# EXPMON System Web UI and APIs

If you haven't read our document regarding EXPMON's Methodology and Architecture, we highly recommend you read it first; it will help you understand the system and use the system more effectively.

After successful deployment, we are now ready to use the system.

Like other similar systems, there are two interfaces for the input and output of the EXPMON system: the web UI and the web APIs. The two interfaces are, in fact, quite simple and easy to use.

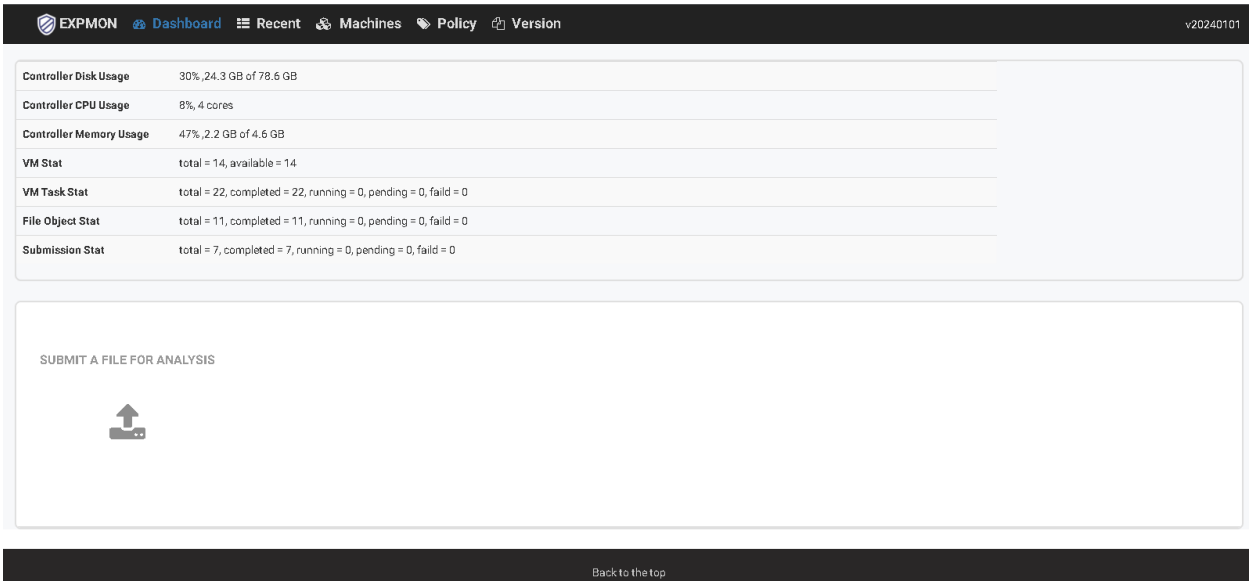
For the online EXPMON Public service, please replace the Controller url with “https://pub.expmon.com”.

We've released a helper tool named "expmon_sample_submit.py", which assists users in automatically submitting samples to the EXPMON system and obtaining analysis results, via the Web APIs. Please check it out on our GitHub repository https://github.com/EXPMON/PubTools.

## The Web UI

### *Home/Dashboard Page*

With the correct IP and port of the Controller, we could use any modern browser to open the web UI, for example, http://172.16.1.200:8000. The Home/Dashboard looks like the following:



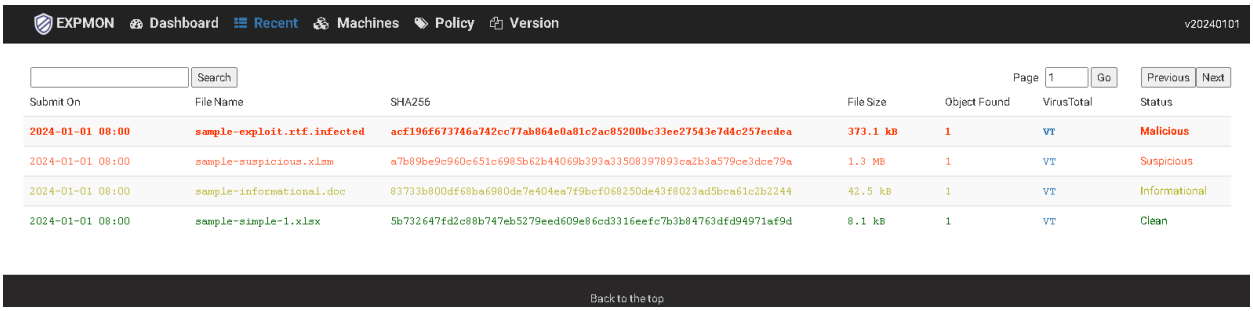

First, you can view information about the Controller machine, such as "Controller Disk Usage," "Controller CPU Usage," and "Controller Memory Usage." This information provides insights into the current status of the Controller machine. Next, we have "VM Stat," "VM Task Stat," "File Object Stat,"

"Submission Stat," etc., providing information on the current status of the running EXPMON system. For instance, with "VM Stat," you can determine how many VMs are currently available in the VM pool.

Below the stats information, there's a button labeled “SUBMIT A FILE FOR ANALYSIS”, this is where you click to manually submit samples. This process is straightforward, similar to many other systems with a similar functionality.

***Recent/Analysis Page***

Next, let's click on "Recent" to navigate to the analysis page.




This is where you find the analysis results, presented in a list containing information about all submitted samples, whether they were submitted via the web or API. The samples are listed in the order of submission time.

The information includes:

- **Submit On**: The time when the sample was submitted (in UTC).
- **File Name**: The original submission name.
- **SHA256**: The SHA256 hash of the submitted sample.
- **File Size**: The file size of the submitted sample.
- **Object Found**: Indicates how many objects were found during the static analysis process. Refer to our "Methodology and Architecture" document for further clarification.
- **VirusTotal**: This provides a simple link to VirusTotal for convenient checking of detections on VT for the same sample. Clicking "VT" opens a new browser tab for easy access to VirusTotal.
- **Status**: The current status of the analyzing process of the sample.

For "Status," please note that Status is not refreshed automatically, you need to manually refresh/click the “Recent” page via Web UI. The value of Status could be one of the followings:

- **pending**: The sample is submitted, but it is pending analysis (usually indicating that all VMs are fully occupied).
- **running**: The sample is currently being analyzed by the system.
- **UNSUPPORTED**: The sample file type is not supported by the system, and no supported file object has been found.

Following are the "detection result" statuses reported by the system after the samples have been analyzed. There are four levels, ranging from the least risk to the most:

- **Clean**: No exploit-related threat found.

  We want EXPMON users to understand that even if it reports "Clean," it doesn't necessarily mean it's definitively not malware. Here are the reasons:

  1. As explained in our "Methodology and Architecture" document, EXPMON is an exploit-focused system and doesn't deal with or detect typical malware. Its primary focus is on exploits, especially zero-day/unknown advanced exploits. Therefore, it could be malware, but EXPMON, being an exploit-focused system, might not detect it.

  2. It could still be an exploit, but not for the sandboxing environments we tested, so EXPMON might not detect it. Hence, it's crucial to deploy VMs that match your real IT environments. For example, if a .pdf exploit is not designed for Adobe Reader or Foxit Reader but for some other PDF software, detection might not occur in the standard deployment of the system.

     It could also be an exploit targeting a vulnerability only present in an older environment than the VMs used. In such situations, because the exploit couldn't execute in a relatively new IT environment, there is typically no significant cause for concern even this is a real exploit.

  Nonetheless, we encourage users to report False Negative findings (real exploits that our system didn't detect) so that we can manually test and improve the system.

- **Informational**: This level indicates that the system has found something mildly "interesting" that we believe is necessary to show to the user. However, the sample is not a real exploit according to the system. It won't cause direct harm to end-users if they follow best practices – for example, not enabling macros on Office files. Users should bear in mind that this information pertains to the sandboxing VM environments they have deployed.

- **Suspicious**: This level is higher than Informational. If a sample is detected at this level, the system has indicators suggesting that the sample is a potential exploit, but there is no decisive proof confirming it as definitively malicious. We recommend users be cautious when a sample is detected at this level, especially if it is delivered from an untrusted source.

- **Malicious**: This indicates that there is proof showing that the sample is an exploit-related threat. This is the highest level of threat detection.

For Informational, Suspicious, and Malicious levels, in addition to the Detection Results, the Detection Details are provided. Usually, it's one (or sometimes multiple) lines of strings describing the threat the system found and/or what actions should be taken. For example:

- Informational - file contains office macros, do not enable macros unless the file is trusted
- Suspicious - malicious office macros detected (credit: mraptor)
- Malicious - potential historical exploit, please check

Note: If the system detects a zero-day exploit, it will display the keyword "zero-day" in the detection details. However, the detection level remains Malicious. For example:

- Malicious - unclassified exploit, please check for potential zero-day

On the top left of the list, there's a simple search field and button where you can input keywords and search for results. The function, currently, will search in combined “File Name”, MD5/SHA1/SHA256 hashes of the submitted sample, Status name, Detection Result, as well as the Detection Details fields. For example, you could search for the word "zero" to find all the samples detected as potential zero-day exploits, as the keyword "zero-day" would be shown in the Detection Details field.

***Analysis Details Page***

When the system finishes analyzing a sample, it becomes clickable on the Detection Result words - the Clean/Informational/Suspicious/Malicious word. If you click on that, you will be led to a page showing all analysis details.

Following is a part of an Analysis Details page:

| | |
|---|---|
| **Detection Overall Result** | Malicious |
| **Detection Overall Detail** | potential historical exploit, please check |
| **File Object Count** | 1 |
| **File Object 1** | |
| **MD5** | d1e8c1498d56e7da0e8fc0671b5ab2b5 |
| **SHA1** | 8276facc6ef40df6385dd3dd49e41848228c7735 |
| **SHA256** | acf196f673746a742cc77ab864e0a81c2ac85200bc33ee27543e7d4c257ecdea |
| **File Type** | rtf |
| **Page Count** | 1 |
| **Analysis Start Time** | 2024-01-01T08:00:15 |
| **Analysis Finish Time** | 2024-01-01T08:04:06 |
| **Object Detection Result** | Malicious |
| **Object Detection Detail** | potential historical exploit, please check |
| **Tested Env Count** | 2 |
| **Tested Env 1** | winx64(update20230925)_office2021x64(16827.20080)[word2021x64] |
| **Tested Env 2** | win7sp1(updated)_office2010(14.0.7208.5000)[word2010] |
| **Indicators** | crash event found |
| | crash found |
| | office crash handler detected |

As the example shows, the information presented aligns with the concept discussed in the "Methodology and Architecture" document. One submitted sample may be extended to one or many file objects, and each file object will be analyzed in multiple sandboxing environments. Each file object has its own Detection Result and Detection Details. Essentially, it's a file object-based detection, and the overall/final detection result is the highest detection level for all the file objects.

For each sandboxing environment, a field called "Indicators" is displayed to users if it's available/detected. Indicators signify that we detect something which could be related to exploit-related activities. However, it's important to note that finding an indicator doesn't necessarily mean it is an exploit or malicious. Our core logic module primarily detects exploits based on the indicators, and we believe it's valuable to show them to users to assist potential manual analysis/investigation.

One file object for one environment could produce more than one Indicator. Indicators are listed as a list of strings and are highlighted in red.

**The Web APIs**

In addition to the Web UI, EXPMON also provides Web APIs to submit samples and query the analysis results. We recommend using our APIs to integrate this cutting-edge system into your existing cybersecurity solutions, making it a great addition for detecting advanced zero-day/unknown exploits.

The APIs can perform all the previously-discussed functions that the web UI offers. Beyond that, the APIs can do more things - while querying the result, the server will return the raw/meta sandbox data.

The Web APIs are REST-based APIs; currently, we offer two: one for submitting samples and the other for querying the results. They're fairly easy to use and straightforward. Below, we will quickly go through the two APIs.

- http://ip:port/submit/api/expmon_submit_file

  Description: submit a sample
  Python example for how to use:

  ```
  url_submission = 'http://172.16.1.200:8000/submit/api/expmon_submit_file'
  files = {'file_data': (sample_name, open(sample_fullname, 'rb'))}
  response = requests.post(url_submission, files = files, verify = False, timeout = 60)
  resp = response.json()
  submit_sha256 = resp['sha256']
  submit_uuid = resp['uuid']
  ```

  It returns two fields: one is the SHA256 for the submitted sample, and the other is a unique UUID for the submission. Keep these two fields, as we will need them for querying the detection result later.

- http://ip:port/analysis/api/query/<submit_sha256>/<submit_uuid>

  Description: query the detection result for a submitted sample
  Python example for how to use:

  ```
  url_query_prefix = 'http://172.16.1.200:8000/analysis/api/query/'
  #continue to check until the server returns analysis result
  time.sleep(15)
  while 1:
      response = requests.get(url_query_prefix + submit_sha256 + '/' + submit_uuid + '/', verify = False)
      resp = response.json()

      #code 0 means we get the result
  ```

```
    if resp['code'] == 0:
        break
    #code 1 means sample is being analyzed
    elif resp['code'] == 1:
        print('[INFO] Sample is being analyzed, wait additional 15 seconds')
    elif resp['code'] == 2:
        print('[INFO] Sample is pending to be analyzed, wait additional 15 seconds')
    else:
        print('UNKNOWN ERROR: %s' % resp['message'])
        exit(-1)

    time.sleep(15)
```

After successfully submitting the sample and obtaining the 'submit_sha256' and 'submit_uuid', we may now query the results.

As this is sandbox-based analysis, the jobs won't be finished immediately. There's no need to query the results too quickly. Instead, as shown in the above sample code, we recommend querying the result every 15 seconds until we get the result ('code' == 0). This way, we will save resources.

In addition to all the information that could be returned by the Web UI, Web APIs can return the raw/meta sandbox analysis data/logs. You may follow the example code below to parse the returning fields:

```
#print overall detection result
detection_obj = json.loads(resp['detection'])
print('Detection Result: ' + detection_obj['result'])
print('Detection Description: ' + str(detection_obj['desc']))

#print detailed analysis result based on file objects
file_objects = resp['file_objects']

i = 0
for file_obj in file_objects:
    i = i + 1
    print("file object %d:" % i)
    print("\t md5: %s" % file_obj['md5'])
    print("\t sha1: %s" % file_obj['sha1'])
    print("\t sha256: %s" % file_obj['sha256'])
    print("\t file type: %s" % file_obj['file_type'])
    print("\t page number: %d" % file_obj['page_num'])
```

```
print("\t object analysis start time: %s" % str(file_obj['analysis_start_time']))
print("\t object analysis finish time: %s" % str(file_obj['analysis_finish_time']))

object_detection_result = json.loads(file_obj['detection'])
print("\t object analysis result: %s" % object_detection_result['result'])
print("\t object analysis description: %s" % object_detection_result['desc'])

file_analysis_logs = json.loads(file_obj['analysis_logs'])

#make folder to dump analysis logs
obj_folder_name = '%s__%s' % (file_obj['sha256'], file_obj['file_type'])
folder_pathname = os.path.join(folder_root, obj_folder_name)
os.makedirs(folder_pathname)

env_count = 0
for env_name in file_analysis_logs:
    env_count = env_count + 1
    print("\t test env %d: %s" % (env_count, env_name))

    folder_pathname_env = os.path.join(folder_pathname, env_name)
    os.makedirs(folder_pathname_env)

    for log_type in file_analysis_logs[env_name]:
        if log_type == "indicators":
            indicators = file_analysis_logs[env_name][log_type]
            print("\t " + str(indicators))
        else:
            log_data_hexstr = file_analysis_logs[env_name][log_type]
            if log_data_hexstr == '' or log_data_hexstr == None:
                continue

            #log data is hex string in the traffic and is compressed
            log_data = zlib.decompress(codecs.decode(log_data_hexstr, 'hex'))

            #dump log data to file
            log_file_name = os.path.join(folder_pathname_env, log_type + '.txt')
            #print(log_file_name)
            open(log_file_name, "wb").write(log_data)

#print("\t sandbox activity logs dumpped in folders")
```

**Last Updated on April 7, 2024**