To: CyberSpark Open Source Project
From: Sky
Date: 4/2/11
Subject: Documentation of **CyberSpark** the *service* and the *daemon*

Red7 Communications, Inc.
PO Box 27591
San Francisco CA 94127-0591
USA
Phone +1 415.759.7337
web.red7.com

The newly rewritten **CyberSpark** monitoring software provides a platform for alerting bloggers, journalists and human rights NGOs when their web sites come under attack. This important task is accomplished by cloud servers running this software from various hosting facilities around the world.

The software is free open source written in the PHP scripting/ programming language.

Infrastructure

The monitors run on Ubuntu or other virtual hosts. Installed in cloud facilities like Rackspace Cloud or Amazon EC2, they require only PHP (command line) and a couple of PHP-based support packages. Alerting is done through SMTP servers like those of gmail. The systems are "hardened" in the sense that they present no face to the outside world except for an SSH interface, which is highly secure. There is no web server, email or other service(s) running or exposed on these servers.

Installation of a new CyberSpark instance requires about 30 minutes from scratch, or about 5 minutes if performed by cloning an existing instance of a monitoring server.

Requirements

Requirements: Ubuntu (Linux) 8.xx or 10.xx; PHP 5; PHP CLI (command line interface); PEAR Mail, Network, Sockets; a separate SMTP email server (preferably using SSL). Optional components: Google Safe Browsing interface (in Python, available from Google Code).

Cyberspark—the *service*

Cybersparkd - cyberspark is usually installed as a *service* to be launched upon system startup. A file /etc/init.d/cyberspark controls the starting and stopping of the cyberspark daemons. (this is a standard method used in Debian and Ubuntu

to launch and control services.) A primary process "cybersparkd.php" is launched at startup - it looks for an available configuration file and the *properties* files that specify which URLs are to be monitored and the conditions under which exceptions are to be reported to third parties by email. It launches one instance of cyberspark.php (note no "d" at the end) for each properties file it finds. These processes run as daemons, continually monitoring their assigned targets and issuing alerts as required. When cybersparkd is terminated (SIGINT or SIGKILL), it terminates those child processes, each of which shuts down gracefully, saving data as required for the next startup.

Cyberspark.php—the *daemon*

Cyberspark.php: Each cyberspark process initializes by reading any persistent data from its store, then reads its properties file and begins processing directives one by one. It can run in daemon mode, meaning that it will loop until terminated, or it can run in a once-only mode where it drops out after processing all directives once only. If started by **cybersparkd** as part of the system daemonization process, it generally runs until the master (parent) process terminates it with a SIGINT. Each "URL=" specification in a properties file tells the daemon to spider a URL, process it using any filters that have been selected for that URL, report out by email as directed by those filters, and then go to the next directive in the properties file. Each line in the file is a standalone directive. The frequency of looping through the directives is also specified within the properties file.

Concurrency and threading: Because the service launches a daemon for each properties file, in a system with multiple CPUs it is possible for several daemons to be executing concurrently. Since they do not communicate with each other and share no resources, this shouldn’t pose any problems.

Data storage—the persistence model

Persistent data store: Each cyberspark.php daemon maintains its own persistent data store in a file. The store is an associative PHP array that contains a few daemon-related values and separate sub-arrays for each filter (see below for more on *filters*). The daemon itself takes care of saving each filter's data and presenting it to the filter when it's next needed.

General monitoring: Even without complex filters there are a number of monitoring operations built into the system.

Basic: Even without filters specified, the URL is checked for proper HTTP response and you’re alerted if certain basic errors occur.

IfExists: The URL is monitored and everything is considered to be “OK” as long as the URL does not respond. (404, 30x, 500 errors are all considered OK.) But as soon as an HTTP 200 response (a “normal” result) comes through, you are alerted. This can be used to watch for pages or sites that should not be available, and you’ll be alerted when they appear.

Monitoring by IP address: There are times when a sensitive site may be “in preparation” for launch, but not listed in DNS for the domain. CyberSpark provides a way to monitor such a server by IP address, and *ifexists*, as described above, can alert you when the server comes online. In addition the *dns* filter (see below) watches and alerts when new server DNS records appear. To monitor a site by IP address, simply insert the IP address in the URL to be monitored, and prefix that by the site’s *virtual host* name between brackets. The virtual host name will be used in the HTTP request that is sent to port 80 on the IP address you specify. For example:

```
url=[sub.mydomain.com]http://8.8.8.8/home.html;email=me@me.com
```

would make a request of the server at 8.8.8.8, and would request that it serve up content for “sub.mydomain.com” - the server will only present content if it has been configured to do so, but will do it regardlessof whether the name “sub.mydomain.com” exists in any DNS record for the domain.

Filters—flexible scanning and alerting

Filters: Self-contained PHP filtering software can be written and dropped into the /filters directory. As each cyberspark daemon launches, it examines these filter files and gives each of them an opportunity to initialize and assert its own priority. Once installed in this fashion, each filter is activated only when a particular *URL=* directive requests it. This means that filters may be applied where desired or required on a per-URL basis. During the development process it is even possible to write and test new filters with minimal or no disruption to an ongoing monitoring system. A filter is presented with data in an isolated environment where it receives the URL, contents of the page (or file) behind the URL, a number of other parameters or values, and the filter can "reply" by returning both a *status* and a *message* to be sent as part of an email alert as well as recorded in a log file.

Examples of filters:

Length: For web sites with static and unchanging content, the simplest filter is the *length* filter. This filter records the length of each page or file, and if it has changed since the previous check, it generates an alert. This filter is appropriate for JavaScript or CSS files for which a change in length would indicate either that the file has been hacked or was changed by the webmaster (and presumably the

webmaster would know the difference). Length data is kept in this filter's private store (maintained by the daemon but available only to the filter).

Smartscan: this filter reduces the page so it contains only the script, and iframe contents of the page. These are then compared with the previous contents, and an alert is sent if they have changed. Only the contents on the page are compared, though the filter certainly could be extended to fetch any contents from a *src=* JavaScript specification using only the tools readily available in this system.

Gsb, gsb2 and *gsbdaily*: Google Safe Browsing is a system that searches out malware in web sites and flags them. Browsers that subscribe to this service will show a warning page if you try to navigate to a site that has been flagged as containing malware. This database can be interrogated, though the API is moderately complex, and any URL can be checked against the database. Because of its complexity, we do not provide code as a part of the CyberSpark package, but instead we interrogate by making requests of a separate web-based service. (You can set up your own by downloading source from Google Code and installing it on a separate server.) the *gsb* filter checks the URL and all links contained on the page against the a gsb web service and generates an alert if any of the domains is on either the malware or the phishing lists. The *gsb2* filter checks the URL itself, any pages that page links to, and any pages those pages subsequently link to. In other words it checks "two levels" out from the base URL. Because this can generate hundreds of links to be checked, and thus hundreds of requests to the gsb web service, we have also made a *gsbdaily* filter, which only performs this exhaustive check once a day per URL. The filter is capable of surviving even if the gsb web service goes down, and can alert a supervisor by email when this happens.

dns: This filter watches you DNS entries for changes. In most domains, changes are infrequent, and could indicate a *domain hijacking*. The records watched are SOA, MX, NS, TXT (including SPF), CNAME, “A” and “AAAA”. Any change in any record is reported, including record additions and deletions.

Alerting

Alerting: All alerts are sent by email. Preferably your interface to an SMTP server is ssl-encrypted, though it may be through any SMTP server on any available port you wish to use and need not be encrypted. If you specify port 465, a secure connection will always be attempted. We recommend, for the sake of resiliency that you use a mail service that is robust and won't fail under attack. We do not suggest sending alerts directly from the monitor server itself, though you could certainly do that if you wanted to combine monitoring with such other (not so secure) services on the same machine. Sending SMS text messages can be accomplished by sending to an email address associate with the mobile device, if such an address is provided by the mobile carrier. These addresses are

typically of the form 0000000000@txt.ATT.net (where the phone number replaces the zeros, for AT&T subscribers, for example).

Inside a filter

Writing a new filter: Each filter must reside in a file with extension ".php" in the /filters subdirectory. When the daemon launches, it checks each file in this directory, attempting to execute a set-up function with the same name as the file, and that set-up function may install any number among three types of *callback* functions. The first type, *init*, is executed by the daemon after all filters have been installed and before the first pass through any monitoring loop. This would provide the filter with an opportunity to initialize itself, read any private data that it needs from elsewhere (either on the server's file system or on a remote URL) and set up the filter's private store. The second type is the *destroy* callback, which is executed when the daemon enters its shutdown process. This is the filter's opportunity to save anything special or to alert a remote web service if necessary. And probably most important the *scan* callback, which can examine the contents of a URL and trigger alerts or messages. Each of these is presented with the private data store as it begins execution, and can write values into the store at any time. Those values will be preserved from one execution of the filter to the next.

To write or test a filter without upsetting a production system: Even though you generally don't want to put experimental code on a production system, it is possible to write a new filter, or to change an existing filter on a production system. The process is dangerous in that it can upset the running production system, but if you're careful you can do it properly. First, you have to have the daemon running (/etc/init.d/cyberspark start) - it will read all properties files and set up daemons for each of them. These daemons will continue to run while you're testing. Run *top* or *htop* to be sure they continue running.

Now, duplicate an existing properties file, or create a new one, with a different name. (Let's say you currently are running CS8 and have CS8-0 and CS8-1 daemons running using properties files CS8-0.properties and CS8-1.properties...you might create CS8-2.properties for this testing.) Now you can create a new filter file. Be sure it uses a name not in use already, and adjust the name of the primary set-up function within that file so it matches the file name (without the ".php" extension of course). If you're rewriting an existing filter, you copy the existing file and change the set-up function name, but you must also change the names of other functions included in the filter, otherwise they'll conflict with (duplicate) existing function names from the old filter. (If you plan to leave the old filter in place, you can always choose to use those old functions and simply not duplicate them at all in the new filter.)

In order to test the new filter, you'll need to create or alter an existing *url=* directive in the new properties file. Insert the name of the new filter in the

";conditions=email@email.com" conditions section. If your new filter is named "test" then you might have a new directive like:

url=http://web.red7.com/;test=me@cyberspark.net

Make all of the coding adjustments you want to make within the new filter. The new filter is ignored by the running daemons because they only look for filters when they start up. (If you restart cyberspark, of course, then your new filter will be discovered, along with the properties file, and a daemon will be created to run it...you won't want to have this happen when you're testing.) You can now run the cyberspark.php process to test the new filter:

php /usr/local/cyberspark/cyberspark.php --id CS8-2

This does a one-time run which will help you test the filter. You can freely insert *echo* statements in the PHP code to help you debug, since you're running from the command line.

If you make any changes to existing filters' code, they may be picked up by the running daemons - so you'll want to avoid that unless you're very, very careful.

Once you've finished testing your new filter, you can either use it as-is and insert the new condition name into the existing properties files, or you can change its function names and replace an existing filter. While you do this you want to have the cyberspark daemons all shut down.