mod_antimalware: a novel apache module for containing web-based malware infections

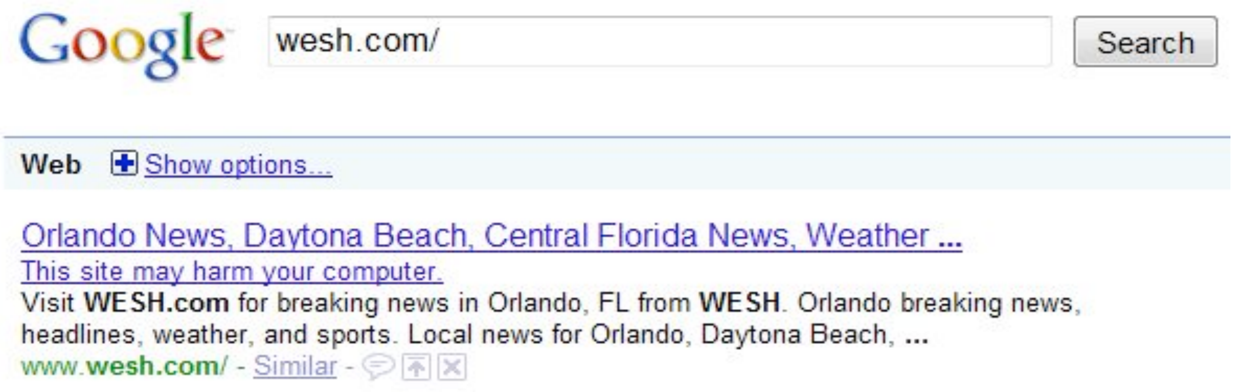
Neil Daswani, Pete Fritchman,
Ameet Ranadive, Shariq Rizvi, Ravi Reddy
Dasient, Inc.
{neil, petef, ameetr, shariq, ravi}@dasient.com

Abstract

Drive-by downloads planted on legitimate sites (e.g., via "structural" and other vulnerabilities in web applications) cause web sites to get blacklisted by Google, Yahoo, and other search engines and browsers. In this paper, we describe the technical architecture and implementation of mod_antimalware, a novel, open-source containment technology for web servers that can be used to 1) quarantine web-based malware infections before they impact users, 2) allow web pages to safely be served even while a site is infected, and 3) give webmasters time to recover from an attack before their web sites get blacklisted by popular search engines and browsers.

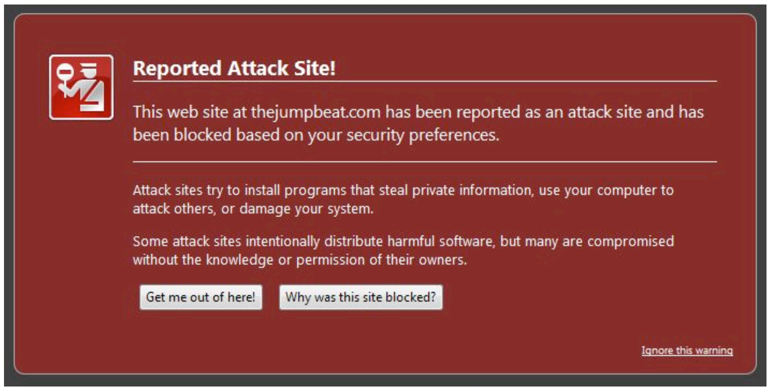
1.0 Introduction

Over the past few years, attackers have been increasingly using compromised, legitimate web sites as delivery vehicles for malware. In a typical attack, instead of simply defacing a web page as attackers used to in the early 2000s, they simply "infect" web pages by injecting malicious code that will result in a drive-by-download when the page is loaded by a user's browser. When search engines discover such infected pages, they "blacklist" the web pages (and/or entire web sites), add annotations to their search results warning users that the web site may harm their computers (see Figure 1), and distribute lists of compromised sites to popular web browsers such as IE, Firefox, Chrome, and Safari. Browsers display "red screens of death" (see Figure 2) instead of rendering infected web pages, which helps protect users, but has a very significant negative impact for web-based businesses when such attacks occur. For example, e-commerce sites can no longer sell when their product catalogs are infected, and publishers lose their advertising revenue streams when search engine traffic disappears and/or pages not rendered by browsers due to infections. Blacklisting typically results in web sites losing large portions of their web site traffic -- not only while the infection is present, but even after the infection is cleaned, as webmasters find difficulty in recovering their original levels of traffic due to loss of trust and brand damage.



Orlando News, Daytona Beach, Central Florida News, Weather ...
This site may harm your computer.
Visit WESH.com for breaking news in Orlando, FL from WESH. Orlando breaking news, headlines, weather, and sports. Local news for Orlando, Daytona Beach, ...
www.wesh.com/ - Similar -

Figure 1. Google's "This site may harm your computer" warning.



Reported Attack Site!

This web site at thejumpbeat.com has been reported as an attack site and has been blocked based on your security preferences.

Attack sites try to install programs that steal private information, use your computer to attack others, or damage your system.

Some attack sites intentionally distribute harmful software, but many are compromised without the knowledge or permission of their owners.

Figure 2. The Firefox "Reported Attack Site!" warning.

The current model of mitigating infections by curtailing search engine traffic and blocking pages is a necessary defense if we are to prevent malware from more rampantly spreading over the Internet, but can also be said to be a bit "draconian." After all, when your PC gets infected with a virus, anti-virus software does not put up a "red screen of death" and tell you that you cannot use your computer[1]. Rather, anti-virus software attempts to quarantine and/or clean the infection by moving and/or deleting infected files, but does indeed allow the user to continue to use his or her PC. On the other hand, when a web site is infected today, the web site becomes unusable to large numbers of users.

What is needed is a solution that brings the complete "lifecycle" of malware protection to web sites. In particular, when a web site becomes infected, we need to quarantine the infected web pages, but allow the web site to continue to be usable (to whatever extent possible). Mod_antimalware, the main topic of this paper, is a novel web server module which does exactly that. Once installed, mod_antimalware allows a web server to continue serving web pages in a safe manner even when some or all of the pages on the site have become infected.

Note that mod_antimalware is expected to be useful in cases where application-layer attacks are mounted. (Application-layer attacks account for over 75% of attacks on the web[2].) For instance, in the case that an attacker injects malicious code into a web application such as WordPress, Joomla, or Drupal, mod_antimalware should successfully block or filter the infections. However, if the attacker has compromised the operating system of the web server, it is likely that "all bets are off" and that mod_antimalware may not be a sufficient defense mechanism as the attacker could simply disable mod_antimalware, make arbitrary changes to the web server configuration, or even install and run alternate web server processes of the attacker's choice.

2.0 Usage

Mod_antimalware operates in conjunction with a website malware monitoring service that regularly scans a web site for web-based malware infection. When infections are found, information about the infections are sent to mod_antimalware such that it can "quarantine" the infection. The quarantining allows users to continue to access web pages on the server in a safe manner, even if some of the pages on the web site may be infected. In addition, the benefit for webmasters is that once web pages are quarantined, their web site is no longer at risk of being blacklisted and losing traffic, trust, and brand.

Mod_antimalware supports multiple forms of quarantining. In its most basic mode of operation, mod_antimalware can block infected web pages from being served altogether and allows webmasters to choose messaging of their own choice to display to users in the case of an infection. The open-source version of mod_antimalware, called mod_antimalware_lite, supports such a basic mode of operation. Another more advanced mode of operation is one in which mod_antimalware continues to serve the page, but automatically strips out any malicious code on the page in real-time. In the remainder of this

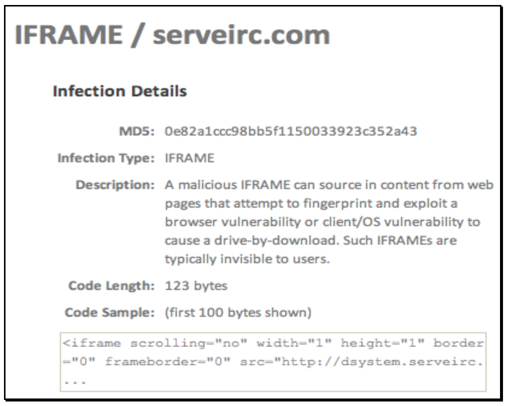
[1] Note, however, that some viruses can result in a blue screen of death which may effectively result in a user not being able to use his or her PC.
[2] https://www.cenzic.com/resources/reg-required/videos/Gartner-on-Web-Application-Security/

paper, while the operation of mod_antimalware and mod_antimalware_lite is illustrated using Dasient's website monitoring service, mod_antimalware_lite uses an open protocol to communicate with the monitoring service and can be used with another malware monitoring service. In the remainder of this paper, any statements made regarding mod_antimalware_lite are also true about mod_antimalware, as mod_antimalware supports a superset of functionality. Mod_antimalware's additional functionality is provided by an extended set of configuration directives.

The user experience for mod_antimalware works as illustrated in the following example, motivated by a true story in which Mr. Rogers web site was infected in early 2009, and a Dasient team member helped disinfect the site. Mr. Rogers is a famous children's show that runs on public television in the United States. Mr. Rogers was a well-known educator and minister, and it just goes to show that the attackers have no shame as their automated attacks infect even the most innocent of audiences.

In real web-based malware infections, attackers inject malicious code into web pages using a variety of means, including compromising: 1) web applications that have software implementation vulnerabilities such as SQL Injection or Cross-Domain vulnerabilities, 2) third-party widgets and ad networks, 3) web site FTP passwords, and 4) hosting platforms. In each of these cases, the attacker injects malicious code, often in the form of IFRAMEs and JavaScripts. When loaded as part of a web page by an unsuspecting user, the malicious IFRAMEs and JavaScripts fingerprint what version of operating system, browser, and third-party browser plug-ins are in use on the client machine. The attacker maintains a database of vulnerabilities and exploit code for each of the browser, plug-in, and operating system versions, and their scripts automatically serve optimal exploit code to take control of the users machine. Once the attacker has control of the user's machine, malware is sent. See Figure 3 for some examples of malicious IFRAME and JavaScript code that attackers inject into web pages. For more information about how such web-based malware attacks work, see Google's papers on "The Ghost in the Browser" and "All your IFRAMEs belong to us" by Niels Provos et. al.

In our Mr. Rogers example that we use to illustrate the operation of mod_antimalware below, we set up a mock version of the Mr. Rogers web site and "infected" it by adding a "malicious" IFRAME that serves an image with skeletons, as shown in Figure 4. In a real infection, of course, the IFRAME would not be visible (e.g., would have a width/height=0 or a "visibility: hidden" style attribute). The Mr. Rogers web site is monitored by Dasient's behavioral-analysis-based malware monitoring service, and has mod_antimalware installed on the web site. (More information about Dasient's behavioral-analysis-based malware monitoring service is available at https://wam.dasient.com/wam/info? prod=18)



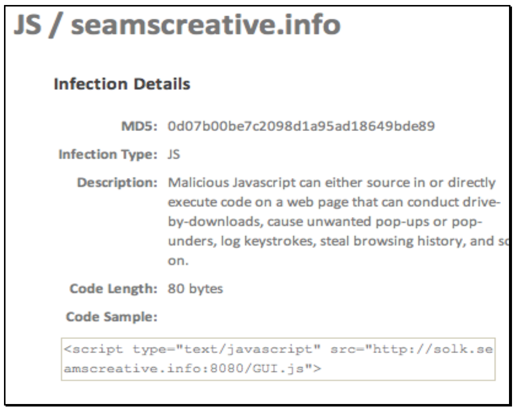
IFRAME / serveirc.com

Infection Details

MD5:	0e82a1ccc98bb5f1150033923c352a43
Infection Type:	IFRAME
Description:	A malicious IFRAME can source in content from web pages that attempt to fingerprint and exploit a browser vulnerability or client/OS vulnerability to cause a drive-by-download. Such IFRAMEs are typically invisible to users.
Code Length:	123 bytes
Code Sample:	(first 100 bytes shown)

```
<iframe scrolling="no" width="1" height="1" border
="0" frameborder="0" src="http://dsystem.serveirc.
...
```

```
<iframe scrolling="no" width="1"
height="1" border="0"
frameborder="0"
src="http://dsystem.serveirc.
...
```



JS / seamscreative.info

Infection Details

MD5:	0d07b00be7c2098d1a95ad18649bde89
Infection Type:	JS
Description:	Malicious Javascript can either source in or directly execute code on a web page that can conduct drive-by-downloads, cause unwanted pop-ups or pop-unders, log keystrokes, steal browsing history, and s on.
Code Length:	80 bytes
Code Sample:	

```
<script type="text/javascript" src="http://solk.se
amscreative.info:8080/GUI.js">
```

```
<script
type="text/javascript" src="http://
solk.se
amscreative.info:8080/GUI.js">
```

Figure 3. Example Malicious IFRAMEs and JavaScript, courtesy of Dasient's Infection Library, http://wam.dasient.com/wam/infection_library_index. Dasient's infection library stores snippets of code that attackers inject into web sites to infect them.

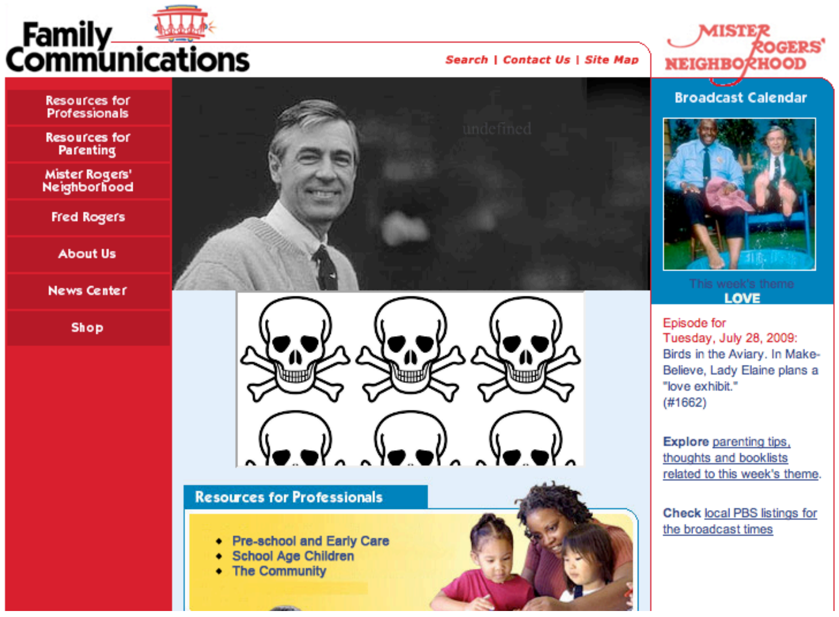
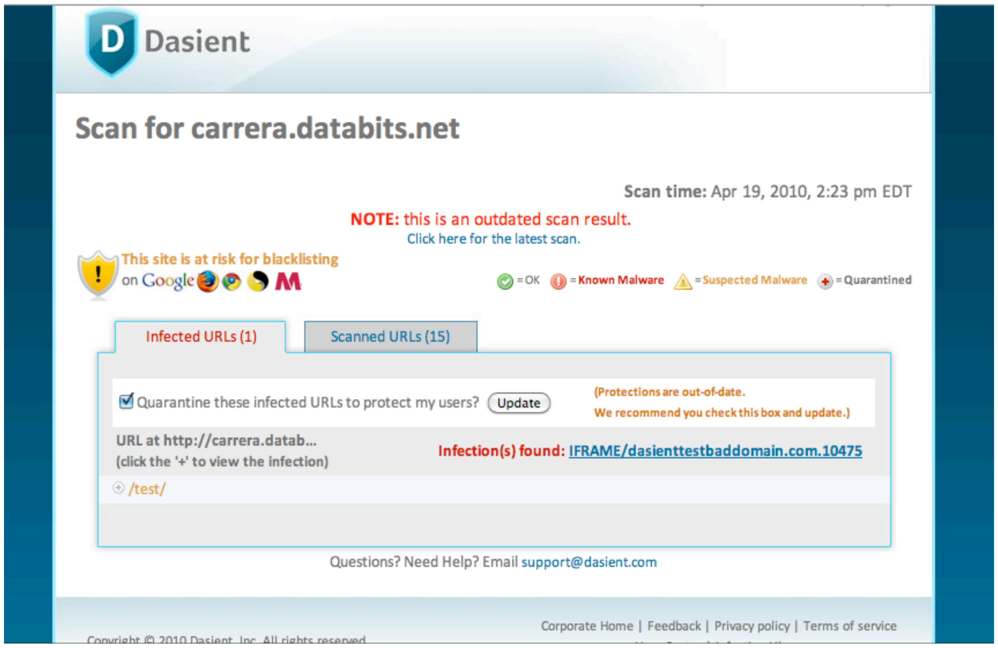

Family Communications
Search | Contact Us | Site Map
Mister Rogers' Neighborhood
Resources for Professionals
Resources for Parenting
Mister Rogers' Neighborhood
Fred Rogers
About Us
News Center
Shop
Broadcast Calendar
This week's theme
LOVE
Episode for Tuesday, July 28, 2009: Birds in the Aviary. In Make-Believe, Lady Elaine plans a "love exhibit." (#1662)
Explore parenting tips, thoughts and booklists related to this week's theme.
Check local PBS listings for the broadcast times
Resources for Professionals
Pre-school and Early Care
School Age Children
The Community

Figure 4. "Infected" Mr. Rogers web site. The infection is depicted by an IFRAME that renders an image with skulls. In a real infection, the IFRAME would be invisible to the user, and would serve a drive-by-download instead of an image.

Once the malware monitoring service detects that the Mr. Rogers site has been infected, it sends the web site's security team and/or webmaster an email alert about the infection. The email alert contains a link to Dasient's web-based malware interface which gives the user the option to enable quarantining of the infected web pages (see Figure 5). Upon checking the "quarantine these infected URLs" checkbox and clicking the "Update" button (as also shown in Figure 5), the malware monitoring service authenticates itself to the mod_antimalware instance running on Mr. Rogers' web site and sends quarantining directives. Once the quarantining directives are received and applied by mod_antimalware, subsequent users that access the home page are served the web page, but the malicious IFRAME is automatically filtered out in real-time, as shown in Figure 6.

Mod_antimalware's quarantining functionality allows the web site to continue to be used 1) without putting users at risk of being infected by web-based malware, and 2) without putting the web site at risk of being blacklisted by popular search engines or browsers. While mod_antimalware effectively "contains" the infection, however, the underlying infection may still be present in file on disk, or in a database that is used to render content on the web site, and the infection does need to be removed. If the infection is left untreated, it is likely that the web site would get blacklisted and start losing traffic, at which point the web site's technical team would be working under an emergency in which they would have to determine which parts of the web site are infected, what malicious code was injected, what application-layer vulnerability was exploited, how to patch it, and potentially how to appeal the blacklisting. With the mod_antimalware quarantining directives in place, the infection is actively and automatically "contained," such that the web site's technical team does not need to address the various problems associated with the infection under pressure in "code red" mode. Once the underlying infection is removed, the quarantine can be "lifted" and the web site can be re-scanned.



Scan for carrera.databits.net

Scan time: Apr 19, 2010, 2:23 pm EDT

NOTE: this is an outdated scan result.
Click here for the latest scan.

This site is at risk for blacklisting on Google

✓ = OK ● = Known Malware ⚠ = Suspected Malware ⊕ = Quarantined

Infected URLs (1) | Scanned URLs (15)

☑ Quarantine these infected URLs to protect my users? Update (Protections are out-of-date. We recommend you check this box and update.)

URL at http://carrera.datab...
(click the '+' to view the infection)

Infection(s) found: IFRAME/dasienttestbaddomain.com.10475

⊕ /test/

Questions? Need Help? Email support@dasient.com

Corporate Home | Feedback | Privacy policy | Terms of service

Copyright © 2010 Dasient, Inc. All rights reserved.

Figure 5. Enabling Quarantining.

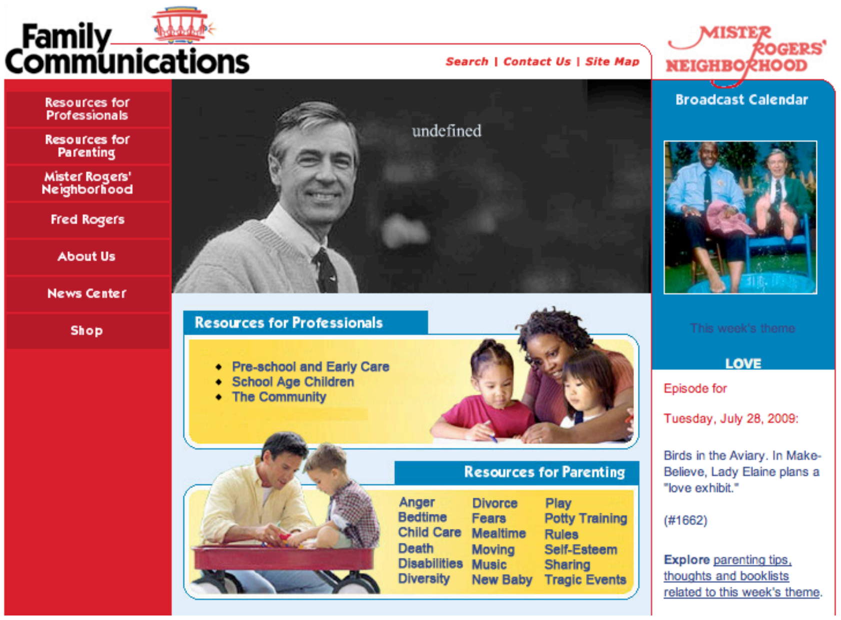


Figure 6. Quarantined version of infected Mr. Rogers home page

3.0 Architecture

Mod_antimalware works in conjunction with a website malware monitoring service that regularly scans a web site. The malware monitoring service delivers quarantining directives to mod_antimalware when an infection takes place. In Figure 7 below, a web server protected by mod_antimalware is shown on the right. The malware scanning service is shown on the left. The malware scanning service crawls the pages and web applications hosted on the web server, dynamically analyzes the content for malware, and in the case of an infection reports the infection to the user. In addition, the malware scanning service deploys quarantining directives to the mod_antimalware module running on the web server.

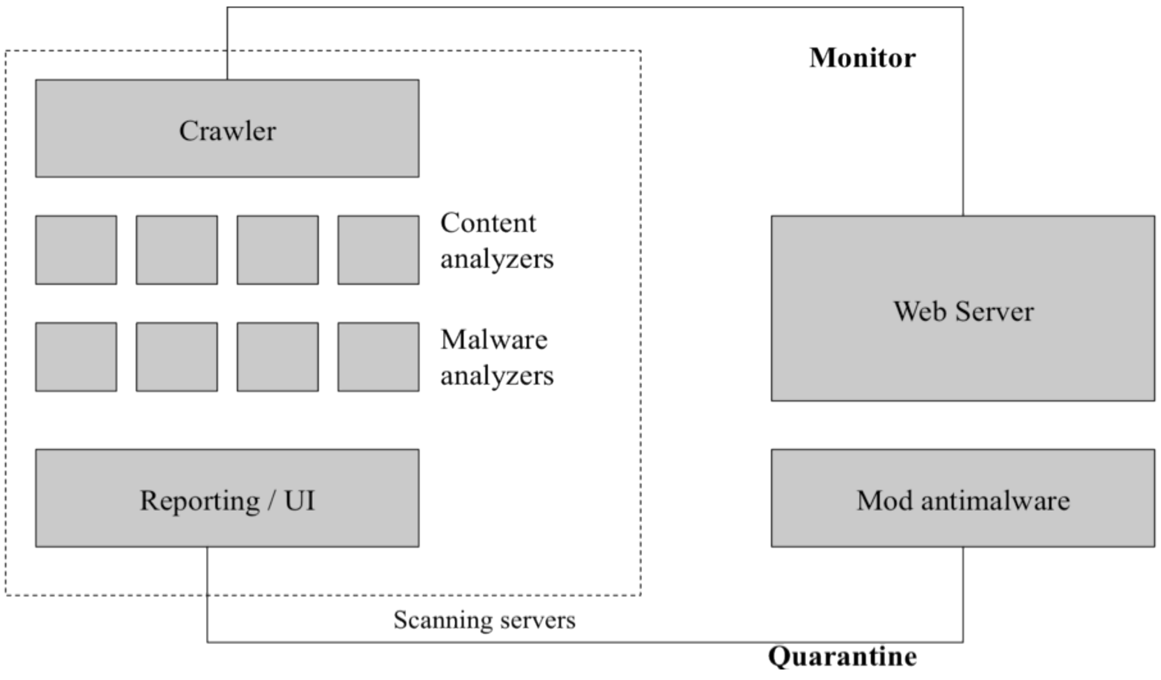


Figure 7. Mod_antimalware Architecture.

The malware scanning service authenticates itself to the mod_antimalware module using a shared key set up ahead of time, such that only it is authorized to make changes to the web server configuration to protect users from infected web pages.

Without mod_antimalware, the amount of time that users of an infected site are exposed to malware is much larger than with mod_antimalware installed, as shown in Figures 8 and 9 below. In a setting where mod_antimalware is not deployed (see Figure 8), after a site is compromised and turned into a distribution point for malware, it often takes days for the site to 1) discover the attack, 2) diagnose which pages and parts of the site are infected, 3) identify what malicious code was injected onto the site, and 4) contain the infection by manually removing the malicious code. However, with mod_antimalware installed, the exposure is limited to minutes or hours. As per the timeline shown in Figure 9, as soon as the malware monitoring service detects the infection, the webmaster can be alerted and either with approval or in parallel, quarantining directives can be sent to mod_antimalware, which will automatically contain all pages on which the infection occurs.

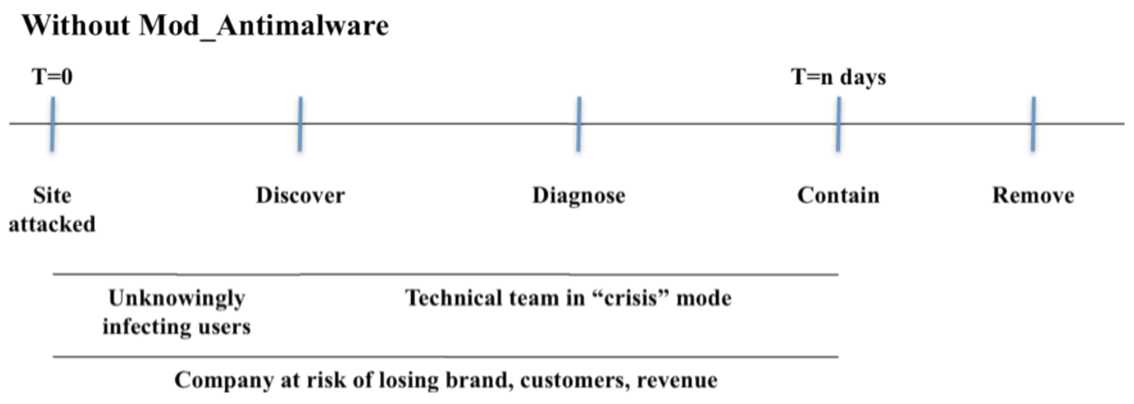

Without Mod_Antimalware
T=0
T=n days
Site attacked
Discover
Diagnose
Contain
Remove
Unknowingly infecting users
Technical team in "crisis" mode
Company at risk of losing brand, customers, revenue

Figure 8. Event timeline without mod_antimalware.

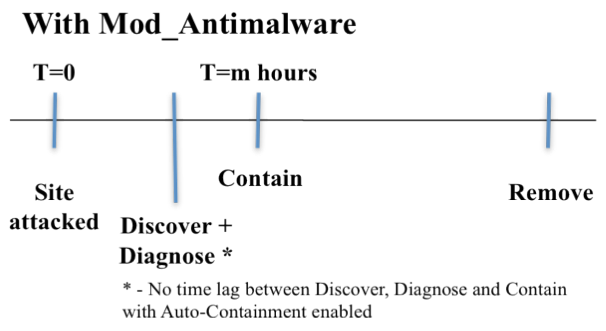

With Mod_Antimalware
T=0
T=m hours
Site attacked
Contain
Remove
Discover + Diagnose *
* - No time lag between Discover, Diagnose and Contain with Auto-Containment enabled

Figure 9. Event timeline with mod_antimalware.

The malware monitoring service is configured to regularly scan a website for malware infections, and provides the necessary information to mod_antimalware in the case of an infection to help contain it. For instance, Dasient's malware monitoring crawls a website and analyzes the content on each page for signs of compromise and malware infection, using both browser emulation and virtual machine emulation technology. In the case of an infection, the malware monitoring service identifies all URLs affected by the infection, and supplies enough forensic information in the form of quarantining directives to allow mod_antimalware to automatically contain the infection.

Now that we have described how mod_antimalware functions at a high-level, we walk through the technical details of its architecture and operation. It is assumed that the reader is familiar with Apache and/or IIS modules in general, and we only describe those aspects specific to mod_antimalware. For more information on Apache or IIS web server modules please see http://modules.apache.org/ or http://msdn.microsoft.com. Also, while versions of mod_antimalware exist for both Apache and IIS, the examples that we use in this paper are of the Apache flavor of mod_antimalware (although there are only typically minor differences for the IIS flavor).

Mod_antimalware is structured as an output filter that can intercept outgoing web pages and modify them on-the-fly. It uses the standard Apache module plug-in architecture, which allows for straightforward deployment. Much as webmasters install modules such as mod_ssl to allow their web servers to conduct secure transactions, webmasters can install mod_antimalware to protect their sites of the negative ramifications of web-based malware infections.

The functionality of mod_antimalware can be characterized by: 1) the output filter itself, 2) the set of "handlers" that it makes available, and 3) the set of configuration directives that it supports.

To work properly, mod_antimalware must be configured to run after any other output filter that may be used to construct the content to be sent to the user, but before any other output filter responsible for compressing or encrypting the content. Mod_antimalware is typically configured to only process HTML documents output by the web server or by any web applications installed on the web server.

Mod_antimalware makes available three “handlers” that are used to configure and monitor its operation. *Handlers* are URLs that are served by the module instead of by files or web applications installed on the web server.

Mod_antimalware _lite supports four configuration directives: 1) DasientSharedAuthKey, 2) SetBlacklistRedirectMessage, 3) BlacklistRedirectUrlPrefix, and 4) RemoteDirectivesFile. Mod_antimalware supports additional configuration directives as well. In the following, we describe the handlers and configuration directives in more detail.

3.1 Handlers

The three handlers that mod_antimalware supports are: /statusz, /configz, and /topz. For example, if mod_antimalware is installed on a web server for www.domain.com, then the current status of the module can be queried via the URL http://www.domain.com/statusz. As we do not want casual users of the web site or passers-by to check the status or configure mod_antimalware, any request to the handlers that mod_antimalware supports must include the key shared by the authorized monitoring service and the web server as an “auth” query string parameter. In addition, the request must be made using the HTTP POST method (instead of the GET method), as we would not want the shared key to inadvertently be logged at any intervening proxy server or in the web server’s logs. Further, the requests to these handlers must take place over SSL, else the request will be rejected. These handlers are typically accessed by the malware monitoring service, but can also be accessed via properly-formed and SSL-protected requests for debugging purposes.

3.1.1 Statusz

The /statusz handler outputs the following information upon request:

- process id: the id of the web server process that handles the request
- parent process id: the id of the parent web server process
- the current time
- the time at which the last successful set of configuration directives was sent
- the content of the last successful set of configuration directives that were sent

The /statusz handler is used to check if the module is running properly, and is also checked after a new set of quarantining directives are sent to the web server.

3.1.2 Configz

The /configz handler is used to send new configuration directives to mod_antimalware. Configuration directives are provided to the /configz handler as POST data following the “auth” parameter. Upon receiving a POST request to the /configz handler, mod_antimalware first attempts to write the newly provided directives to a configuration file on the web server (typically called antimalware.conf) for persistence. Then, mod_antimalware processes the new configuration directives for usage by the web server process that received the request. At the same time, the web server may be running many processes to serve client requests, and it is important that all such processes start using the updated configuration directives. As such, mod_antimalware is designed such that all web server processes use the configuration directives stored as part of a global state via shared memory. Once the new configuration directives are stored to disk, the global state is updated with the new configuration directives.

After making a request to the /configz handler to provide updated configuration directives, it is highly recommended that the malware monitoring service should make a request to the /statusz handler to confirm that the updated configuration directives were successfully received and enabled.

3.1.3 Topz

The third handler /topz is provided as an optimization to give the malware monitoring service information about what are the most frequently accessed URLs. Each time that an HTTP request is made to the web server, mod_antimalware updates a hash table keyed by URL to increment a count of the number of times that URL was recently accessed. The number of keys tracked by the hash table is limited, and entries with keys corresponding to the least frequently accessed URLs are dropped from the hash table, so as to conserve the amount of web server memory used by the hash table. The malware monitoring service can periodically make a request to the /topz handler to get a list of the "top" URLs most frequently accessed, and can configure itself to scan those URLs most frequently.

It should be noted that when the malware monitoring service makes a /topz request to mod_antimalware, the request could be served by any web server process, and each web server process keeps a local hash table for efficiency. (If the hash table were to be stored in shared memory, updates to it could become a bottleneck.) Mod_antimalware assumes that requests for URLs are approximately evenly spread across all web server processes, and when the malware monitoring service makes a /topz request, it expects that the response it receives is only a sample. For more accuracy, the malware monitoring service could make multiple /topz requests and note which process id provides the response to average across samples provided by individual processes.

3.2 Configuration Directives

Configuration directives are, in general, instructions provided to a web server that specify how it should go about serving web pages. We distinguish between two types of configuration directives: static and dynamic. *Static* configuration directives are specified in web server configuration files that are present at the time the web server starts, and are not expected to change while the web server runs. On the other hand, *dynamic* configuration directives are specified while the web server runs.

Mod_antimalware_lite supports dynamic configuration directives, and has a mechanism built into it that supports restart-free reconfiguration. *Restart-free-reconfiguration* allows remotely-specified, dynamic configuration directives to be applied and used by web server processes without requiring a restart of web server process. Such dynamically specified directives are written to disk (as will be described shortly) in a web server configuration file such that their effects are persistent across web server restarts. Such directives are not be confused with static directives even when they are loaded from a file. In particular, the contents of the file can be changed remotely at any time, hence making them dynamic.

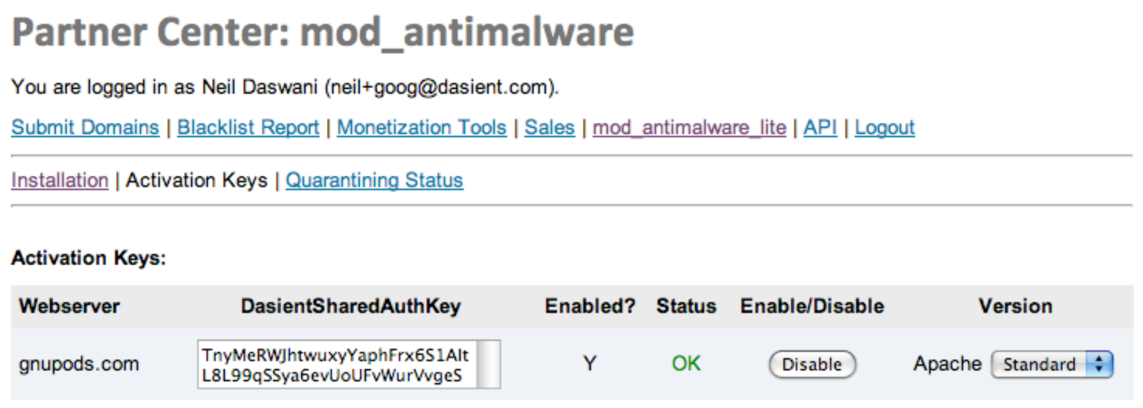
Quarantining directives are a type of configuration directive supported by mod_antimalware and mod_antimalware_lite. A quarantining directive can be a blocking directive or a filtering directive. A *blocking* directive will prevent an infected URL from being served altogether, and such that known clean content can be served instead. A *filtering* directive can serve the non-infected parts of content for a URL that has been affected by a web-based malware infection, while suppressing the infected parts from being served to users. Mod_antimalware supports both blocking and filtering directives, while mod_antimalware_lite supports only blocking directives.

In the following, we describe those configuration directives supported by both mod_antimalware_lite and mod_antimalware. The DasientSharedAuthKey, SetBlacklistRedirectMessage, and RemoteDirectivesFile configuration directives are static directives, and it is expected that there only be one instance of each such directive in the web server configuration. The BlacklistRedirectUrlPrefix directive is a dynamic directive that is sent remotely by the malware monitoring service when a URL or a set of URLs is infected.

3.2.1 DasientSharedAuthKey

The DasientSharedAuthKey directive specifies the shared authentication key that a client malware monitoring service must use when making requests to the mod_antimalware handlers. The authentication key must larger than the number of unicode characters specified by the AUTH_KEY_SIZE_IN_CHARS constant, which is set to 8192 (or 4KB) by default. An example of the DasientSharedAuthKey directive is shown below. Of course, please do not ever use the shared authentication key used below for illustration purposes in your mod_antimalware or mod_antimalware_lite installation.

```
DasientSharedAuthKey
DontTryMeRWJhtwuxyYaphFrxDLKJLjlSDFskdjfosknKNDSLKfjslSDklfSLKKZZwslPIsDFDlE9sVjTVK
klMzYvUpkoVgxvkFqqz5iaMvtdpveqON3BTfpKXpP25w6iu6…
```



Partner Center: mod_antimalware

You are logged in as Neil Daswani (neil+goog@dasient.com).

Submit Domains | Blacklist Report | Monetization Tools | Sales | mod_antimalware_lite | API | Logout

Installation | Activation Keys | Quarantining Status

Activation Keys:

Webserver	DasientSharedAuthKey	Enabled?	Status	Enable/Disable	Version
gnupods.com	TnyMeRWJhtwuxyYaphFrx6S1Alt L8L99qSSya6evUoUFvWurVvgeS	Y	OK	Disable	Apache Standard

The shared key is assigned by the malware monitoring service. The figure above shows a screenshot of Dasient's Partner Center, which assigns a shared authentication key. After adding the shared key as a parameter to the DasientSharedAuthKey directive in the web server configuration file, the webmaster can click the "Enable" button. Dasient's Partner Center then issues a request to the /statusz handler with the shared key specified as an "auth" POST parameter to test that a secured communication channel between the malware monitoring service and mod_antimalware works.

3.2.2 SetBlacklistRedirectMessage

SetBlacklistRedirectMessage is a static directive that allows a webmaster to specify what text or HTML should be output to the user when the user attempts to access an infected URL. When a blocking directive (such as BlacklistRedirectUrlPrefix) specifies that an infected URL should not be served, the text or HTML specified by the SetBlacklistRedirectMessage directive is output to the user instead of the content of an infected web page.

An example of usage of the SetBlacklistRedirectMessage directive is shown below.

```
SetBlacklistRedirectMessage "This server is experiencing technical difficulties.
Please come back later."
```

In the case that a URL becomes infected and the malware monitoring service issues a blocking directive for the URL (presumably, due to the URL having become infected), the message "This server is experiencing technical difficulties. Please come back later." will be served to the user instead of serving an infected web page. The benefit of the message is that is gives the webmaster control of what the user sees in the case of an infection, instead of serving infected content or having the user see a "red screen of death" or other message issued by a browser with anti-malware protections or a search engine that has blacklisted the URL.

3.2.3 BlacklistRedirectUrlPrefix

BlacklistRedirectUrlPrefix is a dynamic, remotely-specified, blocking directive that takes a URL prefix as a parameter, and will instruct mod_antimalware or mod_antimalware_lite to serve the pre-configured message specified by the SetBlacklistRedirectMessage instead of serving a URL that matches the specified prefix.

Quarantining directives (whether they be blocking or filtering directives) can be used to protect entire sections of a web site that match the specified URL prefix with a single directive. For instance, in the case that an attacker has injected a malicious IFRAME into the header or footer of a blog that is part of a larger site under the path domain.com/ugc/blog, all blog posts can be quarantined with a single directive, even if the blog is made of up hundreds of thousands of URLs with blog posts.

3.2.4 RemoteDirectivesFile

The RemoteDirectivesFile directive is a static configuration directive that specifies the on disk location where dynamic, remotely-specified directives should be stored for persistence. An example of such a directive can be found below:

```
RemoteDirectivesFile /etc/apache2/antimalware_remote_directives.conf
```

3.2.5 Example configuration directives file

Following is an example antimalware.conf file that is used to configure mod_antimalware_lite:

```
<IfModule mod_antimalware_lite.c>

# Static mod_antimalware_lite config directives

RemoteDirectivesFile /etc/apache2/antimalware_remote_directives.conf

DasientSharedAuthKey your_key_goes_here

SetBlacklistRedirectMessage "This server is experiencing technical difficulties.
Please come back later."

# Standard Apache Directives for proper mod operation
Include /etc/apache2/antimalware_remote_directives.conf
SetOutputFilter antimalware-lite-output-filter

# Sample blacklist directive (for testing purposes)

BlacklistRedirectUrlPrefix /foo1.html

</IfModule>
```

The configuration directives above are only used if the mod_antimalware_lite.c module is installed, as per the <IfModule> block. Note that the file above utilizes:

- the three static mod_antimalware_lite configuration directives: RemoteDirectivesFile, DasientSharedAuthKey, and

SetBlacklistRedirectMessage

- two standard, static Apache configuration directives: an Include directive that ensures the remote directives file is sourced in and used, and a SetOutputFilter directive that instructs the web server to actually use the mod_antimalware_lite output filter

- a BlacklistRedirectUrlPrefix directive; such a directive is typically a dynamic configuration directive supplied by the malware monitoring service, and written to the remote directives file (antimalware_remote_directives.conf) at run-time in the case of an infection. Solely for testing purposes, the directive is supplied in the antimalware.conf file above.

3.3 Usage Scenario

Now that we have described the directives and handlers supported by mod_antimalware, we describe how they are used in little more detail in the Mr. Rogers example from Section 2.0. The malware monitoring service makes periodic calls to the /topz handler, and finds that the home page itself (with path '/') is the most frequently accessed, and decides to monitor that page once per hour. When the malware monitoring service determines that the Mr. Rogers web site is infected (due to the malicious IFRAME being injected), it sends a filtering quarantining directive for the home page to mod_antimalware by making a request to the /configz handler. It then makes a request to the /statusz handler to confirm that the quarantining directive was successfully received and enabled. Mr. Rogers web server therefore invokes the output filtering function in mod_antimalware when the home page is accessed. The contents of the home page, with the exception of the malicious IFRAME, is served to users, thereby allowing Mr. Rogers to safely delight all the parents and children accessing the Mr. Rogers home page despite the best efforts of the hackers to infect the web site and serve malware drive-by-downloads to them.

The heart of the operation of mod_antimalware is that on each URL request, the module determines if any of the quarantining directives should be applied (based on the URLs and/or prefixes specified in the directives), and outputs either the message specified by the SetBlacklistRedirectMessage directive or filters the content for malicious code specified by filtering directives.

4.0 Quarantining Verification

Depending upon the nature of the web-based malware infection, it is possible that filtering directives may or may not be successful in containing the infection. While filtering directives are indeed successful in the majority of cases, we do not want to serve infected pages when filtering directives are not successful.

As such, when the malware monitoring service sends filtering directives, it is highly advisable that it should re-scan the infected URLs to determine if infections may still be present. If so, it is recommended that it "upgrade" to sending blocking directives for URLs that may still result in infections due to unsuccessful filtering. This process of re-scanning URLs for which filtering directives have been sent and upgrading to blocking directives if necessary is called *quarantining verification*.

To allow the malware monitoring service to properly conduct quarantining verification and upgrade directives when necessary, it is important to distinguish between three cases: 1) a page that is infected but is being successfully quarantined, 2) a page that is infected but is not being successfully quarantined, and 3) a page that is no longer infected because the web-based malware code snippets have been cleaned and eliminated from the page. In case (1), no upgrade is necessary. In case (2), an upgrade is necessary. In case (3), the quarantining directive can be lifted.

To help the malware monitoring service distinguish between these three cases, mod_antimalware returns a "X-Quarantine: 1" HTTP response header in the case that a blocking or filtering directive was used in serving a URL. In case (1), the malware monitoring service will see the X-Quarantine header and will not see an infection on the page, indicating that quarantining is successful. In case (2), the malware monitoring service will see the X-Quarantine header and will see an infection on the page, telling it that needs to upgrade from a filtering directive to a blocking directive. In case (3), the malware monitoring service will see a clean page and no directive, letting it know that it can “lift” the quarantining directive.

5.0 Summary

This paper has described the architecture and implementation of mod_antimalware, a web server module that works together with a malware monitoring service, to contain web-based malware infections in an automated fashion. Mod_antimalware is an output filter that supports configuration directives that allow for two types of quarantining of infected web pages: blocking directives and filtering directives. Mod_antimalware provides support for the remote management of such directives by a malware monitoring service. Together with the malware monitoring service, mod_antimalware provides “the complete lifecycle” of malware protection for web sites, consisting of detection and automated mitigation. The combination of these capabilities allows web-based malware infections to be quarantined before they impact users and before a web site suffers from the negative ramifications of being blacklisted by popular search engines and browsers.

Appendix: Installation Guide for mod_antimalware_lite

```
Download mod_antimalware_lite from:

http://sourceforge.net/projects/modantimalware/


=== REQUIREMENTS ===

* You must be running Apache 2. This mod is not compatible with 1.x.

* You must be running the prefork MPM. To verify this, run:
  $ httpd -V | grep MPM
  Server MPM:     Prefork

  (sometimes httpd is installed as "apache2")

* You are using Apache as a dedicated server. This module only supports
  quarantining pages that are in the default host. Virtual host ("vhost")
  support is coming in a future release.

* You have mod_ssl and port 443 enabled. Inbound control connections from
  the Dasient back-end come over SSL on port 443. It's OK to have a self
  signed certificate. There are many HOWTOs and guides available on how
  to install mod_ssl and create a self signed SSL certificate; here is one
  example: http://www.securityfocus.com/infocus/1818

=== COMPILE & INSTALL ===

1. Install apxs (http://httpd.apache.org/docs/2.0/programs/apxs.html).

To see if the mod is already installed, type "apxs" or "apxs2" at a
shell. Apxs typically comes with the apache2-devel or httpd-devel package,
depending on your distribution. Example installs:
  [CentOS] yum install httpd-devel
  [Ubuntu] apt-get install apache2-prefork-dev

2. Compile and install the mod.

The module is packaged using GNU autoconf, which should make for a familiar
install process.

$ ./configure
$ make
$ sudo make install

3. Get an activation key (DasientSharedAuthKey) from Dasient.

The mod_antimalware_lite web server module communicates with Dasient's malware
monitoring service. If a web site gets infected with malware, Dasient's
back-end malware analysis servers will make an authenticated SSL connection to
your web server to provide mod_antimalware_lite with instructions on which URLs
have been infected, and will block those web pages from being served to protect
```

your users. In order to authenticate to your server, an activation key is used by Dasient. To get your activation key, visit:

    http://wam.dasient.com/wam/partner_center

Click into the "mod_antimalware_lite" tab. If you have not already done this yet, sign up for trial malware monitoring, and find the activation key that you are given once the trial malware monitoring has been set up. This key will be put inside a configuration file in one of the following steps.

4. Apache Configuration.

[A] The main configuration files. These files load the module and sets some base parameters. Template files come with the distribution: antimalware.conf and antimalware.load. Some basic instructions are below for how to include them in your Apache configuration.

[[Ubuntu && other distributions that have a2enmod]]

Place antimalware.conf and antimalware.load in /etc/apache2/mods-available/. Run "a2enmod antimalware" to enable the mod. In the future if you wish to disable the mod, run "a2dismod antimalware".

(note: on some distributions, the path is /etc/httpd/mods-available/).

[[Custom Apache install // no a2enmod]]

Place the files in your main configuration directory, and add "Include" directives from your main httpd.conf:

```
Include antimalware.load
Include antimalware.conf
```

It's not important where in httpd.conf the Include directives go, as long as they are not wrapped in a virtual host.

Once the configuration files are installed, there are two important changes you need to make:

- The file referred to by the "Include" and "RemoteDirectivesFile" statements should point to a valid path. It's ok if the file doesn't exist yet. The file passed to these directives must match. See [B] for more details on the remote directives file.

- DasientSharedAuthKey. Replace the text your_key_goes_here with the actual key that you obtained from the partner center:
  http://wam.dasient.com/wam/partner_center

[B] The remote directives file. When an infection is detected on your web server, Dasient remotely sends additional directives to mod_antimalware_lite to block infected web pages from being served. These additional directives are stored in a file that must be writable by the web server process, and ideally in the same location as other Apache .conf files. As every web server configuration may be slightly different, you should choose a location that makes sense for your web server.

Here's an example for a default Ubuntu installation, where you are placing the remote directives file in /etc/apache2 and the web server is running as the default user, www-data:

```
$ sudo touch /etc/apache2/antimalware_remote_directives.conf
$ sudo chown www-data /etc/apache2/antimalware_remote_directives.conf
$ sudo chmod 644 /etc/apache2/antimalware_remote_directives.conf
```

If you aren't sure where to put the file, it's a safe bet to put it in the same directory as your main httpd.conf file.

If you aren't sure what user your webserver runs as, there are a couple ways to find out:

- Look in httpd.conf for a "User" directive
- Run "ps -ef | grep apache" and "ps -ef | grep httpd" to see what user owns the majority of the processes.

5. Check your configuration and restart apache

```
$ apachectl configtest
$ sudo apachectl restart
```

This will test your new config parameters and then restart Apache to include the new module and config.

6. Basic testing

The sample antimalware.conf file is pre-configured to block a webpage called /foo1.html as if it was infected. Create a page called foo1.html under your web root directory. Confirm that this page is blocked by the webserver (due to mod_antimalware_lite) by accessing it through a browser or command-line tool. The text message configured by the SetBlacklistRedirectMessage directive inside antimalware.conf ("This server is experiencing technical difficulties. Please come back later.") should get displayed when the page is blocked.

=== Using mod_antimalware_lite ===

Now your mod_antimalware_lite installation is ready to be tested by receiving directives from Dasient's malware analysis servers. Test out your mod_antimalware_lite installation by conducting the infection test as described in the Dasient Partner Center at http://wam.dasient.com/wam/partner_center (click on the mod_antimalware_lite tab).

Questions? Please email support@dasient.com.