What is WebJob?

Andy Bair

February 15, 2006

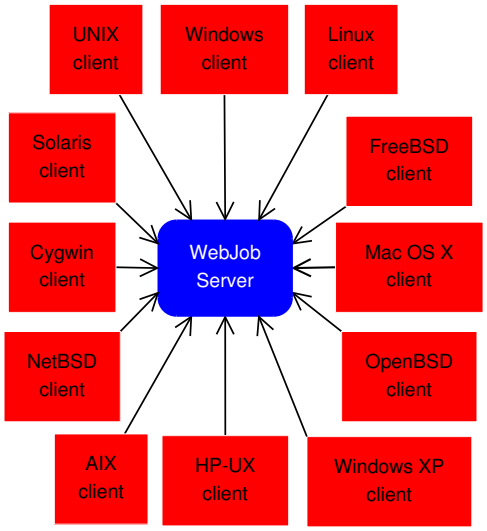

Figure 1: WebJob: The High-Level View

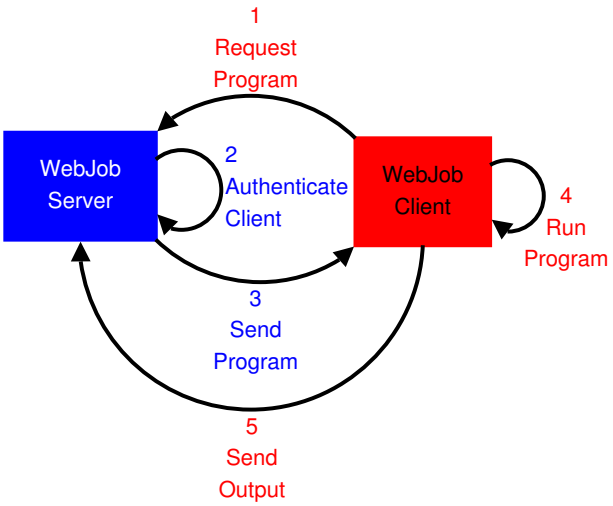

Figure 2: WebJob Actions

WebJob Overview

WebJob is a client-server system where the client requests and downloads a program from a server, executes that program on the client, then uploads the results to the server. Figure 1 depicts a number of diverse WebJob clients contacting a single server. WebJob is an open-source project that can be found at the following URL.

`http://webjob.sourceforge.net/WebJob/index.shtml`

WebJob is useful because it provides a mechanism for running known good programs on damaged or potentially compromised systems. This makes it ideal for remote diagnostics, incident response, and evidence collection. WebJob also provides a framework that is conducive to centralized management. Therefore, it can support and help automate a large number of common administrative tasks and host-based monitoring scenarios such as periodic system checks, file updates, integrity monitoring, patch/package management, and so on[1].

WebJob Details

Figure 2 shows the five steps performed by a typical WebJob session. The WebJob server is the Apache[1] web server configured to run the WebJob server com-

[1] http://httpd.apache.org

mon gateway interface (CGI[2]) program. The communication between the WebJob server and client can be unencrypted or encrypted through a Secure Sockets Layer (SSL) tunnel[3]. Please refer to Figure 2 as the WebJob steps are discussed in detail below.

1. First, a WebJob client makes a request to a WebJob server for a program. The example command shown below, requests the testenv program from the WebJob server. The `--execute` option directs the WebJob client to execute the program. The `--file` program controls the WebJob configuration. There are many options available, but generally this configuration file specifies the client's credentials and the WebJob server to contact. See the WebJob man page for more detailed discussion of the configuration file options[4].

   ```
   webjob --execute --file upload.cfg testenv
   ```

2. Next, the WebJob server receives the request and authenticates the client's credentials. The WebJob server can be configured to authenticate clients with username–password combinations or SSL certificates[5].

3. If the client's credentials are authentic, the server sends the requested program to the client.

4. Next, the client receives and executes the program. Optionally, the client could validate digitally signed binaries, providing a much greater level of security[6].

5. Lastly, the WebJob client uploads the three files to the WebJob server. The first file, known as the *out* file, contains the results of the standard output stream from the command executed. The next file, known as the *err* file, contains the results of the standard error stream from the executed command. The last file, known as the *env* file, contains timestamps, hashes of the output and error streams, and other information. The WebJob server produces a fourth file, known as the *rdy* ("ready") file, which serves as a trigger file indicating the four files are ready for processing.

[2] http://en.wikipedia.org/wiki/Common_Gateway_Interface
[3] http://en.wikipedia.org/wiki/Secure_Sockets_Layer
[4] http://webjob.sourceforge.net/WebJob/ManPage.shtml
[5] http://en.wikipedia.org/wiki/Public_key_certificate
[6] http://en.wikipedia.org/wiki/Digital_signature

WebJob Advantages

There are many advantages to WebJob. These advantages are listed below – they are quoted from the WebJob website[1].

- Ported to many operating systems. WebJob has been ported to many of the popular UNIX's, MacOS X, and Windows.

- Small client footprint. The WebJob is between 600 and 800 Kilobytes when compiled with SSL. So, for less than one Megabyte, you can have a secure, flexible, and powerful client. The small size means that there is much less software to manage on each client: only one binary and one or more configuration files.

- Critical components centrally managed. The tools that actually do what you need to have done are managed in one location, namely the WebJob server. Thus, scripts and programs can be kept in a state of continual readiness. Effectively, this increases your ability to adapt and respond to unforeseen events.

  For example, assume you have an environment where 1,000 clients check-in to a single WebJob server hourly. Normally, they download and run an empty wrapper script. Now assume you have a situation that requires collecting file hashes on all system to determine the presence of a malicious program. You can modify that wrapper script to run a program to collect file hashes and determine its presence on 1,000 computers in about an hour.

- Secure. Client-Server data can be exchanged safely and securely using SSL encryption and certificate authentication.

- Aggregates data. All harvested data is aggregated in one location – the WebJob server.
- Requires minimal networking. WebJob only requires an outbound TCP connection – typically on port 443. A WebJob server never initiates communication with a WebJob client. This eliminates an entire class of network-based attack vectors.
- Does not diminish client security posture. WebJob does not diminish the client's security posture because it is strictly a client side application and it runs in the security context of the user invoking it. In other words, the WebJob client does not accept inbound requests, and there are no inherent SUID/SGID issues.
- Jobs can be time limited. WebJob's GET, RUN, and PUT timers ensure that runaway jobs are terminated once user-specified time limits have been exceeded.
- Scales horizontally. In other words, a single WebJob server can handle multiple clients, and multiple servers within a single-tiered framework create additional capacity.
- Scales vertically. In other words, WebJob servers can be configured as clients to create a multi-tiered framework.
- Does not limit what you can do.

WebJob Disadvantages

Despite all the advantages to WebJob, there are some dangers that decision makers and engineers need to be aware of. The disadvantages are listed below – they are quoted from the WebJob website[1].

- Because WebJob is a general purpose tool, it cuts both ways. In other words, an attacker could use it to infiltrate and execute malicious tools.
- WebJob can't be completely trusted on a compromised host even when statically compiled – think kernel patch. The best you can hope for is to detect a breach before such a patch is put into effect. This could potentially be done by running host integrity checks on a frequent basis. By the way, if you suspect a kernel patch, your only true recourse is to take the system down and inspect it from another vantage point.
- To support batch processing, WebJob stores authentication credentials on the client system. Therefore, one must take measures to prevent and/or detect spoofing and replays. This becomes an issue as soon as the client is compromised.
- WebJob can't protect client-server exchanges when used without encryption and mutual authentication.

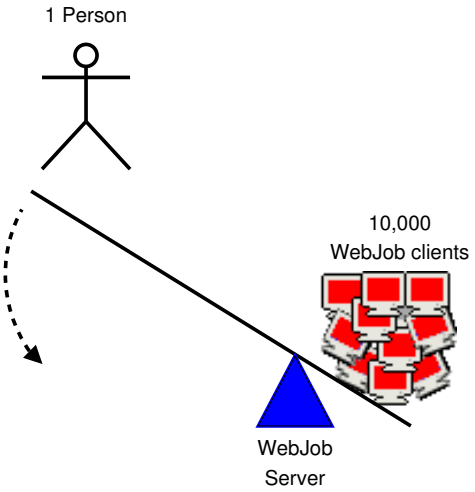

Figure 3: WebJob Leverage

Why Use WebJob?

While not ignoring the disadvantages, I would argue that the advantages far outweigh the disadvantages. With a basic understanding of how WebJob works, you can see in Figure 3 how WebJob can empower one person to conduct activities on thousands of clients. The capabilities are enormous.

WebJob has been used in a variety of ways and most are documented in recipes at the following URL.

`http://webjob.sourceforge.net/WebJob/Cookbook.shtml`

The WebJob-specific recipes are listed below.

- Harvest system information, load it into MySQL, and create a set of browsable HTML reports
- Manage system config files, rc scripts, and other selected, text-based files
- Insert/Remove specified cron jobs on an as needed basis
- Manage root's crontab
- Periodically (hourly/daily) run administrative tasks
- Periodically run administrative tasks via command bundles (scripts)
- Deploy and verify the installation of a FreeBSD package
- Deploy and verify the installation of a Solaris package
- Harvest and check Solaris ndd security settings
- Harvest and monitor argus data
- Harvest and monitor ps data
- Harvest ftimes map/dig data from Windows platforms using self-extracting executables (NSIS)
- Harvest lsof socket data (TCP/UDP)
- Harvest uptime data once a minute and periodically rsync it to a central server
- Run tcpdump on a group of IDS sensors to collect network traffic
- Run DISA's UNIX Security Readiness Review Scripts (SRRs)
- Check Solaris patch levels for compliance with Sun Alert reports
- Synchronize data (push/pull) using rsync, ssh, and dynamic keys
- Automatically update or repair a webpage
- Automatically update or repair a website
- Automatically compress WebJob uploads using triggers and configuration overrides
- Run a command if its hash matches a predetermined value

References

[1] WebJob can be found here: `http://webjob.sourceforge.net/WebJob/index.shtml`