The following paper was originally presented at the
Ninth System Administration Conference (LISA '95)
Monterey, California, September 18-22, 1995

# Multi-platform Interrogation and Reporting with Rscan

Nathaniel Sammons
Colorado State University

# Multi-platform Interrogation and Reporting with Rscan

*Nathaniel Sammons* – Colorado State University

## ABSTRACT

This paper describes Rscan – a tool that allows a System Administrator to easily write and execute scripts on a single machine or an entire network of machines. Rscan can be used to automate security checks, configuration checks and many other tasks. If a module (a collection of scripts) is written with different sections for different operating systems and different versions of each operating system (as they are intended to be written), Rscan will automatically select the correct set of scripts to execute on any particular operating system.

As Rscan runs it generates reports which can be written as either plain ASCII or HTML files. Rscan is specifically designed to be run in a heterogeneous environment and is easily adapted to new and different operating systems. Two running modes are available: text mode and GUI mode. When running in text mode, all output that is generated is sent to either standard out or standard error. When running in GUI mode, Rscan starts a WWW browser such as Netscape Navigator or NCSA Mosaic and the user can configure the system, run scans and read reports using the browser as a GUI.

### Motivation

There have long existed programs like COPS [1] and TAMU [3] that are capable of scanning an individual computer for a set of security holes, but they have traditionally been limited to running generic scans that look for problems that occur in the "general case." These utilities have historically not included code that examines a system for security holes that are unique to a particular operating system, nor have they included a way to easily integrate custom scans, or to automatically select specific scans based on the operating system of the machine being scanned. COPS alluded to the fact that a system able to automatically select which scans to run based on the operating system of the machine being scanned would be a step in the right direction, but did not offer any solutions.

There have also been applications such as `rdist` that are designed to quickly and easily distribute files to many hosts on a network. There has not, however, been a tool that is specifically designed to run in a heterogeneous environment and execute complicated scripts on many machines and to execute different scripts under different operating conditions.

Rscan is intended to fill this void. It offers a uniform way to run any number of independent scans on any number of computers and organize the results of all the scans in a clean, formatted report. Rscan will also automatically select the appropriate scan for any operating system provided that a module has been written with specific code for that operating system. It also provides a straightforward, uniform method of selecting which modules to execute on a particular system.

### Designing Rscan

Rscan started out as an IRIX-only security utility, but has outgrown those bounds and become a modular, extensible interrogation tool. In rewriting the original code, the following design goals were considered:

1. The solution must not be overly cumbersome or its obfuscation will overshadow its usefulness.
2. Setup and configuration must be simple and logical.
3. The system must be as robust and fault-tolerant as possible.
4. Writing modules must be a straightforward process and there should exist an API to help facilitate this.
5. Reports must be nicely formatted for ease of reading, and there should be a uniform method to write reports to help keep them organized.
6. The interface must be simple and functional and not get in the way of actually getting work done.

Software should bend over backwards to meet the user's needs, not visa versa, so a lot of effort was put into making Rscan robust enough to keep users from needing to modify the code to meet their needs. Almost every aspect of Rscan can be customized through the use of command line options and/or configuration file options.

Rscan was originally intended to be used only for security checking, and currently all of the publicly available modules do just that, but Rscan is robust enough to provide functionality far beyond security checks. Rscan can be used to implement custom scripts for a site that check for proper configurations and perform other tasks that have primarily been done manually in the past. For instance, a module can easily be written to check that a set of utilities has been installed on a new machine, or that the utilities that are installed have not been tampered with, etc.

An Rscan module can have both operating system independent and operating system specific parts to it. If the module is selected for execution on a particular machine, then all the OS-independent scans will be run, as will any OS-specific scans. For every module, there is an OS-independent initialization script that sets up any variables, functions, and other data that the module will need to run under any operating system. For each OS-specific set of scans in a module, there is a corresponding OS-specific initialization script that can set up other, more specialized information that the module will need to run (such as the path to a precompiled MD5 checksum utility, etc). Within an OS-specific section of a module, there is a set of scans that run on all versions of that operating system and a set that run only on a specific version of that operating system (e.g. 4.0.5 or 5.3 under IRIX or 4.1.1 or 4.1.3 under SunOS). All, some or none of these sections need to be present for the module to function, as Rscan will automatically decide what to run. A module could easily be written that runs the proper copy of Crack or Tripwire for a particular operating system on a network of hundreds or thousands of hosts, summarizing its results in a central report as it runs.

Rscan can be used to encapsulate the functionality of any tool that only runs on one machine at a time by writing a simple "wrapper" script (which could be as simple as starting a subprocess and relaying everything it says using the `&report()` API function) that runs the tool on the remote machine. Trivial modifications can be made to a module so that the right, often operating system specific, command line arguments are passed to the tool when it is run on the remote machine. This is Rscan's most attractive feature: making it easy to write a tool that runs on all the operating systems in an organization, and can be run from a central location with the touch of a button.

The design incorporates a flexible reporting mechanism that can generate a series of reports (one for each host), or a single report for the entire network. A set of functions to be used by all modules to report progress and test outcomes to the screen and to write data to the report file is provided. Reports can be written in either plain ASCII or in HTML. Rscan will automatically properly reformat data based on whether it's writing an ASCII or HTML report.

**General Implementation**

Rscan is implemented in perl [2] for many reasons. Perl is widely regarded as the de-facto system administration language, and it provides many attractive features, especially when designing for multi-platform use. Since the code for Rscan does not use any of the new functionality provided in perl 5, it will run under either version 4 or 5 of perl.

When scanning, each module is run in its own isolated process. Inside each module, every individual scan is also run in its own process. This is done to ensure that every module can do anything it wants to (including modification of system-defined variables, etc) and not have to worry about how its actions affect the other modules. Variables can be changed and anything can be done inside a scan without worrying about how it will affect other scans in that module or other modules that will be run after its run has been completed. In addition to these benefits, if a scan or an entire module crashes, it will not abort the entire scanning run. In the case that a scan in a module or an entire module exits abnormally (with an exit value other than 0, usually indicating a crash), Rscan will log to the report that the process exited abnormally, giving the scan or module that was running, its process ID number, and its exit value.

---

| |
|---|
| `rscan` is run on the local machine, it then copies the necessary modules and the `scan` script to the remote machine, initiates an `rsh` to start the remote copy of `scan` and listens to its output. |

↑
(network pipe carries data)

| |
|---|
| `scan` starts, and forks a copy of itself off into the background, relaying everything its child process outputs to the network pipe. A new child process is started synchronously for each module that will be run. |

↑
(pipe carries data)

| |
|---|
| `scan` reads the module's initialization scripts and then executes its scanner scripts. See the "Running order for files" section for the exact order that initialization and scanner scripts are run in. Processes are started synchronously for each scanner script after the initialization scripts have been run. |

↑
(pipe carries data)

| |
|---|
| This process simply executes a particular scan script. |

**Figure 1**: Typical Rscan process structure

Rscan is composed of three scripts, named "`rscan`," "`scan`" and "`modman`." `rscan` is run on the machine doing the scanning of the other hosts, and acts as a supervisor for the actual scanning script, `scan`. `rscan` copies `scan` and the necessary modules to the host that will be scanned. `scan` is then executed by `rscan`, and it sends data back across a UNIX pipe. Each module is executed in much the same way on the remote machine. `Scan` forks a copy of itself off and that second copy then runs any initialization scripts for the module it will be running (both OS-independent and OS-specific ones), then forks a copy of itself off to run each scan in its module. Each process is connected to its parent by a UNIX pipe. Figure 1 shows the typical process structure of an Rscan run.

The third script, `modman`, is a *MOD*ule *MAN*agement utility that simplifies installing new modules and other module administration tasks. Modman installs, removes, and backs up modules and provides an easy way of tracking installed modules.

### GUI Implementation

When running in GUI mode, Rscan starts two very simple HTTP [4] servers that each listen to a unique port on the machine acting as the server. Rscan generates a 30 character session key by viewing the current network statistics (obtained from the output of `netstat`) and by examining the current process table. This session key makes it much harder (though not impossible, as with any key) for another host to interact with Rscan remotely. Rscan then starts up a WWW browser (Netscape Navigator is recommended, but others will work) with the URL that points to the first HTTP server.

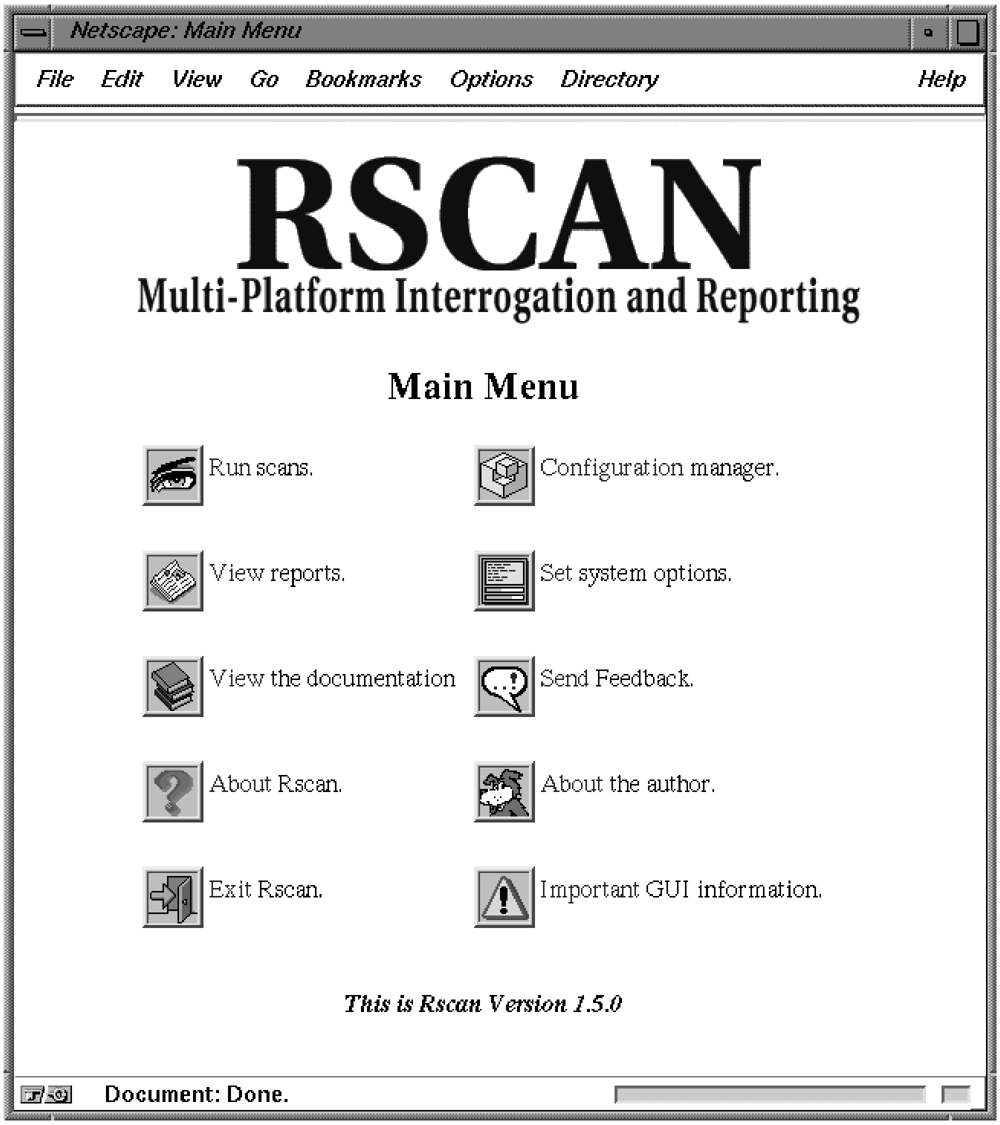


**Figure 2**: Main Menu in GUI Mode

One of the HTTP servers only serves images and the other only serves documents and controls scans. This is done so that while a scan is running, and thus tying up the document server, inlined images can still be inserted into the output of the document server.

When Rscan starts up in GUI mode, it displays the main menu, which is shown in Figure 2. From this menu, the user can choose to run scans, edit configuration files, view reports, set system options, view documentation, send feedback to the author, view the Rscan home page, read about the author, exit Rscan or view special documentation about running in GUI mode.

Using a WWW browser as a GUI presents one glaring problem. Most WWW browsers now implement the `Referrer:` field in the HTTP request header, which will disclose the session key to the server which holds the next URL being accessed. If a user chooses to jump directly to a new URL that is outside of the servers that Rscan is running by selecting the "Open URL" menu or its equivalent, there is no way to keep the session key from being disclosed, and there is no way for the servers run by Rscan to know that the user chose to open a URL that's outside of the Rscan servers – this is simply a fact of how HTTP and WWW browsers function internally. Because of this, users are *strongly* advised not to jump directly to a URL while running Rscan. Occasionally, it is necessary to present the user with a URL in a hypertext reference that points outside of the Rscan servers (this happens frequently when viewing reports, and even in some of the main menu options). When this is done, Rscan translates that URL into a Referrer screen page. The Referrer screen page notifies the user that they are about to jump outside the confines of the Rscan servers. The user is presented with a hypertext reference that will cause Rscan to shut itself off while leaving the WWW browser running. If the user chooses to follow the reference, then Rscan shuts down the document and image servers, and prints a final HTML page that tells the user that the servers have been shut down and they will not be able to do more scanning until they restart Rscan. At the bottom of the page is a hypertext reference that points to the original URL that is outside of the Rscan servers. This may seem like a lot of work to go through, but it ensures that any URL that is presented to the user will not compromise the security of the Rscan servers.

Once the `Referrer:` problem has been dealt with, WWW browsers provide one great advantage over traditional GUI methods: They are very portable. All that needs to be available on a prospective platform for Rscan to run is a version of perl that supports the use of UNIX sockets and a WWW browser. Although this does not let Rscan take advantage of some of the features of, for instance, the TkPerl implementation, it is much easier to port to a new operating system. Because of the simplicity of the HTTP specification, writing an HTTP server to act as a core for a GUI is not particularly hard, and since most browsers support the FORMS standard, input and interaction with the user is clean and the interface is aesthetically pleasing.

## Using Rscan

Rscan was designed to be easily used by System Administrators of almost all skill levels. It can be run in two basic modes: regular (text) mode, and GUI mode. Quiet mode is just like text mode, but standard out is closed so that no progress information is written while scans are being run. In text mode, progress information is written to standard out while Rscan is running. In GUI mode, as discussed previously, a WWW browser is used as an interface and the user can configure Rscan, read reports and run scans.

### Configuration

All command line options (except for `-gui` and `-guidebug`) are only available when not running in GUI mode. Below is a list of command line options and the function of each.

**-ascii** Forces the writing of ASCII reports (which is the default). This is used to override the `html reports` configuration file option in the configuration file being used.

**-cf filename** Uses a different configuration file than the default, which is "`config/default.cf`." If a relative pathname is supplied, then it is taken to be relative to inside the `config` directory, if it's an absolute path, then its relative to the root directory.

**-config** This enters the text-based interactive configuration file editor, from which users can move, rename, delete, edit, and (most importantly) test configuration files. The path to the WWW browser to be used with the `-gui` and `-guidebug` options can also be set.

**-debug** Turns on (copious) debugging output while in text mode. This is handy if users are having a problem or a scan is not working properly.

**-gui** Causes Rscan to run in GUI mode. This option is special in that if it is used, it must be the *only* option used.

**-guidebug** Like the `-gui` option, this causes Rscan to run in GUI mode, but it also turns on debugging output like the `-debug` option. This option is special in that if it is used, it must be the *only* option used.

**-h** This prints a list of command line options, short descriptions of each, and information on their usage.

**-html** Forces the reports to be written in HTML. If reports are written in HTML, report filenames have a `.html` added to them (although if a user specifies a reportname that ends in `.html` then another `.html` will not be added to the end).

**-list** Lists all installed modules by short and long name, and also prints the module's version number. Every scan in each module is also listed with a brief synopsis of what its function is.

**-local** Runs the scans on the local machine only. If a user wants to specify a modlist for the local machine and there is not a default one set in the configuration file, or the user not using a configuration file, they should specify the modlist using the `-modlist` option.

**-machine** Specifies a set of machines to scan and associates a location for perl and a modlist with each. Users can have as many `-machine` options as they like, and conflicting machine definitions will be resolved in the order they appear: the last definition for a machine sticks. The format for this option is as follows:
`machine` *host1,...,hostN /path/to/perl modlist*
where *host1,...,hostN* is the set of hosts to scan, */path/to/perl* is the path to the perl executable on the machines or the word "`same`" if perl is located in the same place as perl that is running the Rscan process, and *modlist* is the modlist to associate with each host. Users can give a modlist of "`any`", in which case all applicable modules and scans in those modules will be run.

**-modlist modlist** Sets a global modlist for all scans. This will override options set in the configuration file, and all modlists given on the command line (in `-machine` options).

**-nocf** Don't even look at any of the configuration files. If this is not given, then the default configuration file is used (which can be overridden by the `-cf` option).

**-quiet** Turns off all printing to standard out. This is good if a user is running Rscan from a crontab file and don't want to get mail all the time about it running. Reports are still written.

**-reportdir directory** Specifies an alternate report directory. If it is a relative path (does not start with the "/" character), then it is taken to be relative to the directory that Rscan is in, otherwise it is taken to be an absolute path.

**-reportname format** Specifies the report naming format to be used. Report name formats are a string of text (whitespace and the % character are not allowed) that can use the substitutions shown in Table 1. The default reportname format is:
`rscan.%H-%M.%D.%Y-%h:%m:%s`
Because some of these characters are considered meta-characters by some shells, it is recommended that a report name format be enclosed in "double quotes" when issued on the command line.

**-separate** Writes a separate report for each host that is scanned. The %H in the report name format is replaced with the name of the host (which is the word "`network`" if separate reports are not being written).

**-single** Forces the writing of a single report. This is used to override the `separate reports` option in a configuration file.

**-tmp directory** Uses the given directory as the temporary directory when running scans on remote machines. The default is `/tmp`.

| | |
|---|---|
| %I | Integer time (seconds since Jan. 1, 1970) |
| %H | hostname or "network" |
| %M | month number |
| %D | day of the month number |
| %Y | two-digit year number (ex "95") |
| %h | hour when Rscan was started |
| %m | minute when Rscan was started |
| %s | second when Rscan was started |

**Table 1**: Substitutions for -reportname

Configuration files are basically a way of collecting command line options together for reuse. The default configuration file is `default.cf` located in the `config` directory. Other configuration files can be selected by using the `-cf` *filename* command line option, as described above. In a configuration file, options are given one per line. Blank lines and lines beginning with the "`#`" character are ignored. Options can be continued over multiple lines by using the "`\`" character at the end of a line. If a line is continued, then all whitespace after the \ and all whitespace at the beginning of the next line is deleted. Configuration file options are listed below.

**html reports** Writes reports in HTML. If reports are written in HTML, report filenames have a `.html` added to them (although if the user specifies a reportname that ends in `.html` then another `.html` will not be added to the end).

**machine** Specifies a set of machines and associates a location for perl and a modlist with each. Users can have as many `machine` options as they like, and conflicting machine definitions will be resolved in the order they appear: the last definition for a machine sticks. The format is as follows:
`machine` *host1,...,hostN /path/to/perl modlist*
where *host1,...,hostN* is the set of hosts, */path/to/perl* is the path to the perl executable on the machines or the word "`same`" if perl is located in the same place as perl that is running the Rscan process, and *modlist* is the modlist to associate with the hosts. If the modlist is left out, a modlist of "`any`" is assumed.

**modlist modlist** Sets a global modlist for all scans. This will override options set in the modlist field for each machine definition in the configuration file.

**reportdir directory** Specifies an alternate report directory. If it is a relative path (does not start with the "/" character), then it is taken to be relative to the directory that Rscan is in, otherwise it is taken to be relative to the root directory.

**reportname format** Specifies the report naming format to be used. See the description of the `-reportname` command line option for how to write a reportname format.

**separate reports** Writes a separate report for each host that is scanned. The %H in the report name format is replaced with the name of the host (which is the word "`network`" if separate reports are not being written).

**tempdir directory** Uses the given directory as the temporary directory when running scans on remote machines. The default is `/tmp`.

**Editing Configuration Files**

Configuration files can be edited in several ways. They can be edited by hand using a text editor such as `vi` or `emacs`. They can be edited by invoking Rscan with the `-config` option, which invokes a rudimentary text-based interactive configuration file editor, or they can be edited from within the GUI.

When edited from within the GUI, the user can edit the file by hand in the WWW browser's window or use the point-and-click editing facilities the GUI provides to edit the file. The user is presented with text input boxes, menus and checkboxes to select options as shown in Figures 3, 4, and 5.

From the main editor panel shown in Figure 3, the user can go onto the Machine Definition Lister, as shown in Figure 4. From there, the user can delete, create new, and edit existing machine definitions for the selected configuration file.

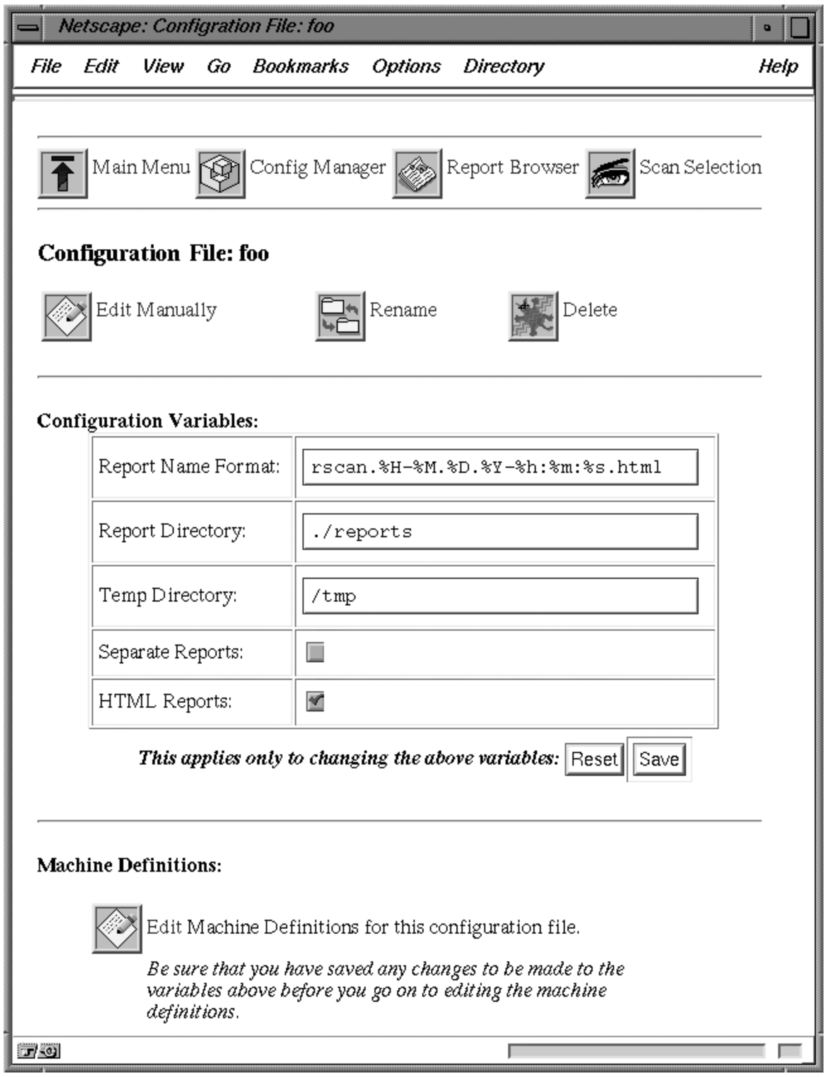


**Figure 3**: Initial GUI Configuration Editor

When the user chooses to edit a machine definition, they will see a panel like that in Figure 5. On this panel, the user can add hosts to the list of machines that the machine definition affects, edit the location of perl for the set of machines, and edit the modlist for the set of machines. When editing the modlist, the user is presented with a list of installed modules, and all they need to do is click on toggle buttons to select and de-select modules to run. The user can also choose to run all the scans in the module, only selected scans that are chosen from a scrolling list, or all scans in the module *except* the ones that they have chosen. When finished selecting scans to run and other options, the user simply needs to click on the "Save" button to have the machine definition saved to the configuration file.

**Modlists**

One of the most powerful features of Rscan is the concept of the modlist. A modlist (*MOD*ule *LIST*) is a way of specifying which modules will be run on specific hosts and which scans in a particular module will be run. It allows the user to be very picky about selecting what to run on any machine.

A modlist may be as simple as the word "`any`," which instructs Rscan to run all applicable modules and scans in those modules when scanning the remote machine. It can also specify that out of module `foo` we want to run only scans `x`, `y`, and `z` and that in module `bar` we want to run all scans *except* the `i`, `j` and `k` scans.

The mechanism for defining modlists is simple and straightforward, and a modlist can be applied to any number of hosts without the need of writing it many times (since they can become long and relatively complicated).

Module names are separated by commas, with no space in between anything. For instance, a modlist of "`ThisModule,ThatModule,TheOtherModule`" tells Rscan to run the `ThisModule` module, the `ThatModule` module and the `TheOtherModule` module.

Modlists can be more complicated and more powerful than this. Using the `-list` option to Rscan, users can get a list of modules that are installed and a list of the individual scans that compose each module. Using these lists, usrs can construct a custom subset of scans to run from each module.

Let's say that the `ThisModule` module had only the scans named `sendmail`, `rdist`, `rhosts`, and `xsession`. Then if a user only wanted to run the `sendmail` and `rhosts` scans out of that module, they could replace the `ThisModule` part of the previous modlist with the following:

```
ThisModule[sendmail:rhosts]
```

and only those scans would be run. The scanner names are the names of the `.pl` files without the `.pl` extension. Users can also get the same result if they use the negation operator ("`-`") in the modlists.

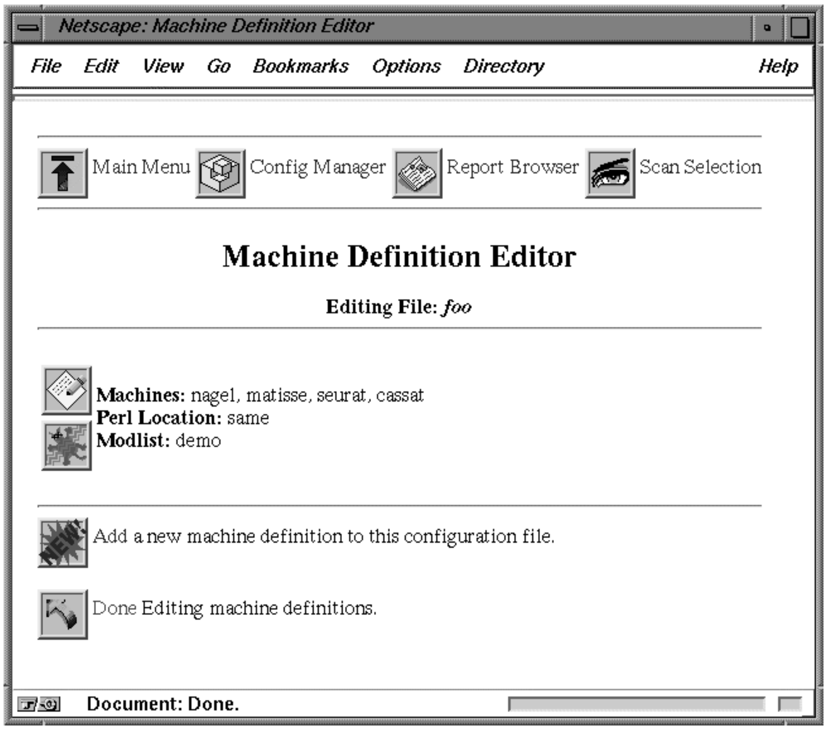


**Figure 4**: GUI Machine Definition Lister

Because in this example we assume there are only those four scans in the module, the above is equivalent to:

```
ThisModule[-rdist:-xsession]
```

This form of a modlists tells Rscan to run everything in the `ThisModule` module *except* the `rdist` and `xsession` scans.

If specifying scans in a modlist, be sure not to mix the two kinds when specifying what scans to run in a particular module. A modlist cannot contain both regular and negated scan selections. For instance, the modlist

```
ThisModule[-rdist:xsession]
```

is invalid, and would be caught by Rscan's modlist parser, but

```
ThisModule[-rdist:-xsession],
    TheOtherModule[thisscan:thatscan]
```

is valid because it does not mix the two kinds in a single module definition.

A modlist has no limit on length, though the user's shell may impose one when they are used on the command line. Users can specify as many modules and as many scans as they want, provided that the space character doesn't show up, and module names are separated with the ',' character and scan names with the ':' character. A modlist may need to be protected with "double quotes"

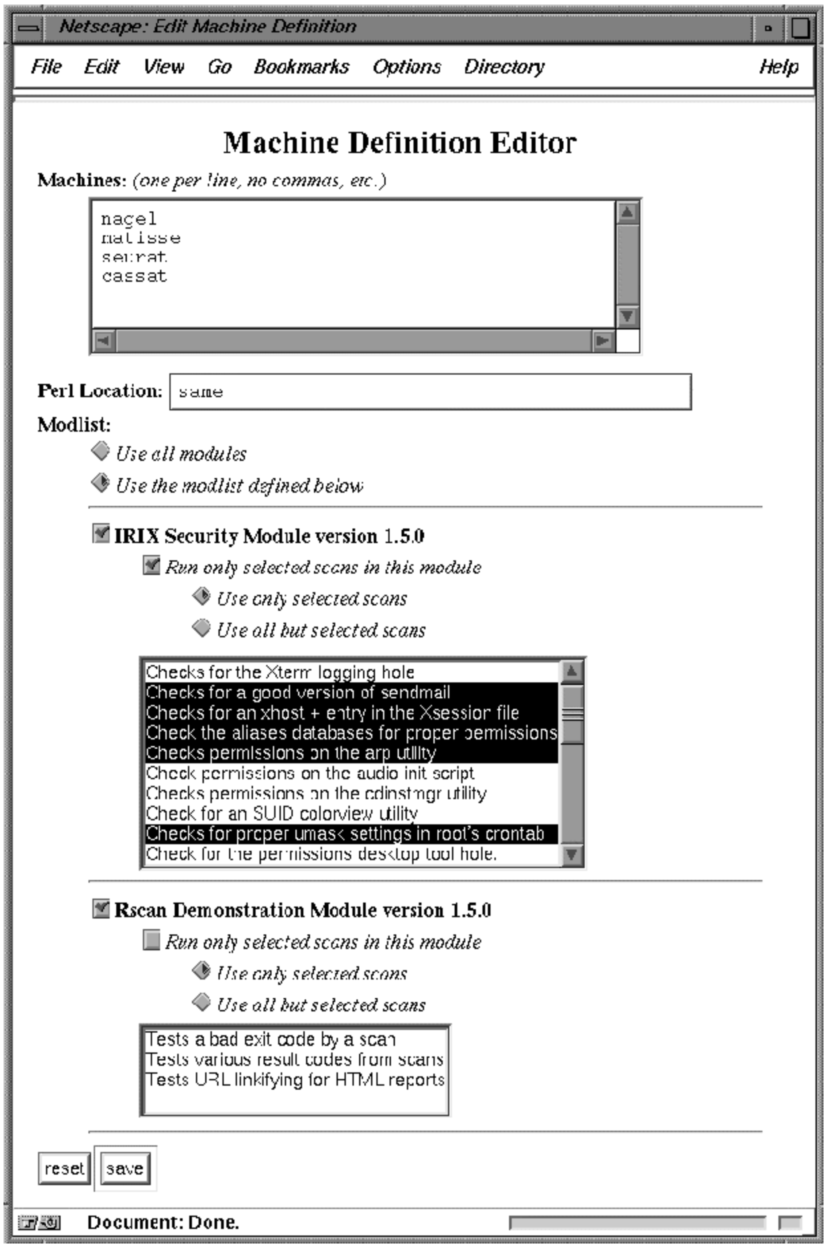


**Figure 5**: GUI Machine Definition Editor

when using them on the command line, since the '[', ']' and ':' characters are treated as meta-characters by some shells. They should *not* be protected when used in the configuration files.

### A Sample Run

As Rscan runs, it gives status messages about its activities to standard out. Such messages may notify the user that Rscan is copying the scanner and modules to the remote machine, or running its initialization scripts, etc. Figure 6 shows a sample run of Rscan in text mode and Figure 7 shows the same run output when run from within the GUI.

```
                    ____
                   |  _ \ ___  ___ __ _ _ __
                   | |_) / __|/ __/ _` | '_ \
                   |  _ <\__ \ (_| (_| | | | |
                   |_| \_\___/\___\__,_|_| |_|
          Multi-Platform Interrogation and Reporting

                         Version 1.5.0

Initializing
+======================================================================+

Copying scanner to cassat
Initiating a remote scan on cassat
Initializing

Rscan 1.5.0 starting scan on cassat
Date: Monday, June 5 1995 at 10:55:20 am

 Test                                                         Condition
-----------------------------------------------------------------------

Rscan Demonstration Module version 1.5.0
 Bad exit code ............................................. [ PASS ]
 Looking at API Variables .................................. [ PASS ]
 Exit with failed .......................................... [ FAIL ] *
 Exit with info ............................................ [ INFO ]
 Exit with passed .......................................... [ PASS ]
  .
  . [ data deleted for brevity ]
  .
 Exit with nothing ......................................... [ ERR  ]
 Exit with warning ......................................... [ WARN ] +
 Exit with nothing ......................................... [ ERR  ]
 Test URL linkify .......................................... [ PASS ]
 Exiting seurat

+======================================================================+

Please read the file
  ./reports/rscan.network-05.05.95-10:55:17.html
for a full scan report.

 It is in HTML format and should be read with a tool like
 Netscape Navigator, NCSA Mosaic or Lynx.
```

**Figure 6**: Sample text mode run output

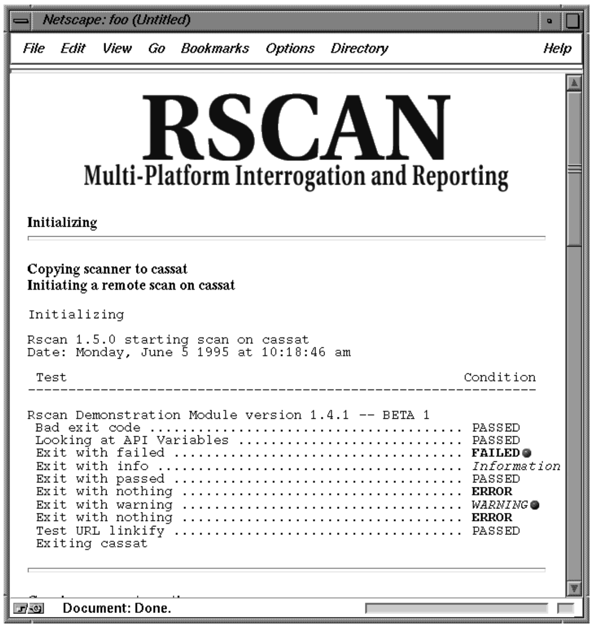


**Figure 7**: Sample GUI mode run output

If the report is written in HTML, then each scan's name is a link to the part of the report that more fully describes that scan's findings, and all text that resembles a URL is converted to a "real" URL. Figure 8 shows an ASCII report generated by the above run and Figure 9 shows an HTML report generated by the same run.

```
                  Rscan 1.5.0 Network Report

+====================================================================+

Report for cassat

The date is:             Monday, June 5 1995 at 10:57:16 am
Rscan Version:           1.5.0
Perl Version:            5.001
Scans:                   Rscan Demonstration Module version 1.5.0
Modlist:                 demo
Machine name:            cassat (cassat.VIS.ColoState.EDU)
Operating System:        IRIX 5.3
OS Patch Level:          11091812
CPU Type:                IP22 mips
System ID:               1762128434

 Test                                                   Condition
----------------------------------------------------------------

Rscan Demonstration Module
  Bad exit code ..................................... [ PASS ]
  Looking at API Variables .......................... [ PASS ]
  Exit with failed .................................. [ FAIL ] *
  Exit with info .................................... [ INFO ]
  Exit with passed .................................. [ PASS ]
  Exit with nothing ................................. [ ERR  ]
  Exit with warning ................................. [ WARN ] +
  Exit with nothing ................................. [ ERR  ]
  Test URL linkify .................................. [ PASS ]

----------------------------------------------------------------


Test Data for Rscan Demonstration Module

 Test: Bad exit code
     This scan will exit with code 42, as opposed to 0.
     Test Results: PASSED

      WARNING: scan "demo/badexit.pl"
        exited with status = 42.  There may have
        been an error in it's run. PID was 1069.
  .
  . [ data deleted for brevity ]
  .
 Test: Test URL linkify
     This is a url:
       ftp://ftp.vis.colostate.edu/pub/rscan
     and so is this:
       http://www.vis.colostate.edu/rscan
     and so is this:
       http://www.vis.colostate.edu/info/staff/nate
     Test Results: PASSED

+====================================================================+
```

**Figure 8**: Sample ASCII report

## Programming Rscan

Rscan can be used to provide integrity checks for a site or to automate any number of complicated or mundane tasks. This section describes how to write custom Rscan modules.

### The Rscan API

Rscan has a set of functions for writing to the screen, reports, and performing other common tasks. These functions should be used if applicable, and writers of modules are invited *not* to rewrite them, since it makes life hard for the writer of the module when a new version of Rscan comes out with changes.

### API Functions

**&pcheck(LIST)** The string formed by the joining of *LIST* is written to the check list. This function should be used to signal the current test that is being run. There can be more than one `&pcheck` call in a scan, but each one should have a corresponding call to one of the result functions (`&passed`, `&failed`, `&warning`, and `&info`) before the next call to `&pcheck` is made. If there are two calls to `&pcheck` without a result function call in between or if a scan exits with an unresolved call to `&pcheck`, Rscan catches this, reports it and inserts an error condition (with `&error`).

**&passed** The last test should be marked as passing.

**&failed** The last test should be marked as failing.

**&warn** The last test should be marked as a warning.

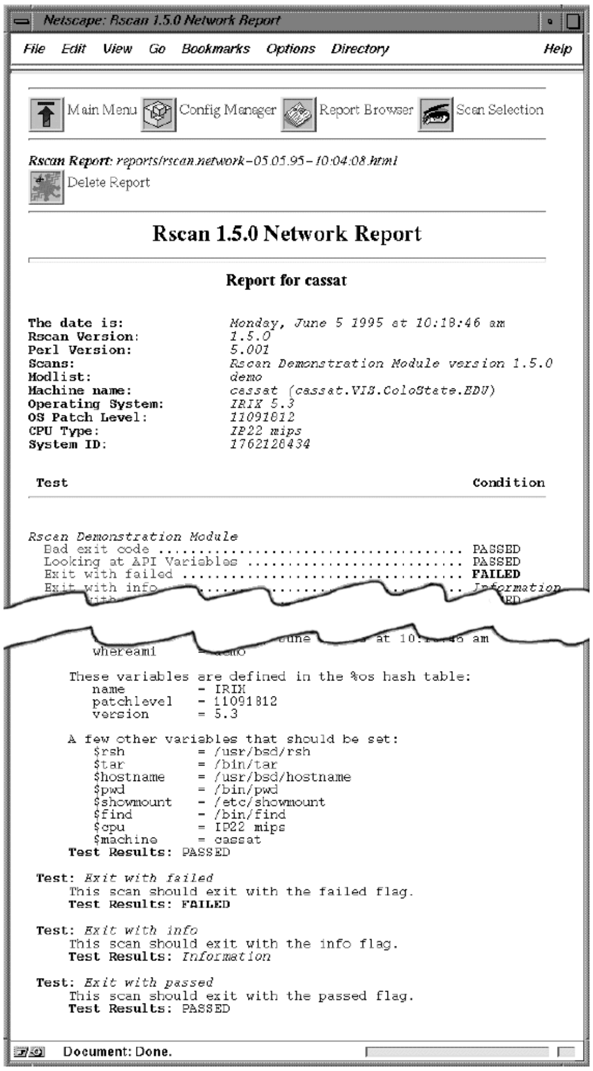


**Figure 9**: Sample HTML report

**&info** The last test should be marked as informational.

**&report(LIST)** The string formed by the joining of *LIST* is written to the report file. This function should be used to write messages to the report file. There is no need to write different things for an ASCII and an HTML report, since Rscan takes care of formatting automatically. If reports are being written in HTML, then any text printed using this function that resembles a URL is converted to a hypertext reference.

**&screen(LIST)** The string formed by the joining of *LIST* is written to the screen. This function should be used to write messages to the screen about what is happening. It is not recommended for use in scans, but primarily in the `init` files for a module.

**&center(LIST)** Returns the joining of *LIST* centered on a 65 character wide page.

**&nprint(space,text)** Returns *text* left justified in a field *space* characters wide, padded with spaces. This function is good for making formatted tables in reports, etc.

**&header(headername,Value1,...,ValueN)** Places a header field named *headername* in the report's header section for the current machine. The values *Value1 ... ValueN* are placed to its right such that they appear in the report's header as follows:

```
headername: Value1
            Value2
            ValueN
```

There is no limit to how many values can be specified. The text for the *headername* is printed using `&nprint` in a 25 space wide field, and successive text is indented 25 spaces. Like the `&screen` function, this is primarily meant to be used in the `init` files for a module.

**&rawheader(LIST)** The text formed by the joining of *LIST* printed *without any additional formatting* into the header section of the report. This could be used to place some text centered in the header, etc. Again, this function should only be used in the `init` files for a module.

### API Variables

Most API variables are collected in the `%api` associative array. Variables of particular interest are:

**$api{'whereami'}** This is the directory that the currently running script is located in. This seems pretty lame, but the script files move around depending on where the temp directory is located, etc. This variable should be used, for instance, when opening a database associated with a scan, and when doing other path oriented operations.

**$api{'scanmode'}** This is either set to `local` or `remote` depending on whether the scanner is running locally or remotely. The important difference between the two is that in local mode, scans are run *in place* and when run in remote mode they are run from within a temporary location and are deleted after the scanning run has been completed.

**$api{'scannerdir'}** This is set to the base directory that the scanner running in. It is set to something like `/tmp/scanner` or `/var/tmp/scanner`, depending on the `-temp` command line option and `tempdir` configuration file option.

**$api{'outformat'}** This is either set to `ascii` or `html` depending on whether the reports are being written as plain ASCII or HTML files. Again, because Rscan automatically formats output, writing different text for ASCII and HTML reports is discouraged, since Rscan will automatically format the output accordingly (this variable is here in case users want to do this anyway).

**$api{'perlversion'}** This is set to the version number of `perl` that is running the scan.

**$api{'now'}** This is set to the integer time (the number of seconds since January 1, 1970) that `perl` was started at.

**$api{'time'}** This is set to the human-readable time that looks something like "Friday, January 1, 1970 at 05:22:44 pm"

There is also a `%os` associative array that contains information about the operating system of the machine currently being scanned. Here are the `%os` variables, along with some other variables that pertain to the operating system and the machine in general.

**$os{'name'}** The name of the operating system. Something like `IRIX` or `SunOS`, as returned by `/bin/uname`.

**$os{'version'}** The version of the operating system. Something like `4.0.5` or `6.0.1`, as returned by `/bin/uname -r`.

**$os{'patchlevel'}** The operating system's patch level, as defined by `/bin/uname -v`.

**$machine** The short name of the machine, as defined by `/bin/uname -n`.

**$cpu** The machine's CPU type, as defined by `/bin/uname -m`.

**$rsh** The location of the `rsh` utility.

**$tar** The location of the `tar` utility.

**$tar_create** The command line options necessary to have `tar` create an archive sending its output to standard out to be used as input to a pipe.

**$tar_extract** The command line options necessary to have `tar` extract an archive non-verbosely.

**$gzip** The location of the `gzip` utility.

**$gzip_extract** Command line options to give `gzip` to extract an archive to standard out (so it can be piped to `tar`).

**$remove** How to delete files and directories. Usually `/bin/rm -rf`.

### Porting Rscan to a New OS

In general, Rscan should run on any UNIX that has either `perl` version 4 or 5 installed on it. There is one operating system specific file that has to be created for each new operating system that Rscan will be run on. This file is called the "pathconfig" file for the operating system, and is located in the `scanner/pathconfig` directory and must be named the same as the output of `/bin/uname` on that operating system.

This file should contain definitions for how to obtain the variables in the `%os` associative array (`name`, `version`, etc.) and the absolute paths to several necessary programs. New pathconfig files should be modeled after the others stored in the `pathconfig` directory. The really important options are the definition of where `rsh` and `tar` are located and how to extract a `tar` archive. If Rscan will be run in GUI mode under perl version 4 on the new operating system, then the pathconfig file must also include definitions for the `$AF_INET`, `$PF_INET` and `$SOCK_STREAM` variables associated with sockets (in perl 5, these variables are available from function calls, but in perl 4 they must be predefined). The best way to get a feel for what these scripts do is to read through a few of the other pathconfig files and give it a shot... the worst thing that can happen is that Rscan will not run the scans properly!

### Module Structure

Modules are organized by short name in the "`scanner/modules`" directory under the main Rscan directory. Figure 10 illustrates the structure of a typical module.

```
module/                               Main directory for the module
       init                           Generic initialization file
       fullname                       Module's "Full Name"
       version                        Module's version number
       desc                           descriptions of all *.pl files
                                      in this directory
       *.pl files                     individual scan files
       OSNAME1/
               init                   Initialization data for OSNAME1
               VERSION1/
                        desc          descriptions of all *.pl files
                        *.pl files    scans for OSNAME1 VERSION1
               VERSION2/
                        desc          descriptions of all *.pl files
                        *.pl files    scans for OSNAME1 VERSION2
               VERSIONn/
                        desc          descriptions of all *.pl files
                        *.pl files    scans for OSNAME1 VERSIONn
               common/
                     desc             descriptions of all *.pl files
                     *.pl files       scans for all versions of OSNAME1
       OSNAME2/
               init                   Initialization data for OSNAME2
               VERSION1/
                        desc          descriptions of all *.pl files
                        *.pl files    scans for OSNAME2 VERSION1
               VERSION2/
                        desc          descriptions of all *.pl files
                        *.pl files    scans for OSNAME2 VERSION2
               VERSIONn/
                        desc          descriptions of all *.pl files
                        *.pl files    scans for OSNAME2 VERSIONn
               common/
                     desc             descriptions of all *.pl files
                     *.pl files       scans for all versions of OSNAME2
       OSNAMEn/
               init                   Initialization data for OSNAMEn
               VERSION1/
                        desc          descriptions of all *.pl files
                        *.pl files    scans for OSNAMEn VERSION1
               VERSION2/
                        desc          descriptions of all *.pl files
                        *.pl files    scans for OSNAMEn VERSION2
               VERSIONn/
                        desc          descriptions of all *.pl files
                        *.pl files    scans for OSNAMEn VERSIONn
               common/
                     desc             descriptions of all *.pl files
                     *.pl files       scans for all versions of OSNAMEn
```

**Figure 10**: Example module directory structure

Module initialization scripts are called "`init`" and (if present) are located in the base directory for the module, and in each subdirectory therein named for an operating system (e.g., "`IRIX`," "`SunOS`," etc.). These scripts (if present) are run when a module is executed on a remote machine. First the `init` script in the main module directory is run, and then the `init` script in the directory named for the operating system of the machine that the scanner is running on is run. A file named "`fullname`" located in the base directory for the module contains a one line "long name" for a module. A file called "`version`" in the same directory contains the version number of the module.

In each directory that contains actual scanner scripts (including the main module directory), there is a file called "desc" which contains one line descriptions for what each scan in that directory does. Rscan uses these files when listing the modules that are installed. They make it easier to understand what a particular scan will be doing when its run. The format for the desc files is as follows:

```
scanner_script.pl
one-line desc. of "scanner_script.pl"
```

Blank lines and lines beginning with the "#" character are ignored.

**Running Order For Files**

When Rscan starts its run on a machine, files are always executed in the following order (if they are present):

1. The generic initialization file for the module is run. This file is located in the main module directory and called `init`.
2. The OS-specific initialization file for the module is run. This file is located in the directory named for the current operating system under the main module directory, and is also called `init`.
3. `.pl` files in the main module directory are run in alphabetic order.
4. `.pl` files in the *osname* `/common` directory under the main module directory are run in alphabetic order.
5. `.pl` files in the *osname* / *osversion* directory under the main module directory are run in alphabetic order.

As discussed previously, the modlist determines the exact set of `.pl` files which will be run on any particular machine, but they are always run in alphabetic order.

**Tips For Writing Effective Modules**

Writing modules for Rscan is very easy, but there are a few tips that will make writing them much easier and make running them go much smoother.

Instead of having many many small scripts, try to collect some of the scripts into one larger script. This is especially true if one script must be run before another script can do its job – which may not happen if the user decides not to run one of the scripts by not including it in a modlist.

Remember that since each script in each module is run in its own process, any variables that are set up in one script will *not* be available to any other script in the module. If there is a need for some variables to be set up for a particular operating system or for every operating system that a module will be run on, they should be set in one of the `init` scripts.

When there is a task that needs to be accomplished on all the different operating systems that the module will run on, but needs to be done in a slightly different way on each system, have a separate script for each operating system (and possibly one for each version of each operating system), but name each script with *the same file name*. This way, if the user wants to accomplish that task on a set of machines, the user simply needs to include that script name in the modlist for those machines and the correct script for that operating system will be run, since there is no way (other than multiple machine definitions) to set a script to be run only under some particular version of an operating system. For instance, let's say that someone is writing a module to check the configuration of each machine on their network and one of the checks is to make sure that the correct sendmail configuration file is present on each machine. Since sendmail configuration files can differ drastically from one operating system to another, the user should write a separate script for each operating system (and, if necessary, a separate one for each version of an operating system) and call each one, for instance, `checksendmail.pl`. Now, when the user wants to check the sendmail configuration files on all their machines they simply need to use a modlist like "`mymodule[checksendmail]`" and the proper sendmail configuration file check script will be run on each machine.

**A Programming Example**

Rscan modules can be trivially simple or extremely complicated. This is a step-by-step description of what is required to write a module that incorporates all of Rscan's major features.

Let's say that an administrator wants to run Tripwire on each machine on their network. The first thing to do is compile a copy of Tripwire for each architecture that the module will be run on. If there are different options needed to run on different versions of one or more of the operating systems in question, the administrator must build a binary for each of them.

Next, write a perl script for each of the binaries that were created above. These may all be identical unless there's some strange things that need to be done on certain operating systems. They could be as simple as just starting the process and with the right command line options for each architecture and using the `&report()` function to relay that information back to the server, like this:

```
open(PROC, "mytool -arg1 -arg2 |");
while (<PROC>) {
  &report($_);
}
close(PROC);
```

The script could also move around the Tripwire databases as needed before running Tripwire itself, make backups of the databases, etc. The scripts should all be named the same for reasons explained in the previous section.

Now that the wrapper scripts have been written, organize the module in the way shown in Figure 10. The module should now be ready to run. It's that simple. The only thing that needs to be kept track of now is the name of the module and the name of the scripts in the module. This process can easily be repeated for other common tools like Crack, TAMU, or other custom scripts or programs that are in use in an organization.

### Future Enhancements

The only real enhancement planned for Rscan is the use of secure communications channels between the server and the remote machine and between the HTTP servers and the WWW browser. For securing communications between the server and the remote machine, Netscape's Secure Socket Library is a possible solution, but the SSH (Secure Shell) Remote Login Program from Tatu Ylönen (*ylo@cs.hut.fi*) is probably better, since the best solution for securing this part of Rscan is to implement a secure version of `rsh` and use that, since it would make Rscan secure with no modifications at all. Securing the communication between the Rscan HTTP servers and the WWW browser would almost certainly require the use of the Netscape SSL library since Netscape Navigator is currently the only security-enabled WWW browser. Unfortunately, the use of any kind of secure communications usually conflicts with ITAR export regulations (because most encryption schemes are classified as weapons by the US Government and are therefore subject to the same export regulations as nuclear weapons), so a secure Rscan would probably not be exportable.

Suggestions for future enhancements (and bug fixes) are very welcome, and reasonable requests for new features are usually implemented. Requests should be sent directly to the author.

## Availability

Rscan is available via anonymous ftp to `ftp://ftp.vis.colostate.edu/pub/rscan`. The file `rscan.tar.gz` is always a link to the most recent copy of Rscan. The Rscan WWW homepage is located at `http://www.vis.colostate.edu/rscan`.

Beta releases of "in the works" copies of Rscan are placed in the `/pub/rscan/beta` directory. The archives in that directory should be considered pre-release, and are not recommended for use in a "production" environment. They are only there for evaluation and bug reports by those people who are kind enough to test the software.

There is also a mailing list that receives announcements about new versions of Rscan and about new modules. The list name is `rscan-announce@vis.colostate.edu` and may be subscribed to by sending mail to `majordomo@vis.colostate.edu` and including the text "`subscribe rscan-announce`" on a line by itself in the message body. The list is moderated by the author, and is extremely low traffic.

Rscan is completely free, but out of curiosity the author requests that users send mail regarding their use of Rscan. The author also requests that if a user hacks up the code and redistributes it they don't take full credit for writing it from scratch.

## Biography

Nathaniel (Nate) Sammons is currently a Computer Science student at Colorado State University where he works for Academic Computing and Networking Services in the Computer Visualization Laboratory. He has been the system administrator there for the past two years. Nate can be reached electronically at *nate@colostate.edu*. His WWW homepage is `http://www.vis.colostate.edu/info/staff/nate`.

## References

[1] Dan Farmer and Eugene H. Spafford. The COPS security checker system. *USENIX Conference Proceedings*, Pages 165-170, Anaheim, CA, Summer 1990.

[2] Larry Wall and Randal L. Schwartz. *Programming Perl*. O'Reilly & Associates, Inc. Sebastopol, CA, 1991. ISBN 0-937175-64-1.

[3] David R. Safford, Douglas Lee Schales, and David K. Hess. The TAMU Security Package: An Ongoing Response to Internet Intruders in an Academic Environment. *Proceedings of the Fourth Usenix UNIX Security Symposium*, Pages 91-118, Santa Clara, CA, October 1993.

[4] T. Berners-Lee, R. T. Fielding, and H. Frystyk Nielsen. Hypertext Transfer Protocol – HTTP/1.0, Third Edition. *IETF HTTP Working Group*. March 8, 1995.