Journal of Physics: Conference Series **664** (2015) 022010 doi:10.1088/1742-6596/664/2/022010

CernVM WebAPI - Controlling Virtual Machines from the Web

I.Charalampidis, D.Berzano, J.Blomer, P.Buncic, G.Ganis, R.Meusel and B.Segal

European Organization for Nuclear Research (CERN), Geneva, Switzerland

E-mail: ioannis.charalampidis@cern.ch

Abstract. Lately, there is a trend in scientific projects to look for computing resources in the volunteering community. In addition, to reduce the development effort required to port the scientific software stack to all the known platforms, the use of Virtual Machines (VMs)u is becoming increasingly popular. Unfortunately their use further complicates the software installation and operation, restricting the volunteer audience to sufficiently expert people. CernVM WebAPI is a software solution addressing this specific case in a way that opens wide new application opportunities. It offers a very simple API for setting-up, controlling and interfacing with a VM instance in the users computer, while in the same time offloading the user from all the burden of downloading, installing and configuring the hypervisor. WebAPI comes with a lightweight javascript library that guides the user through the application installation process. Malicious usage is prohibited by offering a per-domain PKI validation mechanism. In this contribution we will overview this new technology, discuss its security features and examine some test cases where it is already in use.

1. Introduction

During the past years, where virtualization became a standard part of the lifecycle of the experimental software, we have seen a revamped interest in volunteer computing in High Energy Physics. Being able to run any simulation or analysis framework in all known platforms, experiments are now trying to get a share on the volunteer community. Such behaviour is expected, if someone considers the well-established amount of resources available for free, and the boost in publicity a volunteer computing project can offer.

In this paper we are explaining how virtualization became the standard in volunteer computing in HEP and which difficulties had to be addressed. In *Section 2* we are presenting the *CernVM WebAPI* , a technology developed in order to address these problems, including all the technical details. In *Section 3* we are elaborating on the security features of this technology and on *Section 4* we present some cases where it has already been used. Finally, on *Section 5* we are summarizing our results and presenting our future plans.

1.1. Virtualization in Volunteer Computing

The foundation was set in 2011, when the LHC@Home 2.0 (Test4Theory) [1] was the first BOINC project to use virtualization. The base operating system was CernVM, since it offered an off-the-shelf solution for a virtualization-tailored Linux distribution, and a performant file system (CVMFS) [2] capable of delivering software and updates to all instances with minimal

effort. An in-house job submission mechanism, called Co-Pilot [3] was developed for bridging the gap between untrusted volunteer environments and the experiment workload management systems (e.g. ATLAS/PanDA or ALICE/AliEn). The project was quite successful and the implementation became the reference for similar projects to come in the future [4].

Of course the project did not run without any problems. An important note that was underestimated was the fact that not every volunteer is a computer expert. Indeed, when the project was launched there was a lot of feedback by people with deep computer knowledge, but after some time it was not spreading to the rest of the audience, as it would be expected. Similar behaviours have also been observed in other BOINC projects [5]. Judging by their feedback, the installation and the configuration of the hypervisor was an extra burden that not every volunteer was willing to carry.

In addition to that, not all the guest configurations were applicable on all the volunteer hardware. Frequently in such cases the hypervisor just stopped working, providing an obscure error, or no error description at all, making the debugging of such cases quite troublesome.

Finally, the hypervisor window was yet another point of attention in the variety of windows that accommodate the BOINC interface. For example, the user registration and statistics page is opened in a Web Browser, the computing resources allocated through the BOINC agent, while the project details are shown in yet another web browser or application window. This constant switch of contexts was also another topic of complaint by the users.

2. CernVM WebAPI

Following the suggestion of Ben Segal (CERN IT), *CernVM WebAPI* was developed as a solution to address all the problems related to the delivery, installation and control of a Virtual Machine (VM) in the users computer, including the installation and configuration of the underlying hypervisor. It is designed to automatically take all the appropriate actions in order to mitigate most of the known problems a user can face, keeping the input to the bare minimum.

This offload of maintenance tasks to the *CernVM WebAPI* not only benefits the end user, but the developer too. The system provides a very simple API for controlling VMs in the users computer, while in addition it offers a rich set of callback hooks, notifying the developer for every change in the system.

Its most advantageous feature though is the fact that all these operations are executed in the context of the web browser. This consolidates all the individual Graphical User Interfaces taking place in a volunteer computing project in just a single webpage. Even actions that momentarily require the users attention away from the browser appear as blocking pop-up interactions that upon disposal resume normal operations in the original window.

Finally, *CernVM WebAPI* also provides a mechanism for interfacing with an agent application inside the VM. A configurable TCP port can be designated as the *API Port*. The system will automatically set-up the appropriate NAT forwarding rules and will monitor the state of the port. When available, it will fire the appropriate callback to the user interface, including the endpoint to contact in order to establish a connection. It thus becomes possible to seamlessly integrate any kind of front-end the appliance provides, directly in the same website.

2.1. Sessions in CernVM WebAPI

Due to its ambiguous meaning, we would like to clarify the term *session* that will be used in this paper. A session is almost equivalent to a VM. However opening a *CernVM WebAPI* session doesn't mean that you start a VM. We could say that a session is the description of a VM and the channel to control its instance. In order to start or control a VM instance you need to open a session that describes it first. Opening a session either creates a new description or reuses a previous one.

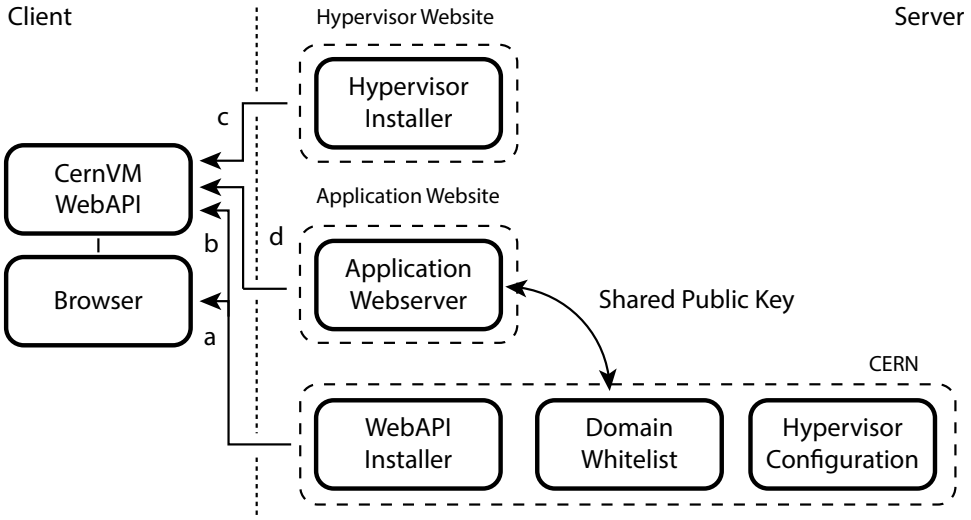

Figure 1. *CernVM WebAPI* component data flow. The client-side components are located on the left side, while the server-side components on the right side. The dotted groups denote external domains.

This is particularly useful in asynchronous environments – such as websites – where the user might navigate away from the application and come back again at a later time. In such cases, instead of creating a VM every time, the developer points the *CernVM WebAPI* to the VM description and it will create such a VM only if a similar one is not found. Each session is indexed by its name and is protected with a secret key, therefore prohibiting unauthorised parties to control them. More details regarding the security of this mechanism can be found in the security section below.

The use of sessions has also another advantage. Because it can only be in a finite number of states, it is very easy to implement it's logic as a Finite-State-Machine. This provides great durability and even simpler programming API for the developer. Since the session is fully aware of its current state, the developer has only to point it to the desired target state. The FSM logic can route the request accordingly with no additional input. For instance, the developer has only to call the function `session.start()` in order to start the VM, regardless of its current state. The VM itself might be missing, paused, stopped or already running – it doesn't matter. The session logic will take the appropriate actions in order to bring it on the ''running'' state.

2.2. External Resources

The entire logic of the *CernVM WebAPI* is located in the user's computer (client-side), however some metadata must be requested from external resources. In figure 1 you can see the data flow between all the components involved. The client-side components are the web browser that runs the desired application and the *CernVM WebAPI* daemon [6]. If the daemon is not installed in the system, the user is prompted to download and install it from CERN (a). Upon initalization, the daemon will contact CERN again and fetch additional metadata in order to identify and install a possible missing hypervisor and/or other components (b). If required, the *CernVM WebAPI* will contact the hypervisor website in order to download its installer binary or other resources (c).

When the daemon is requested to open a new controlling session, it will first contact the application website to fetch the configuration details (d) and it will then validate the response

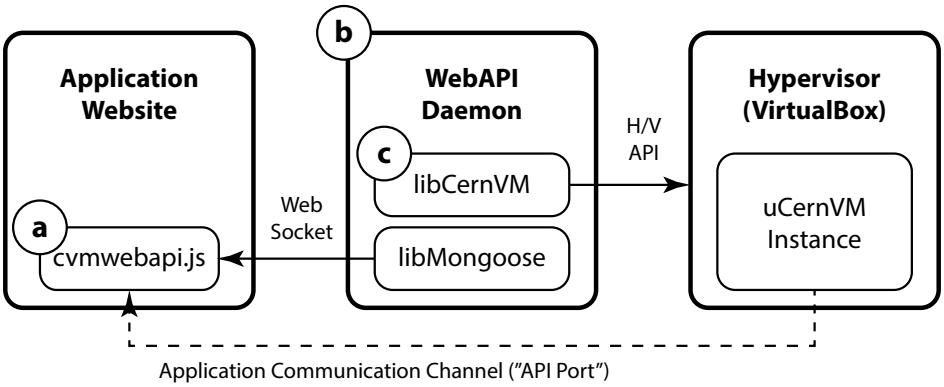

Figure 2. *CernVM WebAPI* Client-Side Components: (a) The high-level javascript library that is included by the web application, (b) The daemon process that runs in the user's computer, (c) The hypervisor interface and abstraction library.

against a whitelist of authorised domains, hosted at CERN.

2.3. Client-Side Components

As mentioned before, we wanted the entire user experience to be focused in the web browser. However, due to the enhanced security features and sandboxing in the modern web browsers, the task of communicating with the hypervisor from the browser is a challenging process.

Originally, the *CernVM WebAPI* system was implemented as a Netscape Plugin API (NPAPI) browser extension, which was cleanly integrated in Firefox and Chrome and provided the desired interface to higher-level system commands. However soon, Google decided to drop NPAPI support from Chrome due to its ''insecure'' nature [7]. In fear of other browsers following [8], we decided to use a different mechanism for escaping the sandboxed browser environment.

We tested various solutions and we concluded that the simplest, cross-platform and cross-browser solution was to use WebSockets to communicate with an external process. That process would then handle the lower-level operations (figure 2). However, this means that the user had to follow at least one installation process. Since this additional installation process could be a potential point of confusion for the end-user we minimised the overall duration and the user's input.

We abstracted the complex process of installation and communication with the external process in a javascript library that complements the daemon (cvmwebapi.js). This library takes care of most of the heavy-lifting for the developer, enabling full integration to any project with just a few lines of code.

The external process runs a tiny web server with WebSocket support (provided by the libMongoose [9] library), listening on the localhost address. Its sole feature is to pass the commands from the javascript interface to the libCernVM [10] library that takes care of all the low-level operations. This library takes care of detecting, downloading, installing and configuring the hypervisor and once it's ready starting and controlling the VMs. Most of its code has been reclaimed from the previous, NPAPI version.

2.3.1. The cvmwebapi.js Library takes care of the low-level communication between the application website and the daemon process. However, in addition to its *interface* logic, it

also makes sure that the application is installed in the user's computer. This *installation*, along with the *interface* logic will be explained with some examples, denoting the simplicity of the code requirements:

Upon including the library, the `CVM` namespace is introduced to the global scope. To request an API access to the daemon process you will need to call the `CVM.startCVMWebAPI` function, as illustrated below:

```
CVM.startCVMWebAPI(
   function(api) { }
);
```

This will trigger the *Interface* logic, that will try to establish a connection to the known websocket endpoint. Upon failing to do so, it will trigger the *Installation* logic, that will inject a the appropriate DOM elements in the website and guide the user through the installation process.

The installation guidelines are customised for every operating system / browser configuration and explains the step-by-step actions the user has to take in order to download the installer and execute it. Upon completion, the daemon will be started and the *Interface* logic will therefore succeed on it's attempt to reconnect. It will then take over, clean-up the injected DOM elements and perform an initial handshake. It will then fire the callback function passed as an argument in the `CVM.startCVMWebAPI` function, indicating that the API channel is now established and ready to use. Now it's possible to request a VM session using the `api.requestSession` like so:

```
CVM.startCVMWebAPI(
   function(api) {
      api.requestSession("http://example.com/vmcp.cgi?vm=1", function(session) {
         // Now you can use your session in the 'session' variable.
      }
   }
);
```

The first argument of the `CVM.requestSession` function is a URL to contact in order to fetch the configuration information for the Virtual Machine to create or resume. This information is signed with an authorized private key in order to prohibit unauthorized use of the *CernVM WebAPI* technology. You can refer to the *VMCP* sub-section of the security section below for more details.

Both `api` and `session` instances fire a set of callbacks which can be used in order to monitor the progress of the request. In addition, the `session` instance, fires additional callbacks when the session state has changed or when the designated *API Port* becomes available or unavailable.

2.3.2. libCernVM Library The *libCernVM* library is a toolset for interfacing with any hypervisor available in the user's computer. It is not strictly bound to the *CernVM WebAPI* technology and can be used in any other project. It provides an abstraction in two levels, in the *hypervisor* and in the *VM instance* level, while in the same time it offers a modular design to minimise the effort of future additions.

The hypervisor abstraction is achieved by reducing the interface operations in just two: ensure hypervisor integrity and create/resume a session. The first checks its configuration and if something is wrong takes the appropriate corrective actions and the latter one creates a new VM or opens the control channel to an existing one.

Each VM instance in a wrapped in a session, as explained in the section 2.1. *libCernVM* provides the FSM engine and the skeleton for implementing the session abstraction in any hypervisor.

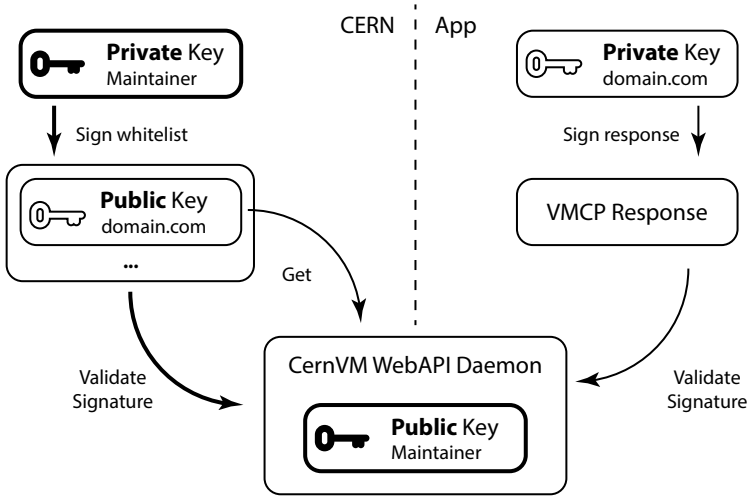

Figure 3. *CernVM WebAPI* Public-Key Infrastructure and trust.

3. Security

Performing low-level system operations, such as controlling Virtual Machines from the browser, introduces a considerable security risk. Therefore, extra care must be taken in order to ensure that the appliance is not launched by an untrusted source.

In order to address these security concerns, the *CernVM WebAPI* system introduces three layers of security:

(i) **Coding-level security**: Ensure no system command is executed, other than the hypervisor Command-Line Interface, and ensure no user-input reaches a system command un-sanitized.

(ii) **Session-level security**: Protect VM sessions so unauthorized parties cannot gain control.

(iii) **Domain-level security**: Ensure that only trusted domains can use this technology.

For the coding-level security, we ensured that the only system command executed is the hypervisor CLI. In addition, we made sure that no user input reaches that command, allowing only numerical values within a specific boundary and only after pre-processing by a sanitisation function.

For session-level security, we are using a *shared secret* between the *CernVM WebAPI* system and the web application that started it. If upon requesting a new session one with the same name exists, the *CernVM WebAPI* is going to check if the secrets match. If not, the request is denied.

For domain-level security, we are using a public-key infrastructure (PKI). A whitelist of trusted domains is published in a trusted server, against which the session requests are validated. The mechanism that was implemented in order to ensure the domain integrity is referred to as *Virtual Machine Configuration Point* and is explained below.

3.1. Virtual Machine Configuration Point (VMCP)

Since the information required in to boot the VM contain information about the application to run, it's important to ensure that they cannot be changed maliciously. For the same reason, it's important to trust only validated sources. Therefore *CernVM WebAPI* must validate the authenticity of each request, which implies that each request is signed. To accommodate

this requirement, the system enforces the VM configuration to be delivered by an authoritative endpoint, called *Virtual Machine Configuration Point* or *VMCP*.

The VMCP is a regular HTTP/HTTPS endpoint, accessible by its URL. *CernVM WebAPI* will contact it directly, using it's own http library (cURL) and not through the browser, ensuring the security of the transport channel. The response is a JSON object that describes the VM properties, including a signature field. The latter is an RSA-SHA256 signature generated using the private key dedicated on the domain the request originates from. The library will also download the domain whitelist from CERN, validate its integrity and then locate the public key that corresponds on the originating domain (figure 3). Upon successfully validating the response, the library will grant access on the requested resources.

An example of a VMCP response is the following:

```
{
    "name": "DemoVM01",
    "secret": "s3cr3tk3y",
    "memory": 512,
    "vcpus": 1,
    "cernvmVersion": "1.18-2",
    "userData": "[amiconfig]\nplugins=cernvm\n[cernvm]\nusers=user:users;password",
    "signature": "01ba4719c80b6fe911b091a7c05124b64eeece964e09c058ef8f9805daca546b"
}
```

In order to protect against replay attacks, along with the request the library passes a random `salt` value, that should be included in the signature calculation.

3.2. HTTPS restrictions

Even though WebSockets are very compatible between platforms and browsers, there is a serious limitation. Because of Same-Origin Policy [11] it's not possible to mix `http:` and `https:` resources, and therefore it's not possible to use *CernVM WebAPI* under a website served under HTTPS. That's because the embedded web server in *CernVM WebAPI* cannot serve content under HTTPS, because a valid, third-party SSL certificate cannot be issued for the ''localhost'' domain. Therefore any request originating from an `https:` towards the local `http:` web socket will fail.

Unfortunately there is no off-the-shelf solution to this problem. Thankfully it is still possible to use a mix of these two protocols with a caveat: Whilst in the `https` domain, the user expresses his/her interest on starting a VM. The server prepares all the information required by *CernVM WebAPI* and stores it in a key/value store, under a random key name. It then opens a pop-up window pointing to a publically accessible `http:` url, that includes the key name as part of the request. *CernVM WebAPI* can be then used on that window normally. This solution is already in use on the CernVM-Online dashboard (https://cernvm-online.cern.ch).

4. Use Cases

The *CernVM WebAPI* technology was field-tested in a pilot event called ''CERN 60 Computing Challenge'' [12]. During this challenge, we asked people from around the globe to start a Monte-Carlo simulation in their computers, trying to get an aggregate virtual event rate comparable to a typical CERN LHC experiment. We designed a clean, informative web interface and created a dashboard for the *CernVM WebAPI* .

The challenge ran for 15 days, through which we reached out to 8,000 unique visitors. Out of them, 5,400 managed to successfully boot the challenge VM without any problem, while 1,100 people had a blocking issue. By processing the analytics information collected during the challenge we realised that about 80% of them were using a poor-quality internet connection.

CernVM WebAPI has also been used in the CernVM-Online dashboard where it is used for testing a context definition.

4.1. The Power of Analytics

For the challenge, we decided to use Google Analytics (GA) for tracking the behaviour of our users. Since *CernVM WebAPI* is embedded in the web page, it required no effort to forward its events directly to GA. We were surprised to see the new horizons this technology has opened: It was very easy with GA to isolate the hypervisor or *CernVM WebAPI* errors and to correlate them to other metrics, such as the provider, its performance, geographic location and demographics. As a result, by the end of the challenge we had very detailed information on which features to improve.

5. Conclusions and future plans

In this paper we explained how the *CernVM WebAPI* technology makes it easy for people without any prior experience with computers to join a volunteer computing project based on virtualization. We explained how it works behind the scenes and how it ensures that no malicious activity can take place in a volunteer's computer. We also demonstrated its power in a real world scenario.

The technology is already in a stable state, however there are a couple of improvements planned for the near future. To start, we would like to add support for other major hypervisors other than VirtualBox, in order to use the most ''natural'' hypervisor for each platform. Moreover, we would like to address the cross-origin request restrictions in order to allow it to operate seamlessly on `https:` domains. Finally, we would like to add a graphical dashboard from which the user can see and control the local instances.

References

[1] Sanchez C A, Blomer J, Buncic P, Chen G, Ellis J, Quintas D G, Harutyunyan A, Grey F, Gonzalez D L, Marquina M, Mato P, Rantala J, Schulz H, Segal B, Sharma A, Skands P, Weir D, Wu J, Wu W and Yadav R 2011 *Journal of Physics: Conference Series* **331** 062022 ISSN 1742-6596 URL `http://iopscience.iop.org/1742-6596/331/6/062022`

[2] Blomer J, Aguado-Sánchez C, Buncic P and Harutyunyan A 2011 *Journal of Physics: Conference Series* **331** 042003 ISSN 1742-6596 URL `http://iopscience.iop.org/1742-6596/331/4/042003`

[3] Harutyunyan A, Blomer J, Buncic P, Charalampidis I, Grey F, Karneyeu A, Larsen D, González D L, Lisec J, Segal B and Skands P 2012 *Journal of Physics: Conference Series* **396** 032054 ISSN 1742-6596 URL `http://iopscience.iop.org/1742-6596/396/3/032054`

[4] Marosi A, Kovács J and Kacsuk P 2013 *Future Generation Computer Systems* **29** 1442–1451 ISSN 0167-739X URL `http://www.sciencedirect.com/science/article/pii/S0167739X12000660`

[5] Anderson D 2014 A Brief History of BOINC URL `https://boinc.berkeley.edu/trac/attachment/wiki/WorkShop14/workshop_14.pdf`

[6] Charalampidis I 2015 wavesoft/cernvm-webapi URL `https://github.com/wavesoft/cernvm-webapi`

[7] Google 2013 Saying Goodbye to Our Old Friend NPAPI URL `http://blog.chromium.org/2013/09/saying-goodbye-to-our-old-friend-npapi.html`

[8] Weiner C 2014 Update on Plugin Activation URL `https://blog.mozilla.org/security/2014/02/28/update-on-plugin-activation/`

[9] Cesanta S 2015 libMongoose URL `http://cesanta.com/libmongoose.shtml`

[10] Charalampidis I 2015 wavesoft/libcernvm URL `https://github.com/wavesoft/libcernvm`

[11] Zalewski M 2015 Part2 - browsersec - Browser Security Handbook, part 2 - Browser Security Handbook - Google Project Hosting URL `https://code.google.com/p/browsersec/wiki/Part2`

[12] Charalampidis I 2014 Cern 60 Computing Challenge URL `http://test4theory.cern.ch/challenge`