

Description of the cPanel Security Policy Plugin System

Prepared for: cPanel, Inc. Partners & Customers
Prepared by: G. Wade Johnson, Linux Development

October 20, 2010

Document Revision 2

Table of Contents

Summary	1
What Is the cPanel Security Policy?	1
The 2 Parts of the cPanel/WHM 11.28 Security Policy	1
1. The cPanel-provided security policies	1
2. A Framework for Server Owners to Implement Custom Security Policies	1
About Security Policies	2
What Makes a Good Security Policy?	2
Security Policies in the Login Lifecycle	2
About Security Policy Plugins	4
Enabling a Security Policy Plugin	4
Security Policy Extensions	4
Makeup of a Security Policy Plugin	4
Security Policy Plugin Modules	5
Testing for Policy Violation	5
User Interaction	6
Configuration	6
User Interface Modules	7
HTML UI Type	8
Report	8
Redirect the User	8
Direct Resolution	8
Text UI Type	8
XML API UI Type	8
JsonApi UI Type	9
Relation Between check and run	9

Current Security Policies	9
Cpanel::SecurityPolicy::SourceIPCheck	9
Cpanel::SecurityPolicy::PasswordStrength	9
Cpanel::SecurityPolicy::PasswordAge	9
Input Parameters	10
appname	10
\$acctref	10
\$sec_ctxt	10
\$cpconf_ref	11
Conclusion	12

Summary

What Is the cPanel Security Policy?

A *security policy* describes an extension to the login process to control the strength of login security after user authentication has occurred.

In version 11.28, cPanel provides a new, 2-part feature that allows server owners to choose how to secure their cPanel servers.

The 2 Parts of the cPanel/WHM 11.28 Security Policy

The new security policy feature is divided into the following 2 parts:

1. The cPanel-provided security policies

The first part consists of the 3 security policies provided with cPanel/WHM version 11.28. These policies allow server owners to:

- Limit logins to verified IP addresses
- Specify a minimum password strength
- Specify a maximum password age

You can enable these 3 security policies on the WHM *Configure Security Policies* screen. While these policies are mentioned in this document (see page 9), they are not its overall focus.

These 3 security policies are part of a larger security policy framework new to version 11.28.

2. A Framework for Server Owners to Implement Custom Security Policies

In version 11.28, cPanel provides a plugin-based security policy system (or “framework”). Its purpose is to allow the root user to add or remove policies as desired, even while the system is running.

The individual policy modules are loaded and exercised through a defined API. This approach allows the root user to extend the supported policies in the future, without having to make major changes to the cPanel and WHM code.

This document focuses on this framework and how server owners can use it. In version 11.28, **the framework for the security policies is alpha quality and subject to change**. As a result of the changing nature of this framework, any security policies you implement under version 11.28 **will need to be updated later** if you upgrade cPanel/WHM.

About Security Policies

What Makes a Good Security Policy?

A security policy makes the login process more secure by increasing the restrictions on the login process. The policy can assume that the user has already completed the normal username/password login correctly (otherwise we would not know which user to test). The user will not be given access to the system until all of the policies have been checked successfully. In other words, a security policy can further *restrict* access, but cannot grant *more* access to the system.

A good security policy is relatively fast to check. It should implement some general security rule that in some way increases the security of the login. It should not duplicate functionality covered elsewhere in the cPanel system. An example of duplication would be a policy that prevents access to the system by a user named *bob*. That can be more easily handled by disabling or deleting *bob*'s account.

The initial cPanel security policies are good examples of general policies:

- **SecurityPolicy::PasswordAge** — Helps the user to change her password if the password is older than some configured number of days, and prevents access until this has occurred.
- **SecurityPolicy::PasswordStrength** — Helps the user to change her password if the password is too weak, and prevents access until this has occurred.
- **SecurityPolicy::SourceIPCheck** — Prevents access to the account from an unknown IP address unless the user can answer the specified security questions.

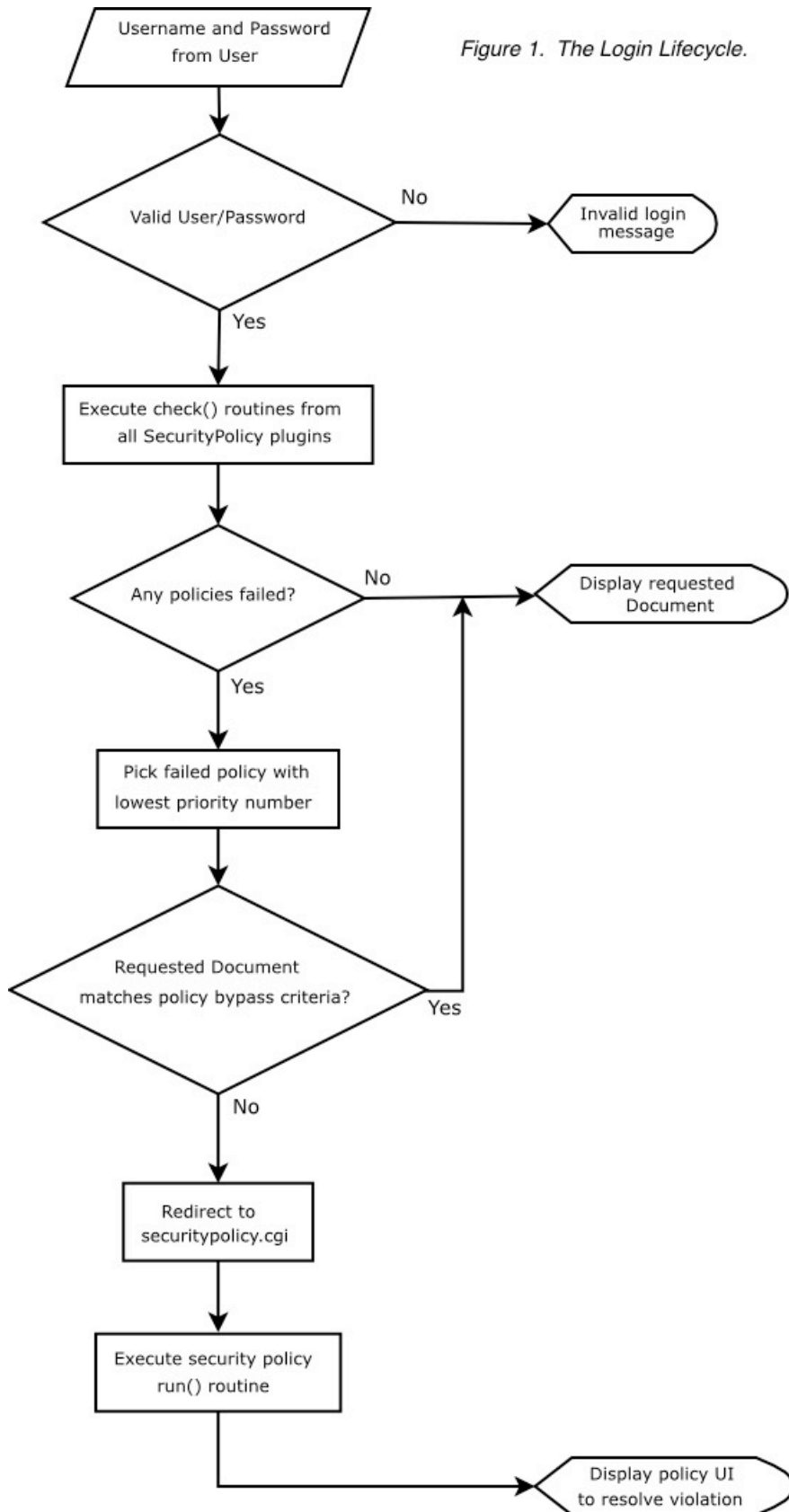
In each of these cases, the user is prevented access until the violation is resolved. Each policy limits the number of people who have access, or increases security by changing the user's behavior.

Security Policies in the Login Lifecycle

When a user attempts to log in, the system goes through a series of steps that are important to understand for security policies (see *Figure 1* below).

1. The user enters a username and password.
2. The username/password combination is validated. If this fails, the user is denied access.
3. Next, the system goes through the list of enabled security policies.
 1. If the *check* subroutine of all policies returns false, the user is granted access.
 2. Out of the policies that fail (*check* returns a priority), the one with the lowest priority number is chosen.
4. The current URL is tested using the *bypass* subroutine. If the routine returns true, the page is displayed.
5. Otherwise, the user is redirected to the *securitypolicy.cgi* script.

Figure 1. The Login Lifecycle.



About Security Policy Plugins

Enabling a Security Policy Plugin

There are 2 ways to enable a security policy. The correct way is to access the security policy configuration interface from the *Security Center > Configure Security Policies* menu option in WHM. This interface provides checkboxes for each security policy plugin found on the system.

Checking a box provides access to any configuration supported by the policy and enables the policy when the form is submitted. As part of the process of enabling or disabling policies, the *cpsrvd* service will restart. This restart is required in order to reload the security policy code.

As an alternative to the official interface, a policy may be enabled or disabled through modification of the */var/cpanel/cpanel.config* file. This approach is not recommended, but might be necessary under very rare circumstances. To use this method, you must edit the *cpanel.config* file as the root user. Find the entry for the policy in question. (For example, the entry *SecurityPolicy::PasswordAge* would control the *PasswordAge* security policy.) If the value for this item is 1, the security policy is enabled. If the value is 0, the security policy is disabled.

If you make changes to the *cpanel.config* file, you will need to restart *cpsrvd*.

Security Policy Extensions

Normally, the security policies are only applied to cPanel, WHM, and webmail access to the system. However, the *Configure Security Policies* interface in WHM allows the root user to apply the security policies to the DNS cluster interface, as well as XML API and JSON API requests.

These modes are not enabled by default, since they would usually be made as automated requests without any way for the user to resolve any violations.

Makeup of a Security Policy Plugin

A security policy plugin consists of Perl modules and templates living in specified locations. The Perl modules should reside in the *Cpanel::SecurityPolicy* namespace. The security policy system looks in the */usr/local/cpanel/Cpanel/SecurityPolicy* directory for the plugins.

The *Cpanel::SecurityPolicy* module implements the interface for the plugin-based security system. The individual policy modules are loaded and exercised through a defined API.

To minimize the memory overhead of the system, the code that executes the security policy is separated from the code that provides the UI portions of the policy. The separate portions of the interface are described in the *Security Policy Plugin Modules* section below.

Since the security policy code is run as root, it is important to be careful about the work done in the plugin.

Security Policy Plugin Modules

The security policy plugin is implemented as 2 or more Perl modules, each with specified functionality. This functionality is partitioned into 3 groups:

- Testing Policy Condition
- Interaction
- Configuration

The first piece of functionality is found in the module `Cpanel::SecurityPolicy::PolicyName`, where *PolicyName* is replaced with the name of the appropriate policy.

The other 2 pieces of functionality are found in the modules `Cpanel::SecurityPolicy::PolicyName::UI::UIType`, where *PolicyName* is replaced with the name of the appropriate policy and *UIType* is the appropriate user interface type (HTML, Text, JsonApi, or XMLAPI). Only the `config` subroutine in the HTML UI file is used for configuring the policy.

For example, the Password Age security policy is implemented in the modules:

- `Cpanel::SecurityPolicy::PasswordAge`
- `Cpanel::SecurityPolicy::PasswordAge::UI::HTML`
- `Cpanel::SecurityPolicy::PasswordAge::UI::Text`
- `Cpanel::SecurityPolicy::PasswordAge::UI::XMLAPI`
- `Cpanel::SecurityPolicy::PasswordAge::UI::JsonApi`

Not all security policies require the `Text` and `XMLAPI` UI modules.

Testing for Policy Violation

The module `Cpanel::SecurityPolicy::PolicyName` is loaded if the policy named *PolicyName* is enabled. This module must provide one public `check` subroutine and may also provide an optional `bypass` subroutine:

check(\$acctref, \$sec_ctxt, \$cpconf_ref, \$cookie_ref)

This subroutine takes a set of parameters and returns 0 if the policy has been satisfied or a priority number if the user failed the policy check. This priority number is used to determine which corrective action is taken first if more than 1 policy fails. If 2 or more policies fail, the `run()` subroutines for the policy with the smallest priority number are executed.

The parameters to the subroutine are as follows:

- `$acctref` — A reference to a hash of account information for the current user.
- `$sec_ctxt` — A reference to a hash of information about the current security context.
- `$cpconf_ref` — A reference to a hash of cPanel configuration information.
- `$cookie_ref` — A reference to a hash containing the user's current cookies.

The `check` subroutine does not directly communicate with the `run` subroutine. These two are normally run in separate processes.

bypass(\$sec_ctxt, \$cpconf_ref)

This subroutine checks the path in the document value of the `$sec_ctxt` parameter to determine whether this security policy should allow this page access. The subroutine returns a true value if the page (and current context) should skip the security policy logic, and a false value otherwise.



This subroutine allows the security policy plugin to specify pages that must remain accessible. If the security policy violation redirects to a page where the user performs actions to resolve this violation, the bypass would allow the user to access this page without another security policy violation. Without a bypass, the user would be prevented from getting to that page by the security policy.

Good examples include the *PasswordAge* and *PasswordStrength* security policies, which redirect the user to the *Change Password* screen. These policies provide a *bypass* subroutine that allows the user to reach the pages associated with changing the password.

The parameters to the subroutine are as follows:

- \$sec_ctxt* — A reference to a hash of information about the current security context.
- \$cpconf_ref* — A reference to a hash of cPanel configuration information.

User Interaction

A different UI module is loaded depending on which interface is being used to access the system. The supported UI types are *HTML*, *Text*, *JsonApi*, and *XMLAPI*. Of these, only *HTML* is required. The *Text*, *JsonApi*, and *XMLAPI* UI Types all have default handlers for the UI supplied by *Cpanel::SecurityPolicy*.

The UI module is loaded if the security policy check determines that the policy check failed. For example, *Cpanel::SecurityPolicy::PasswordAge::UI::HTML* is loaded if the *PasswordAge* policy is enabled, and its check failed. Each UI module must provide the following subroutine:

```
run( $acctref, $formref, $cpresultref, $sec_ctxt, $cpconf_ref, $cookie_ref )
```

This subroutine takes a set of parameters and provides the user interface needed to alert the user to the situation and potentially provide the ability for the user to resolve the issue. This subroutine may be called again (depending on the resolution method) after a form submission to attempt to actually resolve the issue.

The parameters to the subroutine are as follows:

- \$acctref* — A reference to a hash of account information for the current user.
- \$formref* — A reference to a hash containing form parameters that may be used in resolving the security issue.
- \$cpresultref* — A reference to a hash in which the subroutine may place status or error messages, or other information that must be passed back to the calling code.
- \$sec_ctxt* — A reference to a hash of information about the current security context.
- \$cpconf_ref* — A reference to the hash of cPanel configuration information.
- \$cookie_ref* — A reference to a hash containing the user's current cookies.

Configuration

The module *Cpanel::SecurityPolicy::PolicyName::UI::HTML* may also be loaded, if *PolicyName* is enabled to configure the parameters for the policy. For configuration, the UI module can provide one optional subroutine, as follows:

```
config( $formref, $cpconf_ref, $is_save )
```

This subroutine returns information describing the form entries that are needed to configure this policy. If the *\$is_save* parameter is true, the appropriate values from *\$form_ref* are placed in the *\$cpconf_ref* hash.

The return value from this subroutine is a hash reference containing the following data:

header — The value associated with this key is a string properly prepared for writing to the UI. This string is a description of the set of parameters this policy configures.

error — This optional key is used to report an error message if the parameters to save are invalid in some way. This string should already be localized before it is returned.

fields — The value for this key is an array reference containing a set of hash references which describe the individual data fields. Each of these hashes contains the following:

label — A printable label describing this parameter in user terms.

id — The string to be used as the key in *\$formref* when a value for this parameter is returned.

value — The current value for this parameter.

minval — The minimum value allowed for numeric values. (Optional.)

maxval — The maximum value allowed for numeric values. (Optional.)

type — An optional key specifying the type of control. If not specified, the field is a simple text input. If the type has the value *slider*, a slider control is used with a range between *minval* and *maxval*.

The parameters for this subroutine are as follows:

\$formref — This hash reference contains the values returned from the configuration form.

\$cpconf_ref — The hash reference containing the cPanel configuration data.

\$is_save — This boolean parameter is true if we have data from the form to put in the configuration hash. It is false if we have not accessed the form.

User Interface Modules

The User Interface modules handle the interaction with the user. There are 3 different modules, depending on the request type. Normal HTML requests and the configuration interface are both handled by the *Cpanel::SecurityPolicy::PolicyName::UI::HTML* module (where *PolicyName* is replaced by the name of the appropriate policy).

The *dnsadmin* requests are handled by the *Cpanel::SecurityPolicy::PolicyName::UI::Text* module. Any XML requests are handled by the *Cpanel::SecurityPolicy::PolicyName::UI::XMLAPI* module.

All of the UI modules support a *run* subroutine. On the other hand, only the *HTML* module may have a *config* subroutine.

The *run* routine generates the appropriate interface to help the user resolve the violation. At a minimum, the *run* routine should return a message for the user. If possible, the routine should either redirect to a page that helps resolve the violation, or present the needed pages to help the user resolve it.

HTML UI Type

The HTML User Interface module provides the security policy UI for any HTML request that violates the associated policy. The HTML UI module is the only one of the three UI modules that can reasonably be used for interactive resolution of the violation.

There are 3 ways that the *run* routine can report or resolve the security policy violation:

- Just report the error to the user.
- Redirect the user to another page where the user resolves the violation.
- Directly help the user resolve the violation through HTML forms.

Report

The simplest approach to handling the violation is to not attempt any resolution. In this scenario, the *run* routine would just supply an error page explaining the situation to the user and not allow further access. This would be useful for a violation that the user cannot resolve. For example, if the security policy was related to the time of day, the user would just have to wait.

At present, no security policies follow this approach.

Redirect the User

If there is already a page in the system which allows the user to resolve the violation, the best approach would be to send the user there. The *run* subroutine can perform an HTTP redirect to the appropriate page. The security policy's *bypass* routine must return true for the redirected page (or pages). Otherwise, the security policy will be triggered again.

This approach is used by the *Cpanel::SecurityPolicy::PasswordAge* and *Cpanel::SecurityPolicy::PasswordStrength* policies. They use the *main::force_redirect* subroutine to perform the redirection.

Direct Resolution

For security policies that require special resolution, the *run* subroutine can supply all of the HTML and form-handling logic to resolve the violation.

To allow flexibility with respect to branding and such, we would probably want to use templates to define the HTML. This method can support multiple pages guiding the user through any necessary steps. Unlike the previous method, it needs no *bypass* support, because the *securitypolicy.cgi* program is handling the whole process.

The *Cpanel::SecurityPolicy::SourceIPCheck* policy uses this approach.

Text UI Type

The Text User Interface module provides an error return for any *dnsadmin* request that violates a security policy. These requests were designed for a simple text response. Since the text interface has no way to interact with the user, this effectively becomes a simple error message.

Because this interface can be so simple, a default interface is provided if no **::UI::Text* module exists for the security policy. The only effect of the default handler is to send a message that the security policy check failed.

XML API UI Type

The XML API User Interface module provides an error return for any XML API request that violates a security policy. This interface should return an XML output that reports the problem.

The default behavior for this UI, if one is not provided by the security policy, is a simple *cpanelresult* XML document containing a *result* of 0 and a *reason* stating that the check failed.

JsonApi UI Type

The JsonApi User Interface module provides an error return for any AJAX (JSON API) request that violates a security policy. The default behavior, if one is not provided by the security policy, is to return a simple JSON object containing two properties: 1) a *type* property with a value of *text*, and 2) a *data* property containing an object with a *reason* stating that the check failed a *result* (which has a value of 0).

For example:

```
{
  type: "text",
  data: {
    result: 0,
    reason: "PasswordAge check failed"
  }
}
```

Relation Between *check* and *run*

There is no communication between the *check* and *run* subroutines. The *check* routine examines the environment to determine whether or not a violation has occurred. The *run* routine either displays a message or works toward resolving the violation. But, the *run* routine does not actually communicate with the *check* routine. It is important that the *run* routine actually resolve the same violation that *check* tests.

Current Security Policies

As detailed in above, are 3 currently defined security policies.

Cpanel::SecurityPolicy::SourceIPCheck

This policy ensures that a user can only get access to the service if the request comes from a known IP address. If the IP address is not known, the policy presents security questions for the user to answer. These questions and their answers are created by the user.

The priority number for this policy is 20.

Cpanel::SecurityPolicy::PasswordStrength

This policy ensures that the user's password is at least a certain strength. If the user's password is too weak, the user is redirected to the *Change Password* screen. The required strength of the password is configured by the root user.

The priority number for this policy is 9997.

Cpanel::SecurityPolicy::PasswordAge

This policy ensures that the user has changed her password recently. If the user's password has not been changed in too many days, the user is redirected to the *Change Password* screen. The maximum password age is configured by the root user.

The priority number for this policy is 9998.

Input Parameters

This section describes in more detail some of the input parameters to the security policy subroutines.

appname

This parameter indicates which interface the user was accessing when the security policy was triggered. This is useful for limiting the effect of a policy to a subset of the cPanel programs. Legal values are:

- *cpaneld* — cPanel interface
- *whostmgrd* — WHM interface
- *webmaild* — WebMail interface

\$acctref

This parameter is a reference to a hash containing information identifying the user. At present, these are the fields in this hash:

- user* — The username supplied by the user during login.
- virtualuser* — The user portion of the email address for webmail requests.
- webmailowner* — The system user who owns the webmail account.
- homedir* — The home directory of the system user.

\$sec_ctxt

This parameter is a reference to a hash of information about the current security context.

- appname* — The name of the application performing the security check.
- document* — The path of the URL that is being requested.
- is_posessed* — This flag is true if a privileged user is logging into the current account. If it is false, the actual account user is logging in.
- possessor* — This contains “root,” or the name of the reseller that owns the account.
- possessor_homedir* — The home directory of the possessor.
- possessor_pass_change_time* — The last time the possessor changed the password.
- remoteip* — The remote IP from which the user logged in.
- pass_change_time* — The last time the user changed the password.
- webmailowner* — The system user who owns the webmail account.
- virtualuser* — The user portion of the email address for webmail requests.
- auth_by_accesshash* — This flag is true if the user logged in with an access hash instead of through a username and password.
- pwstrength* — The required password strength.
- homedir* — The home directory of the system user for this user request.



request_type — This is an indication of the type of request. The potential types are:

- normal
- dnsadmin
- json
- xml

`$cpconf_ref`

This is a reference to a hash that contains the cPanel configuration information from the `/var/cpanel/cpanel.config` file.

Conclusion

The security policy architecture allows a server owner to increase the security of WHM and cPanel, without making major changes to the core source. It's possible for the system owner to tailor the security of the system as needed, provided the need for higher security.