersonateXXX`** APIs on the server side. In order to "become another user" these tools use **`LogonUser()`** and pass a username and password. The MKS Toolkit commands and utilities and MKS AlertCentre do store LSA Secrets. Storing LSA Secrets is a common practice on Windows.

Specifically

- **`sshd`**. In normal use, **`ssh`** takes a username on the client command line and the server requests the password over an encrypted TCP link. So the password is never sent clear text over the network and is therefore not insecure. It is possible to configure **`sshd`** to "assume that the client is trusted" when a user's public key is on the ssh server and their private key is on the client. If this key exchange succeeds, and if a password for that user has been cached as an LSA secret, then that user may log in without a password (because the password was given to **`sshd`** through **`rsetup`**).
- **`rsetup`** is a tool which takes a password and squirrels it way in the LSA database. This LSA secret is later used by the **`sshd/rshd/rexecd`** to permit a password-less logon. So the big red bells are going off. I can hear them from here. It is important to note that none of this technology can work unless you:
  - Log on to the machine
  - Run **`rsetup`**
  - Provide a password
- **telnetd**. Telnet is not encrypted and passwords are sent using clear text across the network. The username and password are required to log in. There is no password-less login technology in telnet – but the passwords are there for any network sniffer to see. Telnet is provided for UNIX compatibility but ssh is by far the recommended technology.
- **rshd/rexecd**. These technologies are by definition password-less. They require that **rsetup** be run and **`$ROOTDIR/etc/hosts.equiv`** and **$HOME/.rhosts** determine who may become a user. These technologies are provided for UNIX compatibility, but **`ssh`** is by far the recommended technology for these operations.
- **`-S`** options on various tools. Most Win32 administration style APIs provide the option to specify a server name when opening the resource. These connections are in fact done to a standard Windows PRC server which in turn calls **`ImpersonateRPCClient()`** and acts on that machine as if it were the calling user. As long as the calling user has permission on the remote machine, the operation will succeed. It should be noted that trying to cross an organizational (domain) boundary with such a tool where the user is unknown or underprivileged will result in the tool failing to perform any action. Windows security is not compromised in any way. Examples of such tools
  - **`registry`** – view or modify the local or remote registry – assuming you have permission to do this.
  - **`service`** – start or stop services on the local or remote machine.
  - **`mkshare`** – make a share on the local or remote machine – assuming you have permission to do this.
  - **`lsshare`** – list the shares on a local or remote machine – assuming you have permission to do this.
  - **`rmshare`** – delete a share on a local or remote machine – assuming you have permission to do this.
- Miscellaneous security related tools
  - **`su`** – become another user. This requires
    - The calling user to have a series of privileges not usually assigned to a mortal user
    - The username and password of another user.
  - **`priv`** – will adjust the token privileges of an application or in fact grant a user privilege permanently (such as log on as a service, or change the system time). It is not possible to give yourself privileges you do not own (but you can activate dormant privileges) unless you are an administrator. So mortal users will see permission denied messages when trying to use this tool.
  - **`lsacl`** – like the Windows explorer, this is a command line tool to view Access Control Lists on Windows resources such as shares, files, registry keys, and so on, assuming of course you have the permission to do this.

    - **`chacl`** – like Windows explorer, this command allows you to change Access Control Lists on Windows resources, assuming of course you have the permission to do this.
- MKS AlertCentre.
    - MKS AlertCentre uses Windows Task Scheduler to store logon username/password pairs. This is not controlled by MKS AlertCentre but rather by Windows itself.
    - MKS AlertCentre uses globally (network) accessible LSA secrets to store passwords various monitors (e.g. the password for database access in an ODBC or ADO monitor). The global need is for the backup monitoring station that needs to have access to all the username/password pairs from the primary monitoring station. Global LSA secrets can only be accessed by the creator and an administrator on the machine storing the secret but are available by such users over the network, so as long as encrypted sessions are used for this retrieval, there is no chance of an inadvertent interception.

No covert channels exist in the MKS Toolkit products, MKS AlertCentre, or N*u*TCRACKER Platform applications.

The N*u*TCRACKER Platform (which runs applications ported with MKS Toolkit for Enterprise Developers), installed with all versions of MKS Toolkit and MKS AlertCentre, installs a service which is an RPC server and runs as local system and does not impersonate the calling client. It does not act on any system resources and does not return privileged information. The RPCs are really helper functions to ensure that the global shared memory used to preserve state is kept in a sane state. No client knowing the RPC protocol could gain access to information they are not supposed to be able to access. Any attempt to duplicate any handles stored in the service would not be permitted due to permissions on the objects themselves. All other services either log in using **`LogonUser()`**, or do not access protected resources.

Services include:

- N*u*TCRACKER service
- **`telnetd`**
- **`sshd`**
- **`rshd`**
- **`rexecd`**
- Simple Web Server requires logon where web pages are protected by ACLs on disk.
- **`snmptrapd`** – just logs an event in the event log unless **`$ROOTDIR/etc/snmptrapd.conf`** is set up to run a program. This file is protected with an Administrator ACE so the casual user cannot gain access to the system account.

What about the need for password-less logon from a script? There are a number of ways to accomplish this. Either schedule the script to run from **`cron`** or Windows Task Scheduler (both of which will use cached LSA secrets to do the logon on the local machine), after ensuring that the script is not writable by anyone outside the Administrators group, or use public/private key pars and Secure Shell password-less logon using **`rsetup`** to cache the password in the LSA database. But remember that many of the Windows administrative tools included in the MKS Toolkit products are "remote-able" and so you often do not need to run a script on the machine itself, but rather run the script against the machine from a secured server using the individual "**`-S`**" options on the tools themselves.

Caveats

MKS Toolkit products contain a powerful set of tools. They give the mortal user access to low level Security APIs in a fairly simple way. The MKS Toolkit products are security policy neutral. How you implement your security policy is up to you. If you prefer to give everyone Administrative access to their workstations, then every user has the ability to modify shell logon scripts and so potentially set a trap to gain access to Domain Administrator resources when the MKS Toolkit commands and utilities are run by a Domain Administrator logged into either the console or through a remote access session such as Secure Shell. By the way, this is true of **`regedit.exe`**, Windows Explorer, **`net.exe`**, and many other built-in Windows tools too. As with any security policy, you are only as secure as your weakest link. It is beyond the scope of this document to define all of the “best practices”, but here are a few guidelines:

- Do not add all users to Administrators group on the local machine.
- Have a password aging policy combined with account lockout. And monitor your event logs with a tool such as MKS AlertCentre.
- Rotate machine administrator passwords and even rename the administrator account on a regular basis. This can be automated with the commands and utilities included in the MKS Toolkit products from a single administrator workstation – it does not require visits from an administrator.
- Do not put passwords in scripts.
- Ensure that any administrative scripts that you do have are protected such that only the local Administrators have read or execute permissions. Better still, store these on a domain server with domain administrator read/execute permissions on both the files and shares. Even better still apply auditing to the files and monitor for audit failures using a tool like MKS AlertCentre.
- Ensure that Registry keys are properly protected as you would with files.
- Perform routine security audits on your machines to test your policies. Use the scripting tools included in MKS Toolkit products to attempt to gain access to resources as a mortal user and then ensure that your monitoring tools tell you what has occurred.
- Do not install MKS Toolkit on a FAT (FAT, FAT16, FAT32) file system.
- Do use Windows 2000 delegation and organizational unit domain structures.
- Remember that Secure Shell is preferred over r-tools. If you wish to use r-tools, consider putting an ACL on **`rsetup.exe`** that limits its use to Administrators.

Nothing will replace a good set of policies and a proper testing and monitoring plan to ensure that these policies are indeed in effect.

The MKS Toolkit product line and MKS AlertCentre are wonderful platforms to help you with this testing and monitoring to ensure policy compliance by systematically simulating various types of attacks on all machines in all domains or monitoring of security logs to detect attempted intrusions.

Scripts need to be protected such that mortal users cannot corrupt them. Executing scripts containing access to secure resources is not a problem as the operating system will block non-privileged users, but if such a user were to modify an existing script, then they could lay a trap for an unsuspecting administrator when this script is run this higher permission set.

Imagine that you have a server that is locked in a server cabinet. You assume that this server is secure so you do not bother to manage the list of users with privilege to log on locally. Now you install an MKS Toolkit product and with it is a set of remote access logon services (**`sshd`**, **`telnetd`**, and so on) and you forget that a mortal user is allowed to log on. Now any user can log into that server. In and of itself this is probably not all that bad provided resources are properly protected by ACL. Such a login would not allow the user access to anything. But if you only use share level permissions on the file systems, this kind of access could result in unwanted accesses to resources. The same issue exists, by the way, even if an MKS Toolkit product is not installed, as there are Windows tools and utilities that similarly expose these resources.

So, ensure that “Logon Locally” privileges are properly set and that you use resource level ACLs to control access to files, registry, etc. Implement auditing and monitor for audit failures (all part of a well designed security policy) and the MKS Toolkit products will not allow you to make your environment any less secure than it would be if it were not installed. MKS AlertCentre will help you to administer and monitor your environment to ensure its continued adherence to well defined policies.

Conclusion

As with any tool, MKS Toolkit commands and utilities must be used wisely. In general there are no security issues that should concern you if you have a well designed and well implemented security policy. The **`rexecd`** and **`rshd`** tools are the only two services that leave the door open for unwanted accesses to resources and cannot be used if **`rsetup`** is limited to knowledgeable employees. In all other cases, the calling user will bump up against the Windows security (in any domain model) if that user does not have access to that resource on that machine.

Overall the commands and utilities included in all MKS Toolkit products along with MKS AlertCentre are great additions to your tools suite for day to day administration, policy testing, monitoring for attacks and scripting bulk changes and can only enhance a strong security policy.

About MKS Toolkit

There are several products in the MKS Toolkit product line aimed at different kinds of people, performing different kinds of tasks. All MKS Toolkit products are unified by our goal of making your use of Windows more efficient and more enjoyable. The products fall into two broad categories—those for system administrators and those for software developers.

There are three main components included in the MKS Toolkit product line, and each of the individual products may contain some or all of each group. This document primarily refers to the commands and utilities component; a subset of which is included in all MKS Toolkit products.

MKS Toolkit Commands and Utilities

There are over 400 individual commands and utilities that are offered as part of the MKS Toolkit product line. These range from command shells like the KornShell, C Shell, and **`bash`** to invaluable UNIX commands like **`grep`**, **`make`**, **`awk`**, **`sed`**, **`vi`**, and **`find`**, to system connectivity utilities and services like Secure Shell, **`rsh`**, and **`telnetd`**. MKS Toolkit products also include many Win32 specific commands to deal with Windows specific constructs and issues. This includes commands like **`registry`**, **`shortcut`**, **`winctrl`**, **`desktop`**, and **`domain`**.

MKS Toolkit UNIX APIs

Including more than 2700 APIs, this component of the MKS Toolkit product line allows you to port all manner of UNIX and Linux applications to Windows, while maintaining a single source code baseline.

MKS N*u*TCRACKER Platform

The N*u*TCRACKER Platform is a full UNIX runtime compatibility environment for windows platforms, that provides services to each of the MKS Toolkit products and to any application or script migrated from UNIX or Linux to Windows with either MKS Toolkit for Professional Developers or MKS Toolkit for Enterprise Developers.

Additional Reference

For more information on the individual MKS Toolkit products, please visit our web site at http://www.mkssoftware.com.

▪ MKS Toolkit Products:	http://www.mkssoftware.com/products
▪ MKS Toolkit Reference Pages:	http://www.mkssoftware.com/docs (man pages)
▪ MKS Toolkit Command List:	http://www.mkssoftware.com/products/tk/commands.asp