Interested in learning more about security?

SANS Institute InfoSec Reading Room

Sudo for Windows (sudowin)

Sudo was developed in reaction to the standard UNIX security model where although some granularity is possible with group and file permissions, delegating security is largely all or nothing. If a user was designated an administrator this usually meant giving them access to the root account's password. The problem with this model was that it provided no accountability for actions taken on the system since all actions were being executed under the auspices of one user account. In summary, Sudo provides delegation and acc...

Sudo for Windows (sudowin)

GCWN Gold Certification

Author: Schley Andrew Kutz, a.kutz@its.utexas.edu

Adviser: Jim Purcell

Accepted: January 20, 2007

Outline

1. Abstract

The original Sudo application was designed by Bob Coggeshall and Cliff Spencer in 1980 within the halls of the Department of Computer Science at SUNY/Buffalo.[1] For twenty-six years, Sudo has provided the foundation of secure computing on UNIX and Linux platforms by allowing systems administrators to delegate privileged commands to trusted users and audit their use. A trusted user can execute a privileged command in their own user context by reaffirming their identity through confirming their passphrase and this execution will then be recorded in an auditable log. Sudo encourages the principal of least privilege – that is, a user operates with a bare minimum number of privileges on a system until the user requests a higher level of privilege in order to accomplish some task.

Sudo was developed in reaction to the standard UNIX security model where although some granularity is possible with group and file permissions, delegating security is largely all or nothing. If a user was designated an administrator this usually meant giving them access to the root account's password. The problem with this model was that it provided no accountability for actions taken on the system since all actions were being executed under the auspices of one user account. In summary, Sudo provides delegation and accountability.

The current versions of Microsoft Windows lack equivalent functionality to that which Sudo provides. Therefore the security model in Windows is described by delegating a fixed privilege level

[1] http://www.gratisoft.us/sudo/history.html

to distinct user accounts, but this model encourages, and in fact requires, separate user accounts for any user who logs into a computer system as a non-privileged and privileged user account. This security model is a hassle and to circumvent the frustrations associated with multiple user accounts, privileged users inevitably begin to log into a computer system with their privileged user accounts only. This behavior is very unsecure because it results in applications running with unnecessarily high levels of privilege. Therefore the security model that Microsoft Windows encourages is flawed.

Sudo for Windows, or sudowin, was developed in the summer of 2005 in order to provide Sudo functionality to the Microsoft Windows operating system. This practical will review the history of sudowin, its implications and use in practical application, how it is designed, and most importantly, how everyday users can obtain, install, and begin using sudowin.

2. Document Conventions

When you read this practical assignment, you will see that certain words are represented in different fonts and typefaces. The types of words that are represented this way include the following:

command	Operating system commands are represented in this font style. This style indicates a command that is entered at a command prompt or shell.
filename	Filenames, paths, and directory names are represented in this style.
URL	Web URL's are shown in this style.

3. Introduction / Executive Summary

Microsoft Windows has a broken security model that implicitly encourages users to log onto computer systems with administrative privileges at all times. Users are forced to choose between being hindered by the frustrations of a normal user account or being exposed to the viruses and Windows exploits of the world by logging in as an administrative account. This is a choice between bad and worse.

Other operating systems, such as UNIX/Linux and Apple OS X, solve this problem by employing privilege escalation. This privilege escalation is founded upon a command known as sudo. The sudo command allows a user to escalate her privileges when needed by entering her own passphrase. Not only that, but the user also retains her original user context. Microsoft Windows lacks this functionality.

Many mistakenly believe the Windows RunAs command provides Windows with sudo functionality. The RunAs command does not enable a user to escalate her privileges. It allows the user to assume the identity of another account, if the user knows the passphrase for that account. For this reason the RunAs should be thought of as an equivalent to the UNIX/Linux su command, not sudo. Hence, Windows remains without sudo functionality.

Sudo for Windows (sudowin) attempts to fix the Microsoft Windows security model by providing Sudo functionality to the Windows operating system. This practical will explore the history of sudowin, its implications on the world of Windows security, the complex design of this project, and finally, how end-users may

implement sudowin.

4. History

Sudowin is an invention of necessity. In the summer of 2005 Andrew Kutz decided to stop logging into his Windows XP SP2 workstation as an administrative user. This caused an immediate problem. Andrew used a software package called DVD Decrypter to back up his DVDs, but DVD Decrypter would not run as a non-administrator because of the level of access it required to the DVD-ROM drive. The simple solution was to simply use the RunAs command to launch DVD Decrypter as an administrator so that it had the privileges that it needed, except this would not work because of Andrew's permissions scheme.

Andrew did not set explicit permissions inside of his home directory, therefore Andrew's user account, `akutz`, did not own the directories or files within his home directory, the special user `CREATOR OWNER` did. This meant that the user account that created a file or directory inside of Andrew's home directory would own it. If Andrew had launched DVD Decrypter as another user, the administrator, and backed up his DVDs to ISO images inside of his home directory, the other user would own the files, not Andrew's user account. There were two obvious solutions:

1. Change the permissions scheme so that Andrew's user account was the explicit owner of all the directories and files inside of his home directory.

2. Develop sudo functionality for Windows so that DVD Decrypter, and other applications like it that require elevated privileges, could be launched with temporary elevated privileges while retaining the original user context.

Always the road less traveled.

5. Implications

There are other tools available that provide similar functionality, but they are all lacking in some respect.[2345] These tools are not configurable, they are not extensible or worse yet, they actually create security holes because they are subject to man-in-the-middle attacks.

Sudo for Windows is configurable out of the box. There is hardly a setting that cannot be changed by simply editing a text file. And because of its plugin architecture, anyone with some programming experience can develop custom authentication, authorization, and credentials caching plugins, extending its capabilities even further.

Most importantly, sudowin does not decrease security by creating man-in-the-middle-attacks for malicious users and disgruntled administrators to exploit. Sudo for Windows increases overall security by enabling your entire enterprise to run in Least User Access (LUA) mode.

The Windows desktop environment would benefit greatly from Sudo for Windows. Windows could ship with the Administrator account disabled and the first user a member of the Sudoers group (much like Ubuntu Linux). Whenever a user needed to make a system change she could simply escalate her own privileges to do so instead of running

[2] suDown - http://sudown.sourceforge.net/
[3] MakeMeAdmin - http://blogs.msdn.com/aaron_margosis/archive/2004/07/24/193721.aspx
[4] WinSudo - http://winsudo.toadlife.net/
[5] SUperior SU - http://www.stefan-kuhr.de/supsu/main.php3

the command with the Administrator account. Sudowin could bring the learned security practices of Linux to the world of Windows.

Sudowin is also ideal for enterprise deployment. An Active Directory administrator who delegates organizational units (OU) management to other administrators is typically assigned two separate accounts – one unprivileged, everyday account, and one privileged account used for system administration. Keeping up with two accounts is a huge pain for administrators and inevitably results in most of them staying logged into their computers as their privileged account, a massive security problem.

With sudowin, OU permissions can be delegated to groups that OU administrators are not normally members of. The OU administrators could use sudowin to launch Active Directory Users and Computers with privileges elevated to the level needed to manage a particular OU.

This is just one example, but since every object in Active Directory has a permission set, sudowin could be used to manage each and every one of those objects. Sudowin will create happier administrators and a more secure Active Directory.

6. Design

Sudowin is a distributed application that has several components. It must be, and has to, because Windows as an operating system is not conducive to sudo-like functionality, and thus a program that provides it is required by necessity to be tricky.

At its core, sudowin is composed of a client and a server. The client asks the server to escalate a user's privileges and execute a given command, and the server either complies or refuses. Obviously this is a great simplification of a complex process.

Sudowin is designed to be completely extensible and therefore has a rich plugin architecture. It currently supports three types of plugins – authentication, authorization, and credentials cache.

Finally, the linchpin of sudowin may very well be the callback application. This is what enables the entire project to actually work. This is sudowin's flux capacitor.

What follows is a review of all the components that make sudowin work. This includes:

- Server
- Client
- Plugins
  - Authentication
  - Authorization
  - CredentialsCache
- CallbackApplication

Server

The sudowin server is hosted inside of a Windows service named

sudowin. This service runs as the local system account. The server exposes a SingleCall object that clients can access so long as the user account invoking the client is a member of the configured Sudoers group. The server also loads the configured plugins in order to authenticate users, authorize their requests, and cache their credentials.

Configuration

The sudowin server's configuration file is a standard .NET application configuration file and must reside in the same directory as the sudowin server binary, Sudowin.Server.exe. Because a .NET configuration file has the same name as its corresponding binary but with an extension of .config, the name of the configuration file is Sudowin.Server.exe.config. The configuration file is really divided into three sections: the application settings, the remoting settings, and the diagnostics settings.

Application Settings

The application settings section should appear towards the top of the configuration file and is contained by the <appSettings></appSettings> node. Values are added to it in the following format:

```
<add key="keyName" value="settingValue" />
```

The following is a list of the application settings that the sudowin server is configured to use:

- **pluginConfigurationUri** This is the Uniform Resource Indicator (URI) of the file that stores the server's plugin configuration. This URI can point to a local file or one accessible via HTTP. The plugin configuration file must adhere to the plugin configuration schema as defined in PluginConfigurationSchema.xsd.

- **pluginConfigurationSchemaUri** This is the URI to the schema file the server uses to validate the plugin configuration file with. This URI can point to a local file or one accessible via the Hyper Text Transfer Protocol (HTTP). This URI must point to a file that adheres to the plugin configuration schema as defined in PluginConfigurationSchema.xsd.

- **callbackApplicationPath** This is the fully qualified path to the sudowin callback application. This is used by the sudowin server to launch processes in the context of the user that invoked the sudowin client.

Remoting Settings

The remoting settings section appears directly after the application settings section and is contained by the following node <system.runtime.remoting></system.runtime.remoting>. Although there are a few things going on in this section, there are two sections where someone could want to potentially configure things differently than is default:

1. The name of the sudowin service remoting object.
2. The remoting channel.

Remoting Object

The sudowin server remoting object is defined in the configuration file by the following stanza:

```
<service>
  <wellknown
    mode="SingleCall"
    type="Sudowin.Server.SudoServer, Sudowin.Server"
    objectUri="sudowinserver.rem"
  />
</service>
```

- **wellknown** The sudowin server is configured as a WellKnown remoting object. This means that this is a server activated object (SAO), as opposed to a client activated object (CAO). The difference is that the server controls the lifetime of the object as opposed to the client. To change this to a CAO, change the name of the node from wellknown to activated.

- **mode** The sudowin server object is configured as a SingleCall object. This means that a new sudowin server is instantiated for every remote call made to it. The alternative is to configure the server as a Singleton object. In this case the server is instantiated once and that object handles all remote calls. Changing this setting is not recommended. The sudowin server is configured as a SingleCall object because it does not have a high penalty upon creation, an instance where you would want to use a Singleton, and a Singleton synchronizes access to itself meaning that two users logged into the same computer would be prevented from invoking sudowin at the same time.

- **objectUri** This is the URI of the remoted sudowin server object. If you change this value then you will need to change the URI that clients attempt to connect to.

Remoting Channel

The sudowin remoting channel is an inter-process channel (IPC) that the server exposes to allow clients to establish a communications link to the remoted objects registered on this channel, in this case the sudowin server object. The channel is defined by the following stanza:

```
<channel
  type="System.Runtime.Remoting.Channels.Ipc.IpcServerChannel,
        System.Runtime.Remoting, Version=2.0.0.0,
        Culture=neutral,PublicKeyToken=b77a5c561934e089"
  portName="sudowin"
  secure="True"
  authorizedGroup="Sudoers">
  ...
</channel>
```

- **portName** This is the name of the port that the server listens for incoming communications on. If this value is changed then the port name the clients are configured to talk to will need to changed as well.

- **secure** By settings this value to `True` communications between the sudowin server and any clients are secured..

- **authorizedGroup** The security context in which the client is run must be a member of this group or else the client will be refused when the client attempts to establish a communications channel with the server channel. By default this is set to `Sudoers` and the `Sudoers` group is created when sudowin is installed.

Diagnostics Settings

The diagnostics section is contained by the <system.diagnostics></system.diagnostics> node. It is defined by the following stanza:

```
<system.diagnostics>
  <trace autoflush="true" />
  <sources>
    <source name="traceSrc" switchValue="ActivityTracing, Verbose">
      <listeners>
        <add
          initializeData="r:\projects\sudowin\trunk\sudowin\
                          server\bin\Debug\service.log"
          type="System.Diagnostics.DelimitedListTraceListener"
          name="traceListener"
          traceOutputOptions="DateTime, ProcessId, ThreadId, Timestamp"
        />
      </listeners>
    </source>
  </sources>
</system.diagnostics>
```

- **autoflush** Setting this value to true means that all trace data is immediately written to the trace listeners, be they files or event logs. Settings this value to false will result in said data being queued until it is manually flushed or the sudowin service is shut down.

- **switchValue** Upon installation this value is set to Error instead of ActivityTracing, Verbose. This means that only trace data classified as Error or greater, for example, Critical, is actually written to the trace listeners.

Tracing is not of much use without a place to send its data, hence the trace listener. To add a trace listener simply add the correct stanza in the listeners section. For example, in the following example a System.Diagnostics.DelimitedListTraceListener has been added:

```
<add
  initializeData="r:\projects\sudowin\trunk\sudowin\
                  server\bin\Debug\service.log"
  type="System.Diagnostics.DelimitedListTraceListener"
  name="traceListener"
  traceOutputOptions="DateTime, ProcessId, ThreadId, Timestamp"
/>
```

- **initializeData** This is the fully qualified path to the file that will have the trace data written to it.
- **traceOutputOptions** These are essentially columns that get written to the trace listener every time data is written. This alleviates the need to manually write the current date or process ID with every log entry – .NET handles it for you.

Client

Sudowin ships with two clients, the command line client and the Graphical User Interface (GUI) client. However, anyone can design a client for sudowin, all is required is to download the source code and follow the examples.

Command Line Client

The sudowin command line client binary is named Sudowin.Clients.Console.exe but is renamed to sudo.exe upon installation. The path to this file is added to the system's PATH environment variable, so from any command line. The only thing a

user must do in order to invoke sudo is to type `sudo`. For example, to open a new command prompt with sudowin, type:

```
sudo cmd
```

Sudowin will prompt the user for a passphrase and launch a new command prompt with elevated privileges, assuming the user is authorized to invoke sudo on the command shell.

This command also accepts a passphrase from the command line. If a user's passphrase is `foobar`, the user could type:

```
sudo –password foobar cmd
```

Sudowin will not prompt the user for a passphrase, but instead use the passphrase specified on the command line and will launch a new command prompt with elevated privileges, assuming the user is authorized to invoke sudo on the command shell.

Configuration

The command line client has a configuration file located side-by-side with its binary. The default path for this is `INSTALLDIR\Clients\Console\sudo.exe`.

There are only two sections that might need to be modified. The first section defines the sudowin server client with the following stanza:

```
<wellknown
  type="Sudowin.Common.ISudoServer, Sudowin.Common"
  url="ipc://sudowin/sudowinserver.rem"
/>
```

- **url** If the port or URI of the sudowin server object is modified then this value needs to be updated as well.

The other section is as follows:

```
<channel
  type="System.Runtime.Remoting.Channels.Ipc.IpcClientChannel,
       System.Runtime.Remoting, Version=2.0.0.0, Culture=neutral,
       PublicKeyToken=b77a5c561934e089"
  portName="sudowin"
  secure="True"
  useDefaultCredentials="True">
  ...
</channel>
```

- **portName** This should be the same as the port name defined for the server channel in the sudowin server configuration file.

GUI Client

Sudowin also ships with a GUI client. The GUI client is enabled by default for the following types:

- .bat
- .cmd
- .exe
- .msi
- A folder

Right clicking on any of these file types will bring up the

default Windows context menu and Sudo… will be one of the options. Clicking Sudo… will bring up a prompt to enter a passphrase (if it is not cached) and then the intended program or folder will be opened with elevated privileges.

Configuration

The GUI client's configuration file is by default located at INSTALLDIR\Clients\GUI\Sudowin.Clients.Gui.exe.config. It follows the same format as the console client's configuration file.

Plugins

What sets sudowin apart from all other projects that attempt to provide sudo functionality to Windows is its extensible design. Almost everything in sudowin is a plugin, from its authorization handlers, to authentication and credential caching.

The sudowin project defines a master interface that all plugins must implement called IPlugin. However, because sudowin is distributed through remoting, the interface alone is not enough. All remoted objects must derive from a class called MarshalRefByObject. Since an interface cannot derive from a class, it stands to reason that all plugins must derive from some class. This class is called Plugin. All plugins should therefore derive from Plugin, which both implements IPlugin and derives from MarshalRefByObject.

Configuration

Plugins for sudowin are configured in the plugin configuration file. This file is by default located at INSTALLDIR\Server\pluginConfiguration.xml. The plugin configuration

file follows a very strict schema. This schema is by default located at INSTALLDIR\Server\PluginConfigurationSchema.xsd. The path to both of these files can be changed in the sudowin server's configuration file – for example, it would be possible to centrally configure sudowin plugins by placing the plugin configuration file on a network share or make it accessible via HTTP.

The following is an example of a plugin configuration file:

```
<?xml version="1.0" encoding="utf-8" ?>
<pluginConfiguration
 xmlns="http://sudowin.sourceforge.net/
 schemas/PluginConfiguration/">
  <plugins>
    <plugin
      pluginType="authenticationPlugin"
      assemblyString="Sudowin.Plugins.Authentication.NT.
                      NTAuthenticationPlugin, Sudowin.Plugins.
                      Authentication.NT"
    />
    <plugin
      pluginType="authorizationPlugin"
      assemblyString="Sudowin.Plugins.Authorization.Xml.
                       XmlAuthorizationPlugin, Sudowin.Plugins.
                       Authorization.Xml"
      dataSourceConnectionString="r:\projects\sudowin\trunk\
        sudowin\plugins.authorization.xml\sudoers.xml"
      dataSourceSchemaUri="r:\projects\sudowin\trunk\sudowin\
                           plugins.authorization.xml\
                           XmlAuthorizationPluginSchema.xsd"
      dataSourceCacheFilePath="r:\projects\sudowin\trunk\sudowin\
                               server\sudoers.xml.cache"
      dataSourceCacheFrequency="00:05"
      dataSourceCacheEnabled="true"
    />
    <plugin
      pluginType="credentialsCachePlugin"
      assemblyString="Sudowin.Plugins.CredentialsCache.LocalServer.
                      LocalServerCredentialsCachePlugin,
                      Sudowin.Plugins.CredentialsCache.LocalServer"
      serverType="Singleton"
    />
  </plugins>
</pluginConfiguration>
```

Instead of explaining the plugin configuration file line by line, the plugin configuration schema will be examined.

Plugin Configuration Schema

The plugin configuration schema is an XML schema definition (XSD). Employing a schema ensures that a plugin configuration file will always be parsed correctly. When it is loaded it is checked against the schema file and is rejected if it does not conform to the schema.

The current version of the plugin configuration schema is available at http://sudowin.svn.sourceforge.net/viewvc/sudowin/trunk/sudowin/Plugins/PluginConfigurationSchema.xsd?view=markup.

A basic outline of the schema looks like this:

```
<?xml version="1.0" encoding="utf-8" ?>
<pluginConfiguration>
  <plugin attribute1="value" attribute2="value" ... />
</pluginConfiguration>
```

Missing from the basic outline above are the attributes for the `plugin` node. What follows is a list of all valid `plugin` node attributes and their definitions:

- **assemblyString**
  - **type** string
  - **use** required
  - **description** This is the fully qualified assembly type.

- **serverType**
  - **type** string
  - **use** optional
  - **default** SingleCall
  - **description** This defines whether a plugin is loaded as a `SingleCall` object or a `Singleton`. If the plugin takes a very long time to load or it needs to persist its state between calls then choose Singleton, otherwise `SingleCall` is a safe bet. The credentials cache plugin that ships with sudowin, `Sudowin.Plugins.CredentialsCache.LocalServer`, must be configured as a Singleton object or it will not cache credentials between sudo invocations.

- **serverLifetime**
  - **type** int
  - **use** optional
  - **default** 0
  - **description** This defines the lifetime of a plugin configured as a `Singleton` before the object is released and a new one is created (this is its lease). This value has no effect on plugins configured as `SingleCall`.

- **enabled**
  - **type** bool
  - **use** optional
  - **default** true
  - **description** You can set this value to `false` to disable the plugin without removing its entry from the plugin configuration file.

- **activationData**
  - **type** string
  - **use** optional
  - **description** If a custom plugin is designed and upon activation requires additional configuration information not specified by the normal plugin configuration parameters, this attribute may be used to pass the plugin its required data.

- **pluginType**
  - **type** string
  - **use** required
  - **description** Valid values are currently `authorizationPlugin`, `authenticationPlugin`, and `credentialsCachePlugin`.

- **dataSourceConnectionString**
  - **type** string
  - **use** optional
  - **description** If a plugin uses a data source this attribute can be used to specify its connection string.

- **dataSourceSchemaUri**
  - **type** string
  - **use** optional
  - **description** If a plugin uses a data source and uses a schema to validate it, this attribute can be used to specify the schema URI.

- **dataSourceCacheFilePath**
  - **type** string
  - **use** optional
  - **description** If a plugin enables data source caching this attribute can be used to specify a fully-qualified file path for the cache file.

- **dataSourceCacheUpdateFrequency**
  - **type** TimeSpan
  - **use** optional
  - **default** 00:05
  - **description** If a plugin enables data source caching this attribute can be used to specify how often the cache should be updated.

- **dataSourceCacheEnabled**
  - **type** bool
  - **use** optional
  - **default** true
  - **description** Set this to `true` to enable data source caching.

- **dataSourceCacheUseAsPrimary**
  - **type** bool
  - **use** optional
  - **default** false
  - **description** Set this to `true` to treat the data source cache as the primary source of data. In this scenario the original source is only contacted for updates at the interval defined by `dataSourceCacheUpdateFrequency`.

- **dataSourceCacheUseStaleCache**
  - **type** bool
  - **use** optional
  - **default** false
  - **description** Set this to `true` to let a plugin know that it is okay to use a data source cache file that has not been updated after the given interval defined by `dataSourceCacheUpdateFrequency`. Enable this if sudowin should remain functional even if the plugin cannot connect to the remote data source.

Plugin Types

There are currently three supported plugin types. These are:

- **authentication** Used to authenticate users credentials.
- **authorization** Used to authorize sudo requests.
- **credentialsCache** Used to cache credentials.

Authentication

Authentication plugins are used to authenticate a user's credentials upon invoking a sudowin client. Currently sudowin ships with one authentication plugin, `Sudowin.Plugins.Authentication.NT`.

However, an authentication plugin could be developed that would authenticate users from an Oracle database, a text file, or even simply always returned `true`. The crux is that the credentials the user supplied must be able to be used to launch a process on the computer where they invoked sudo.

NT

The NT authentication plugin uses the native win32 Application Programming Interface (API) function, LogonUser, to authenticate the user's credentials. This plugin authenticates local and domain accounts.

Authorization

The authorization plugin is what authorizes incoming sudo requests. Sudowin ships with an authorization plugin that uses an XML file as its data source – the `Sudowin.Plugins.Authorization.Xml` plugin.

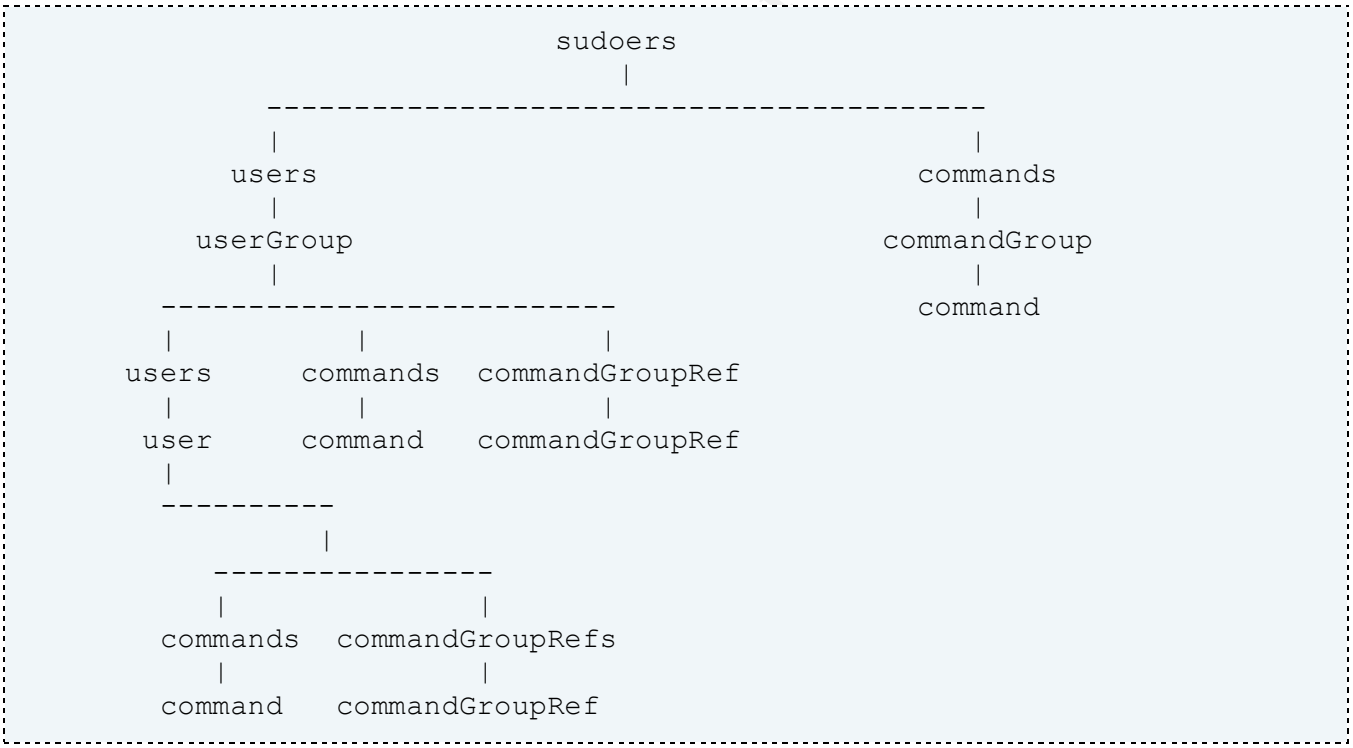
XML

The XML authorization plugin uses an XML file as its data source. This XML file must adhere to the schema file that is provided with the plugin. Both of these values must be configured in the plugin's configuration section in the plugin configuration file. By default the XML authorization plugin is configured as a `SingleCall` object so that it will read the XML file every time it is called – the effect of this is that changes made to the XML file are reflected the next time a sudo invocation occurs. The alternative is to

configure the XML authorization plugin as a Singleton, the effect of which is that the XML file only gets read one time when the sudowin service starts.

The current version of the schema file that the XML authorization plugin must adhere to is available at this link, http://sudowin.svn.sourceforge.net/viewvc/sudowin/trunk/sudowin/Plugins.Authorization.Xml/XmlAuthorizationPluginSchema.xsd?view=markup.

The XML authorization schema appears a little complicated, but it is in fact quite straightforward. Here is an outline of the parent-child node relationship of the schema:



As seen above, the root element of the schema is the <sudoers> node. The <sudoers> node can contain two types of child nodes, <users> and <commands>. The <commands> node can contain <commandGroup> nodes which in turn can contain <command> nodes. The <users> node may

contain one type of child node, <userGroup>. <userGroup> in turn can contain three types of child nodes: <users>, <commands>, and <commandGroupRefs>. The <commands> and <commandGroupRefs> nodes can each contain one child node type: <command> and <commandGroupRef>, respectively. The <users> node can contain one type of child node, <user>. The <user> node can contain two types of child nodes: <commands> and <commandGroupRefs>. As before, the <commands> and <commandGroupRefs> nodes can each contain one child node type: <command> and <commandGroupRef>, respectively.

Here is an example a sudoers.xml file the implements the schema:

```
<?xml version="1.0" encoding="utf-8"?>
<sudoers xmlns="http://sudowin.sourceforge.net/
               schemas/XmlAuthorizationPlugin/"
  privilegesGroup="Administrators"
  invalidLogons="3"
  timesExceededInvalidLogons="3"
  invalidLogonTimeout="180"
  lockoutTimeout="180"
  logonTimeout="180"
  startTime="00:00:00.00000"
  endTime="23:59:59.99999"
  loggingLevel="Both"
  allowAllCommands="false">
  <users>
    <userGroup name="standard">
      <users>
        <user name="poppy\akutz" allowAllCommands="true">
          <commands>
            <command
              path="c:\windows\system32\regedit.exe"
            />
          </commands>
        </user>
        <user name="lostcreations\akutz"
              allowAllCommands="false"
        />
      </users>
      <commandGroupRefs>
        <commandGroupRef commandGroupName="standard"/>
      </commandGroupRefs>
```

```
    </userGroup>
  </users>
  <commands>
    <commandGroup name="standard">
      <!--
        Windows XP SP2 files -
        checksums may vary per operating system,
        service pack
      -->
      <command path="c:\windows\system32\cmd.exe"
        md5Checksum="eeb024f2c81f0d55936fb825d21a91d6"
        argumentString="/^/K echo.*$/"
      />
      <command path="c:\windows\explorer.exe"
        md5Checksum="45757077a47c68a603a79b03a1a836ab"
      />
      <command path="c:\windows\system32\notepad.exe"
        md5Checksum="388b8fbc36a8558587afc90fb23a3b99"
        allowedNetworks="192.168.0.226"
      />
    </commandGroup>
  </commands>
</sudoers>
```

As seen, node relationships and attributes defined on the nodes contain the data that is parsed from the file. The following is a list of nodes that contain attributes and their definitions:

<sudoers>

The <sudoers> node is the root node of a file that implements the XmlAuthorizationSchema. Its attribute list is unique in that all attributes have default values. This is so if an attribute is not defined at a lower level the attribute will always have a value by virtue of having a default value on the root node.

Valid attributes are:

- **privilegesGroup**
  - **type** string
  - **use** optional
  - **default** Administrators
  - **description** The group that a user\u2019s privileges will be escalated to when they use sudo to execute an application.

- **startTime**
  - **type** TimeSpan
  - **use** optional
  - **default** 00:00:00.00000
  - **description** The earliest time of day that sudo can be invoked.

- **endTime**
  - **type** TimeSpan
  - **use** optional
  - **default** 23:59:59:99999
  - **description** The latest time of day that sudo can be invoked.

- **invalidLogons**
  - **type** integer
  - **use** optional
  - **default** 3
  - **description** The number of invalid logon attempts a user is allowed before they accrue one strike towards being locked out.

- **timesExceededInvalidLogons**
  - **type** integer
  - **use** optional
  - **default** 3
  - **description** The number of times a user is allowed to exceed the invalid logon attempt value before they are locked out.

- **logonTimeout**

  - **type** integer
  - **use** optional
  - **default** 180
  - **description** The number of seconds that the user\u2019s credentials will be cached upon a successful authentication.

- **lockoutTimeout**
  - **type** integer
  - **use** optional
  - **default** 180
  - **description** The number of seconds a user is locked out once they have exceeded the value specified by timesExceededInvalidLogons.

- **invalidLogonTimeout**
  - **type** integer
  - **use** optional
  - **default** 180
  - **description** The number of seconds sudowin keeps track of a user exceeding the invalid logon limit. For example, by default a user could exceed the invalid logon limit twice in 3 minutes and once more 4 seconds later, but the counter would have already reset and that user would not be locked out.

- **loggingLevel**
  - **type** loggingLevelType
  - **use** optional
  - **default** Failure
  - **description** The level of logging to adhere to. Valid values are Success, Failure, Both, and None. Success will log all successful attempts to invoke sudo. Failure will log all failed attempts to invoke sudo. Both will log both all successful and failed attempts to invoke sudo. None will log neither successful nor failed attempts to invoke sudo.

- **allowAllCommands**
  - **type** bool
  - **use** optional
  - **default** false
  - **description** Set to `true` to allow the execution of all commands or set to `false` to restrict commands to those explicitly allowed.

- **allowedNetworks**
  - **type** string
  - **use** optional
  - **default** *
  - **description** The networks that users are allowed to execute sudo from. This value can be set to * to match all networks, a host name, an IP address, or a Perl-compliant regular expression that matches a network syntax, such as `192\.168\.0\.[0-255]`. Multiple values are allowed by separating them with commas, for example `spindy,poppy,marmer,192.168.9.0,192\.\d{1,3}\.0.[5-30]`. If no value is specified for this attribute then all networks are considered valid.

<userGroup>

The `<userGroup>` node is used for grouping distinct users into a singular group. This is useful because it allows an administrator to set attribute values on a group of users instead of individual users. Valid attributes are:

- **name**
  - **type** string
  - **use** required
  - **description** The name of the user group. This value does not have to be unique; it is simply for an administrator to be able to easily distinguish between groups of users.

The remaining attributes of the `<userGroup>` node are the same as the `<sudoers>` node except they have no default value.

<user>

The <user> node defines a Windows user by their host or domain name plus their user name (the samAccountName). Valid attributes are:

- **name**
  - **type** string
  - **use** required
  - **description** The name of the user. This value must adhere to the following format: HOST_OR_DOMAIN_NAME\USER_NAME. For example, lostcreations\akutz.

The remaining attributes of the <user> node are the same as the <sudoers> node except they have no default value.

<commandGroup>

The <commandGroup> node is used for grouping distinct commands into a singular group. This is useful because it allows an administrator to set attribute values on a group of commands instead of individual commands. The name of this node, specified by the name attribute is important because it is what the <commandGroupRef> node references. Valid attributes are:

- **name**
  - **type** string
  - **use** required
  - **description** The name of this command group. This value should be unique as it is used by the <commmandGroupRef> node.
- **enabled** See the <command> node.
- **startTime** See the <sudoers> node.
- **endTime** See the <sudoers> node.
- **loggingLevel** See the <sudoers> node.

- **allowedNetworks** See the <sudoers> node.

<command>

The <command> node defines a command by its fully-qualified path, in addition to other possible command characteristics. Valid attributes are:

- **path**
  - **type** string
  - **use** required
  - **description** The fully qualified path of the command. For example, c:\windows\system32\cmd.exe.

- **argumentString**
  - **type** string
  - **use** optional
  - **description** The arguments that are valid with the given command. For example, specify /K so that the command shell must always be executed with the /K switch. This value also supports Perl-compliant regular expressions by beginning and ending the value with a /. For example, /^/K echo.*$/ will restrict this command so that its argument list must always begin with /K echo. If no value for this attribute is specified then all arguments will be allowed.

- **enabled**
  - **type** bool
  - **use** optional
  - **default** true
  - **description** Specify true to enable this command or false to disable it without removing its entry from the file.

- **md5Checksum**
  - **type** string
  - **use** optional
  - **description** The MD5 checksum of the given command. If specified then the server will not okay this command's execution if at execution time the command's MD5 checksum does not match what is in this file. Keep in mind that checksums can differ between OS version and service pack level.

- **startTime** See the <sudoers> node.

- **endTime** See the <sudoers> node.

- **loggingLevel** See the <sudoers> node.

- **allowedNetworks** See the <sudoers> node.

<commandGroupRef>

The <commandGroupRef> node allows an administrator to reference a defined command group instead of having to redefine in it multiple places. For example, if a command group named tools was defined:

```
<commandGroup name="tools">
  ...
</commandGroup>
```

The administrator could reference that group at another location with:

```
<commandGroupRef commandGroupName="tools" />
```

Valid attributes are:

- **commandGroupName**
  - **type** string
  - **use** required
  - **description** The name of the <commandGroup> node that this command group reference points to.

CredentialsCache

The credentials cache plugin persists a user's credentials and information about their authentication attempts between sudo invocations.

LocalServer

The local server plugin caches credentials in local memory. To ensure protection of said credentials it uses the .NET System.Security.SecureString when storing any type of private information.[6] The local server plugin must be configured as a Singleton object so that it is instantiated once and stays resident in memory. Otherwise it will not be useful to persist information between sudo invocations.

CallbackApplication

The sudowin callback application is extremely important. The sudowin Windows service must be able to both launch a process that is attached to the logged in user's desktop and be able to launch that process with a new logon token for the user so the user's temporarily

[6] http://msdn2.microsoft.com/en-us/library/system.security.securestring.aspx

assigned group membership is respected. The win32 API function CreateProcessAsUser enables developers to launch processes with a user's logon token, and that process will be attached to the same desktop that the user's logon token came from. Using the win32 Windows Terminal Services API (wtsapi32) it is possible to get the logon token for any user currently logged onto the computer. So sudowin uses the wtsapi32 to get the logon token for a given user and uses the CreateProcessAsUser function to create a process that is attached to the user's desktop.

However, creating a new process with an existing logon token is not enough. The existing logon token does not respect the given user's temporarily assigned group memberships. This is where the callback application comes into play. The process that the sudowin service creates is always the callback application with several arguments. The first argument to the callback application is the given user's passphrase, the second is the fully-qualified path of the command the user has invoked sudo on, and any additional arguments are arguments that go with the command the given user has invoked sudo on.

The callback application then launches a new process for the command the user invoked sudo on, but does so with the user's passphrase, creating a new logon token and a process that respects the user's temporarily assigned group memberships.

7. Walk Through

Below is a step-by-step walk through of the events that occur when the sudowin service is started and during a sudowin client invocation. There are a few things to keep in mind when reading the walk through. The sudowin server object and some plugin objects are configured as SingleCall objects. This means every time any of their methods are invoked, the object is created anew. Therefore any code that the object runs upon creation, for example its constructor, gets called every time one of its methods is invoked.

Service Startup

1. The sudowin service is started by the Service Control Manager (SCM) either by code, the GUI interface, or the command `net start sudowin`.

2. The sudowin service controller callback method `OnStart` is triggered. The method checks its arguments to see if they were set. If so, the method will sleep the current thread for the number of seconds specified by the first argument passed to the method. This is so developers can easily pause the startup routine of the sudowin service in order to attach debuggers to it before things really get moving.

3. The `OnStart` method then loads the sudowin service's application configuration file and configures its application domain's remoting environment by parsing the `system.runtime.remoting` section from the configuration file.

4. The `OnStart` method then verifies the plugin configuration file as specified by the `pluginConfigurationUri` value in the server's configuration file against the plugin configuration schema as specified by the `pluginConfigurationSchemaUri` value in the server's configuration file. If the plugin configuration file cannot be verified an exception will be thrown that crashes the sudowin service. Administrators can

check the event log in order to determine exactly why the exception was caused.

5. If the plugin configuration file was loaded successfully, the sudowin service will register the configured plugins as remoting objects and then obtain references to the objects it just registered. After the plugin's reference is obtained the plugin has its Activate method invoked. This method invocation will cause the plugin object to be instantiated. Even though no plugins are used at this stage, the point of forcing plugin instantiation at this stage is to see if instantiating any of the configured plugins will cause an exception. If so then it is better it should happen now when the service is starting then later when a user first invokes a sudowin client. If a plugin activation throws an exception then that exception will crash the sudowin service. Administrators can check the event log in order to determine exactly why the exception was caused.

6. The sudowin service has been started and is ready to accept connections from clients.

Client Invocation

1. Mandy invokes a sudowin client.

2. The client verifies that the sudowin service is running. If it is not, the client will inform the user that the sudowin service is not running.

3. The client attempts to establish a secure connection with the sudowin server. The client must then determine whether or not it has established a successful connection to the sudowin server. It does this by invoking the server method `IsConnectionOpen`. This method always returns `True`, the real test is whether or not invoking this method throws an exception. If this method invocation does not throw an exception then the client knows a successful connection has been established. Because the sudowin server is a `SingleCall` object, this method invocation also instantiates the sudowin server object.

4. When the sudowin server object is instantiated it verifies the plugin configuration file as specified by the

pluginConfigurationUri value in the server's configuration file against the plugin configuration schema as specified by the pluginConfigurationSchemaUri value in the server's configuration file. If the plugin configuration file cannot be verified an exception will be thrown that halts the current client invocation. The user can report the error to an administrator so that they can check the event log in order to determine exactly why the exception was caused.

5. If the plugin configuration file was loaded successfully, the sudowin server object will obtain references to the plugin objects that have already been registered when the service was started. After the plugin's reference is obtained the plugin has its Activate method invoked. If a plugin activation throws an exception then that exception will halt the current client invocation. The user can report the error to an administrator so that they can check the event log in order to determine exactly why the exception was caused.

6. If Mandy is not a member of the configured Sudoers group then an exception is thrown that the client should catch and inform Mandy that she is not allowed to continue. If Mandy is a member of the configured Sudoers group then the process continues.

7. The client then asks the server if Mandy has exceeded her invalid login limit, i.e. she is locked out. This action instantiates the sudowin server object. Please see Steps 4 and 5 for a description of what occurs when the sudowin server object is instantiated. If Mandy is locked out then the server will inform the client of this. The client should inform Mandy she is locked out and quit gracefully. If the client did attempt to proceed the sudo invocation would fail because the server knows Mandy is locked out. If Mandy is not locked out then the process continues unabated.

8. The server checks to see if Mandy has cached credentials. This action instantiates the sudowin server object. Please see Steps 4 and 5 for a description of what occurs when the sudowin server object is instantiated. If Mandy has cached credentials then the client invokes the server's Sudo method. If Mandy does not have cached credentials the client should prompt Mandy for her credentials and once obtained the client will invoke the server's Sudo method. Invoking the server's

`Sudo` method instantiates the sudowin server object. Please see Steps 4 and 5 for a description of what occurs when the sudowin server object is instantiated.

9. The sudowin server uses its authorization plugins to verify that Mandy exists in one of the authorization plugins' data sources. If she does not, the sudowin server returns the status code `CommandNotAllowed` to the client. If Mandy does exist in one of the authorization data sources the process continues.

10. The sudowin server checks to see if Mandy is locked out. If she is, the sudowin server returns the appropriate status code to the client. If Mandy is not locked out the process continues.

11. The sudowin server checks to see if Mandy has cached credentials. If she does then the sudowin server will proceed using the cached credentials set. If Mandy does not have cached credentials the sudowin server will attempt to validate the credentials that the client passed to the sudowin server earlier using the authentication plugins. If the credentials are not validated then the status code `InvalidLogon` will be returned to the client. If the credentials are validated then they will be cached for the amount of time specified for Mandy in the authorization plugin data source.

12. The sudowin server then authorizes the command that the sudowin client as invoked on using the authorization plugins. If the command is not authorized the status code `CommandNotAllowed` is returned to the client. If the command is authorized then the process continues.

13. Next, the sudowin server verifies that the server binary and the callback application binary, as specified by the `callbackApplicationPath` value in the server's configuration file, are both signed by the same strong name key. This is to ensure that the callback application has not been tampered with. If the signatures do not match then the status code `CommandNotAllowed` is returned to the client. If the signatures do match then the process continues.

14. The sudowin server uses wtsapi32 to query Mandy's user token from her active desktop session.

15. The sudowin server checks to see if Mandy is already a member of the privileges group specified by her entry in the authorization data source. If Mandy is already a member then a note is made of this, otherwise she is added to the group.

16. The sudowin server uses Mandy's token to create a process for the callback application. Because Mandy's token from her desktop session was used, the callback application's process will be created bound to Mandy's desktop. The server then waits for the callback application process to conclude before continuing.

17. The callback application launches the intended command with Mandy's credentials, thereby creating a new user token for Mandy and associating that token with the process the intended command was launched in. Because Mandy was added to the privileges group by the server before this occurred, the new process respects Mandy's new group membership and thereby has elevated privileges.

18. Once the callback application process has concluded the sudowin server checks to see if Mandy was a member of the privileges group prior to this sudowin invocation. If she was then she is left in the group. If Mandy was added to the privileges group for the sole purpose of this sudowin invocation she is then removed from the privileges group.

19. This concludes a sudowin client invocation.

8. Implementation

Sudowin is a complex product that provides a simple solution to an intricate problem. Luckily for the end-user, implementing sudowin on a desktop is quite easy.

Requirements

Sudowin requires the following:

- Operating System
  - Microsoft Windows XP Professional
  - Microsoft Windows Server 2003
  - Microsoft Windows Vista Professional+
  - Microsoft Windows Server Longhorn
- Microsoft .NET 2.0 Framework
- The Terminal Services service

Installing

1. Download either the EXE or MSI installer for the latest release from http://sourceforge.net/project/showfiles.php?group_id=143653.
2. Install sudowin by double-clicking on the installer.
   a. The installation will prompt the user to decide whether or not to allow anyone to use sudowin, or only the installing user. The default choice is everyone.

b. The installation will prompt for an installation location. The default installation location is %ProgramFiles%\Sudowin.

Upgrading

There have already been several releases of sudowin. For this reason, it is impractical to discuss all of the upgrade scenarios in this document. Please see the online documentation regarding how to upgrade sudowin from a previous version at http://www.lostcreations.com/sudowin/documentation#upgrading.

Configuring

If sudowin has been upgraded from version 0.1.1-r95 or up to a more recent version then it should be ready for use immediately after the upgrade. However, an earlier upgrade or a fresh installation will require some configuration before sudowin can be used.

The Sudoers Group

1. Click "Start" → "Run" and type "compmgmt.msc". This will bring up the "Computer Management" application.
2. Under "System Tools" find "Local Users and Groups" and expand it so that it is possible to click on "Groups".
3. There should be a group called "Sudoers". Double-click on it.
4. Add the user accounts to this group that should be able to use sudowin on this computer.
5. Any user that is added to the "Sudoers" group will need to log out and back in to this computer before they can use sudowin. This is because Windows only loads group

memberships when a user logs in and their profile is loaded.

The Sudoers File

1. Use notepad to open the "sudoers.xml" file the installer places in INSTALLDIR\Sudowin\Server.

2. Find the section called <users> and add a user node for the user that should be enabled to use sudowin. Please see the Sudowin.Plugins.Authorization.Xml plugin documentation for information on how to configure a user and what commands they are allowed to invoke sudo on.

Uninstalling

1. Use "Add/Remove Programs" or the original installer to remove sudowin.

2. The uninstaller will ask if the "Sudoers" group and the "Sudoers" file should be removed. Only remove these if sudowin is not being uninstalled in preparation for an upgrade.

Locations

Some users are very obsessive about uninstalling every last bit of information about a program when that program is being removed. To that end, the following is all of the data that sudowin creates on a computer's hard drive upon installation.

Files

```
INSTALLDIR\
|
|-- Callback\
|   |
|   |-- Sudowin.CallbackApplication.exe
|   |-- Sudowin.Common.dll
|
|-- Clients\
```

```
|    |
|    |-- Console\
|    |    |
|    |    |-- sudo.exe
|    |    |-- sudo.exe.config
|    |    |-- Sudowin.Common.dll
|    |
|    |-- Gui\
|         |
|         |-- Sudowin.Clients.Gui.exe
|         |-- Sudowin.Clients.Gui.exe.config
|         |-- Sudowin.Common.dll
|
|-- Server\
|    |
|    |-- pluginConfiguration.xml
|    |-- PluginConfigurationSchema.xsd
|    |-- sudoers.xml
|    |-- Sudowin.Common.dll
|    |-- Sudowin.Plugins.Authentication.dll
|    |-- Sudowin.Plugins.Authentication.NT.dll
|    |-- Sudowin.Plugins.Authorization.dll
|    |-- Sudowin.Plugins.Authorization.Xml.dll
|    |-- Sudowin.Plugins.CredentialsCache.dll
|    |-- Sudowin.Plugins.CredentialsCache.LocalServer.dll
|    |-- Sudowin.Plugins.dll
|    |-- Sudowin.Server.exe
|    |-- Sudowin.Server.exe.config
|    |-- Sudowin.Server.InstallState
|    |-- WtsApi32.NET.dll
|    |-- XmlAuthorizationPluginSchema.xsd
|
|-- Setup\
|    |
|    |-- CustomActions\
|         |
|         |-- Sudowin.Setup.CustomActions.dll
|         |-- Sudowin.Setup.CustomActions.InstallState
|
|-- CHANGLOG.txt
|-- LICENSE.txt
|-- README.rtf
|-- README.txt

HOMEDIR (or ALLUSERSHOMEDIR depending on the installation step that asks
if this program is to be used by only you or everyone)
|
```

```
|-- Start Menu\
    |
    |-- Programs\
        |
        |-- Sudo for Windows\
            |
            |-- Sudo for Windows.url
```

Registry

```
HKEY_CLASSES_ROOT\
|
|-- batfile\
|   |
|   |-- shell\
|       |
|       |-- sudo\
|           |
|           |-- command\
|           |   |
|           |   |-- (Default) =
""[INSTALLDIR]Clients\Gui\Sudowin.Clients.Gui.exe" "%1" %*"


|           |
|           |-- (Default) = "Sudo..."
|
|-- cmdfile\
|   |
|   |-- shell\
|       |
|       |-- sudo\
|           |
|           |-- command\
|           |   |
|           |   |-- (Default) =
""[INSTALLDIR]Clients\Gui\Sudowin.Clients.Gui.exe" "%1" %*"


|           |
|           |-- (Default) = "Sudo..."
|
|-- exefile\
|   |
```

```
|   |-- shell\
|       |
|       |-- sudo\
|           |
|           |-- command\
|           |   |
|           |   |-- (Default) =
""[INSTALLDIR]Clients\Gui\Sudowin.Clients.Gui.exe" "%1" %*"

|           |
|           |-- (Default) = "Sudo..."
|
|-- Folder\
|   |
|   |-- shell\
|       |
|       |-- sudo\
|           |
|           |-- command\
|           |   |
|           |   |-- (Default) =
""[INSTALLDIR]Clients\Gui\Sudowin.Clients.Gui.exe"
%SystemRoot%\Explorer.exe /e,/idlist,%I,%L"

|           |
|           |-- (Default) = "Sudo..."
|
|-- Msi.Package\
|   |
|   |-- shell\
|       |
|       |-- sudo\
|           |
|           |-- command\
|           |   |
|           |   |-- (Default) =
""[INSTALLDIR]Clients\Gui\Sudowin.Clients.Gui.exe"
"%SystemRoot%\system32\msiexec.exe" /package "%1" %*"

|           |
|           |-- (Default) = "Sudo..."
```

Groups

The installer creates a local group called "Sudoers." The uninstaller will prompt for its removal, and if it is not removed then the group will remain until it is manually removed.

Active Directory

Even if an organization possesses hundreds of servers and thousands of desktops, it is incredibly easy to leverage sudowin on each and every one of them. Sudowin is available for download as a Microsoft Software Installer (MSI) file. This means that sudowin can be deployed to any number of desktops using Group Policy. For more information about deploying software packages via Group Policy please see http://support.microsoft.com/kb/314934.

Known Issues

There are some known issues with using sudowin:

1. The XML Authorization plugin does not currently support adding built-in user groups to the data source. This feature is intended for a future release.

2. Sudowin does not currently support escalating privileges to the level of Active Directory domain groups. This feature is intended for a future release when the domain controller component of sudowin is completed.

3. There is currently no GUI editor for the XML Authorization

plugin file. This is a low priority and the suggested method for editing this file is with a text editor such as notepad.

4. Sudowin does not work with Windows XP Home edition. This is because this version of Windows is stripped of the terminal services component – a service that sudowin relies upon in order to query a user's logon token.

9. Conclusion

Sudowin was developed with the purpose of promoting a more traditional security model like the one found in use by UNIX and Linux systems. Because of the conscious effort put into the sudowin project from the beginning regarding high security and end-users, sudowin enables users of Microsoft Windows to practice the principal of least privilege without sacrificing ease of use.

A lot of people ask why there is still a need for a separate Sudo for Windows project when Windows Vista includes sudo functionality. Sadly, these people are misinformed, because despite the propagated misinformation, there is no sudo in Windows Vista.

Windows Vista implements a feature called User Account Control (UAC).[7] UAC includes support for Over-the-Shoulder (OTS) Credentials – that is a user is prompted for an administrative passphrase when an administrative task needs to be accomplished. If the user does not know the passphrase then they are not allowed to perform the task. This is obviously not sudo because the user must know a passphrase that is not her own to accomplish a task.

UAC also introduces Admin Approval Mode. This is what is confused for sudo. Essentially, administrators are prompted for their credentials or their consent whenever they need to perform a sensitive task. Because the administrators are being prompted for their own passphrase this may seem a lot like sudo, but there is one very important thing to remember – the administrator is not being

[7] http://www.microsoft.com/technet/windowsvista/security/uacppr.mspx

granted any privileges that she does not already have. There is no privilege escalation occurring. There is no sudo in Microsoft Vista.

Microsoft continues its journey into the 21st century with operating systems designed for high security, but these operating systems still lack one of the most basic pieces of functionality towards security that UNIX and Linux have enjoyed for more than 20 years. While it would be a fantastic step in the right direction towards a more secure tomorrow if millions of Windows users started using sudowin, the Sudo for Windows project also exists for another reason. The sudowin software is only a byproduct of a larger goal – to inform those millions of users that there are always ways to make themselves, and their neighbors connected to them all over the world by the internet, more secure.

Users may follow the progress of the sudowin project by visiting its SourceForge project page at http://sourceforge.net/projects/sudowin or its homepage at http://sudowin.sourceforge.net (redirects to http://www.lostcreations.com/sudowin). This practical, although static, will see updates via the documentation on the sudowin site at http://www.lostcreations.com/sudowin/documentation. Finally, this practical leaves the reader with a question.

Do you sudo?

Last Updated: October 2nd, 2013

Upcoming SANS Training

Click Here for a full list of all Upcoming SANS Events by Location

SANS Seattle 2013	Seattle, WAUS	Oct 07, 2013 - Oct 14, 2013	Live Event
SANS Baltimore 2013	Baltimore, MDUS	Oct 14, 2013 - Oct 19, 2013	Live Event
SEC760 Advanced Exploit Development for Penetration Testers	Baltimore, MDUS	Oct 14, 2013 - Oct 19, 2013	Live Event
SANS Bangalore 2013	Bangalore, IN	Oct 14, 2013 - Oct 26, 2013	Live Event
GridSecCon 2013	Jacksonville, FLUS	Oct 15, 2013 - Oct 17, 2013	Live Event
Healthcare Cyber Security Summit	San Francisco, CAUS	Oct 17, 2013 - Oct 24, 2013	Live Event
Securing the Internet of Things Summit	San Francisco, CAUS	Oct 17, 2013 - Oct 22, 2013	Live Event
SANS Tokyo Autumn 2013	Tokyo, JP	Oct 21, 2013 - Oct 26, 2013	Live Event
ICS410	Sterling, VAUS	Oct 21, 2013 - Oct 25, 2013	Live Event
October Singapore 2013	Singapore, SG	Oct 21, 2013 - Nov 02, 2013	Live Event
SANS Dubai 2013	Dubai, AE	Oct 26, 2013 - Nov 07, 2013	Live Event
FOR572 Advanced Network Forensics and Analysis	Washington, DCUS	Oct 28, 2013 - Nov 02, 2013	Live Event
SANS Chicago 2013	Chicago, ILUS	Oct 28, 2013 - Nov 02, 2013	Live Event
MGT415 at (ISC)2 SecureSoCal 2013	Manhattan Beach, CAUS	Oct 31, 2013 - Oct 31, 2013	Live Event
SANS South Florida 2013	Fort Lauderdale, FLUS	Nov 04, 2013 - Nov 09, 2013	Live Event
SANS DHS Continuous Diagnostics & Mitigation Award (CDM) Workshop	Washington, DCUS	Nov 06, 2013 - Nov 06, 2013	Live Event
MGT415 at (ISC)2 SecureDallas 2013	Dallas, TXUS	Nov 06, 2013 - Nov 06, 2013	Live Event
SANS Pen Test Hackfest Training Event and Summit	Washington, DCUS	Nov 07, 2013 - Nov 14, 2013	Live Event
SANS Sydney 2013	Sydney, AU	Nov 11, 2013 - Nov 23, 2013	Live Event
SANS Korea 2013	Seoul, KR	Nov 11, 2013 - Nov 23, 2013	Live Event
Cloud Security @ CLOUD Expo Asia	Singapore, SG	Nov 13, 2013 - Nov 15, 2013	Live Event
SANS London 2013	London, GB	Nov 16, 2013 - Nov 25, 2013	Live Event
SANS San Diego 2013	San Diego, CAUS	Nov 18, 2013 - Nov 23, 2013	Live Event
FOR585 Adv Mobile Device Forensics	Vienna, VAUS	Nov 18, 2013 - Nov 23, 2013	Live Event
Asia Pacific ICS Security Summit & Training	Singapore, SG	Dec 02, 2013 - Dec 08, 2013	Live Event
SANS San Antonio 2013	San Antonio, TXUS	Dec 03, 2013 - Dec 08, 2013	Live Event
SANS Cyber Defense Initiative 2013	Washington, DCUS	Dec 12, 2013 - Dec 19, 2013	Live Event
SANS Oman 2013	Muscat, OM	Dec 14, 2013 - Dec 19, 2013	Live Event
SANS Golden Gate 2013	San Francisco, CAUS	Dec 16, 2013 - Dec 21, 2013	Live Event
SANS Forensics Prague 2013	OnlineCZ	Oct 06, 2013 - Oct 13, 2013	Live Event
SANS OnDemand	Books & MP3s OnlyUS	Anytime	Self Paced