Fine penetration tests for fine websites

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

Pentest-Report Cuckoo Sandbox 06.-07.2018

Cure53, Dr.-Ing. M. Heiderich, M. Wege, MSc. N. Krein, N. Hippert, BSc. D. Weißer, BSc. J. Hector, J. Larsson, Dipl.-Ing. A. Inführ

Index

Introduction

> *“Cuckoo Sandbox is the leading open source automated malware analysis system. You can throw any suspicious file at it and in a matter of minutes Cuckoo will provide a detailed report outlining the behavior of the file when executed inside a realistic but isolated environment.”*
>
> From https://cuckoosandbox.org/

This report documents the results of a large-scale, complex and multifaceted security assessment of the Cuckoo Sandbox. The project was carried out by Cure53 in June and July of 2018 and yielded ten security-relevant findings.

In terms of resources, Cure53 approached this project with a dedicated team of eight senior testers. They were allocated a time budget of twenty-five days to complete this

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

assessment. Moving on to methodology, it is unsurprising that the used approach entailed a white-box premise. This is because the Cuckoo project is available as open source software and Cure53 had access to all relevant sources. In addition, detailed instructions and documentation were furnished to the Cure53 testing team by maintainers of the Cuckoo Sandbox in order to better delineate the scope and facilitate execution of the project.

It has been agreed that the Cure53 testers and the in-house team at the Cuckoo Sandbox entities will remain in close contact during the assessment. The teams communicated on Slack and the testers have been providing timely status updates on the progress of the project to the Cuckoo team. What is more, reporting was accomplished on Slack as well. Cure53 introduced and then discussed the emerging findings with the in-house team. Assurance were regularly made that the test activities in fact covered what the Cuckoo maintainers have chosen as the focal points of interest for this assignment.

An important aspect to note is that the scope of this project was slightly more complex than it is usual for a classic penetration test. In particular, the maintainers of the Cuckoo project wanted to find out whether the Cuckoo Sandbox could hold against the scrutiny without compromising on any of its security promises in a specific context. This specific context encompassed the Cuckoo Sandbox being deployed on the Internet and being used as a crowd-sourced malware analysis platform with an additionally attached crypto coin reward system. Since this clearly makes for a challenging task, the assessment's scope was divided into three Work Packages, separately described in the ensuing *Scope* section.

Among the already mentioned ten findings, six constituted vulnerabilities or detection bypasses and four were documented as general security weaknesses. It needs to be emphasized that none of the issues were deemed to signify a *"Critical"*-level problem. Even though one discovery was a seen as a "close-call", it was ultimately a false alert.

In the following sections, the report will first elaborate on the multifarious scope of the project and list all relevant code repositories that the Cure53 team examined and used during this assessment. Next, each finding is discussed in substantial technical detail and with the inclusion of mitigation advice as applicable. The final part of the document is dedicated to sharing a broader verdict on the general security posture and reliability of the Cuckoo Sandbox in terms of the envisioned deployment goals. Cure53 offers general impressions in the *Conclusions* section.

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

Scope

- **In scope for this assessment was the latest version of the Cuckoo Sandbox**
  - The maintainers are interested in attacks against Cuckoo on three different levels. Based on that, Cure53 derived three different work packages for this project.
- **WP1: Security Tests against the components inside the Cuckoo Sandbox VM**
  - Strong focus on spoofing & evasion attacks, vulnerabilities allowing to bypass analysis or deliver faulty results affecting the components outside the Sandbox.
  - Strong focus on secure VM deployment & attacks abusing VM-specific attacks to escape the guest system and attack the host.
  - Strong focus on attacks introduced by maliciously compressed data used for evasion, memory corruption, ZIP-bombs and the like.
- **WP2: Security Tests against Components outside the Cuckoo Sandbox VM**
  - Strong focus on potential vulnerabilities due to the incorrect handling of malicious data.
  - Strong focus on RCE attacks caused by vulnerabilities allowing malware inside the Sandbox to influence the integrity of the results linked to analysis or attacks against the host system via metadata and similar.
  - Strong focus on attacks abusing the URL submission features leading to DoS, RCE, SSRF or similar attack patterns and vulnerabilities.
- **WP3: Web Application & "REST" API Security Tests**
  - General Injection attacks / Code Execution vulnerabilities (server & client).
  - OWASP Top Ten 2017, CSRF and mass-assignment attacks, ACL / privilege escalation tests.
  - SQL Injection & truncation attacks / NoSQL attacks where applicable.
  - Tests against API functionalities, UI spoofing, focus on any form of XSS & UI redressing.
- **Relevant sources:**
  - https://github.com/cuckoosandbox/cuckoo
  - https://github.com/cuckoosandbox/monitor
  - https://github.com/jbremer/sflock
  - https://github.com/jbremer/tracy
  - https://github.com/jbremer/peepdf
  - https://github.com/jbremer/egghatch
  - https://github.com/jbremer/httpreplay
  - https://github.com/jbremer/roach
  - https://files.pythonhosted.org/packages/a1/ab/61489e4acfaa41ffb53b821f649646cd79f5787886aadc30c4058fd6ec3c/pefile2-1.2.11.tar.gz

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

- **SSH server access**
  - Cure53's SSH public keys have been shared with the Cuckoo Sandbox team and access to a test server was granted.

Identified Vulnerabilities

The following sections list both vulnerabilities and implementation issues spotted during the testing period. Note that findings are listed in a chronological order rather than by their degree of severity and impact. The aforementioned severity rank is simply given in brackets following the title heading for each vulnerability. Each vulnerability is additionally given a unique identifier (e.g. *CSB-01-001*) for the purpose of facilitating any future follow-up correspondence.

CSB-01-001 Analyzer: Analysis evasion via special characters *(High)*

It was found that the analysis of a file is not executed in case the filename contains certain special characters like <, >, :, *. When such a file is submitted, Cuckoo starts the analysis and this item appears to finish shortly after. However, no analysis is in fact executed and the displayed *results* link leads to a 404 page.

The problem here is that the Windows *guest* does not allow certain characters in filenames, thus the file cannot be uploaded to the virtual machine. The list of characters above is likely not exhaustive. Therefore, to mitigate this problem, it is recommended to hash filenames so as to make sure that only valid characters can be contained.

CSB-01-002 Web: Lack of protections against CSRF *(Medium)*

It was found that the Cuckoo web interface implements no protections against Cross Site Request Forgery (CSRF), thus allowing a malicious website to force the user's browser into performing actions in the panel. It appears that attempts to mitigate this issue were made by adding the *csrftoken* cookie but, since cookies are sent automatically by the browser, this attack is not prevented. A malicious website could have a Cuckoo user perform actions like submitting samples or deleting old reports.

To prevent this issue, it is recommended to add CSRF tokens to all requests that make changes to the database maintained by Cuckoo.

CSB-01-003 Sandbox: Web interface reachable from inside a container *(Medium)*

While analyzing the capabilities of the virtual machines, it was discovered that the web interface of Cuckoo can be reached from the inside of the Sandbox. This could allow a malicious website or malware to *stop/delete* the analysis, potentially bypassing the detection of malicious code. The following screenshot displays the results for a scan of

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

the internal website running at *192.168.122.1:41501*. Although the interface is not displayed correctly, it is evident that the page can be accessed. Combined with the fact that the web interface has no authentication mechanisms or CSRF tokens in place, simple JavaScript invocation could instruct the web server to delete all reports or execute other administrative actions.

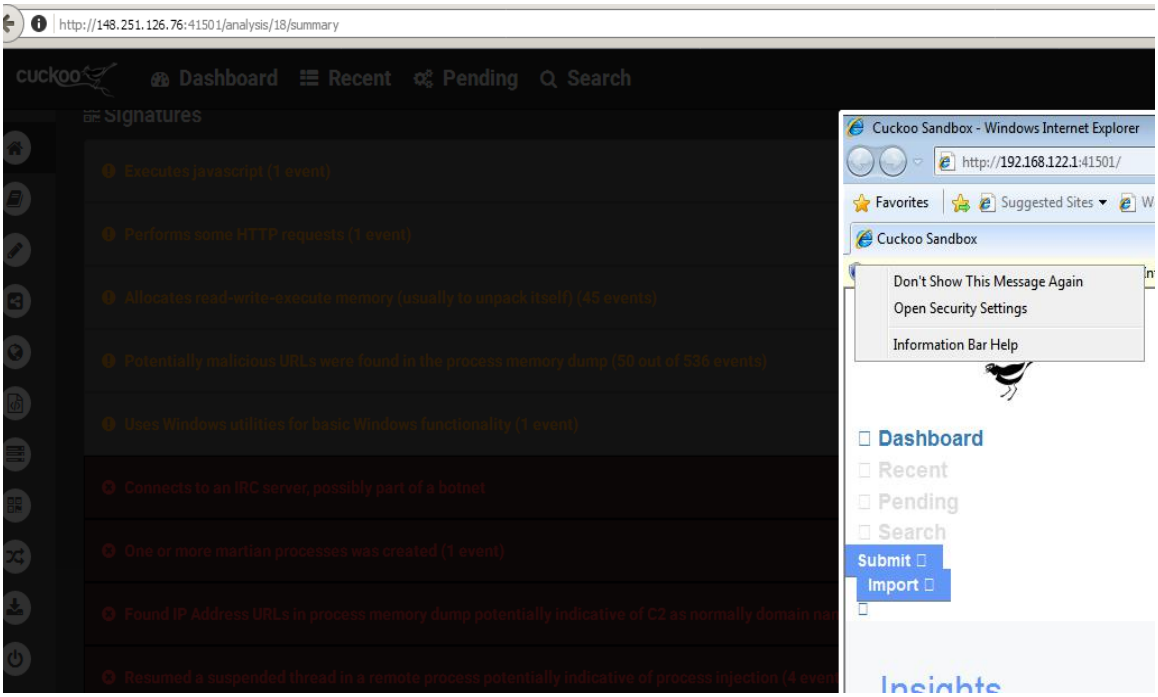

Fig.: Accessing the Cuckoo interface as guest

A quick test showed that it is easily possible to instruct IE to navigate to the Cuckoo web interface and delete all currently linked reports. The following HTML file can simply be hosted on the malicious domain.

PoC.html:

```
<body id = "body">
<script>
	for (var i = 150; i >= 100; i--) {
		var img = document.createElement("img");
		img.src = "http://192.168.122.1:41501/analysis/remove/" + i + "/";
		document.getElementById("body").appendChild(img);
	}
</script>
</body>
```

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

Moreover, if the host's IP is unknown, the PoC can be extended to include all kinds of subnets. It is recommended to isolate the virtual machine from the entire local network. This cannot only rely on the separation from the Cuckoo panel since potentially malicious code simply should not be able to access internal systems.

CSB-01-005 Sflock: PDF attachments are not properly detected (*Low*)

The Cuckoo web interface utilizes *sflock* to extract file resources from the archives when one of them is uploaded for analysis. One of the supported file types is PDF. *Sflock* uses *peepdf* to parse PDFs and extract any kind of PDF attachments. It was discovered that not only are PDF file attachments not properly supported but the detection can be bypassed completely.

The *make-pdf-embedded*[1] tool from Didier Stevens was used to embed a file as an attachment in a PDF. After uploading the PDF, the embedded file is detected and displayed by the GUI. As soon as the user is trying to analyze the attachment, an error is returned. The reason behind this is that *peepdf* is able to extract the filename but not the contents of the attachment.

File:
sflock-master/sflock/unpack/pdf.py

Code:
```
def unpack(self, password=None, duplicates=None):
       [...]
       obj = f.body[version].objects[ref.id]
       // contents will be empty
       contents = obj.object.decodedStream
       filename = filename.value
```

It could not be verified why this issue persists as the command line version of *peepdf* is able to parse the stream properly.

Additionally, it was discovered that the detection of the attachments inside PDFs can be completely bypassed. After *sflock* parsed the PDF in question with *peepdf,* it was trying to detect if a PDF object contained the */F* key responsible for specifying the filename. Afterwards, the presence of the */EF* has been verified and this item specifies the contents of the file. This approach works for most of the *FileSpec* objects.

File:
sflock-master/sflock/unpack/pdf.py

[1] https://blog.didierstevens.com/programs/pdf-tools/

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

Code:

```
def unpack(self, password=None, duplicates=None):
      [...]
      for version in range(f.updates + 1):
            for obj in f.body[version].objects.values():
                  if not isinstance(obj.object, peepdf.PDFCore.PDFDictionary):
                  continue
                  # Trying to detect the /F Key in FileSpec
                  if "/F" not in obj.object.elements:
                  continue
                  if "/EF" not in obj.object.elements:
                  continue
```

Standard FileSpec object:

```
7 0 obj
<<
/Type /Filespec
/F (what.pdf)
/EF << /F 8 0 R >>
>>
endobj
```

It is possible to avoid the use of the */F* key without breaking the attachment functionality in a standard PDF reader, therefore bypassing the detection.

Modified FileSpec object:

```
7 0 obj
<<
/Type /Filespec
/DOS (what.pdf)
/Mac (what.pdf)
/Unix (what.pdf)
/UF (what.pdf)
/EF << /F 8 0 R >>
>>
endobj
```

It should be taken into consideration to deep-dive into the library so that it can be detected and determined why the command line tool, which seems to use similar code to the currently deployed Python code, is able to extract the attachment payload but the Cuckoo sandbox is unable to do so. Moreover, the currently deployed check needs to be modified to detect the presence of the */UF* key as well. The latter will ensure that a malicious PDF cannot bypass the attachment check.

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

CSB-01-008 Sflock: File enumeration bypass in RAR archive via symlink (*Info*)

The *sflock* library supports RAR archives and is trying to extract resources of any kind. It was discovered possible to break the extraction process by including a symlink which traverses down the file system. As soon as the RAR program attempted to extract the symbolic link, the *zipjail* executable detected a system call and killed the process.

Zipjail exception:

```
Extracting  /tmp/link_test 23%Blocked system call occurred during sandboxing!
ip=0x4a8ec7 sp=0x7ffdb55a3998 abi=0 nr=88 syscall=symlink
Killing child 22565
```

File:
bypass.rar

Content:

```
tttttevil.exe
test.txt
link_test
```

File:
link_test -> ../../../../../../../../etc/passwd

It should be taken into consideration to check the output of the *zipjail* process to determine if any exceptions happened. These could then be displayed in the GUI to inform the user about the matter.

CSB-01-009 Web/API: Unauthorized access due to missing authentication (*High*)

Neither the web interface nor the API server deploys any form of authentication. This enables full access to the entire functionality provided by Cuckoo to nearly everyone. The only precondition is to know the IP address of the web interface/API server. For the API server, having an API access token, which is usually sent along as a request header, would also go a long way in helping to prevent CSRF attacks as mentioned in CSB-01-002.

It is recommended to implement some form of authentication for the web interface as well as the API server. For instance, this could be done with a standard session management with a regular *username/password* login.

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

Miscellaneous Issues

This section covers those noteworthy findings that did not lead to an exploit but might aid an attacker in achieving their malicious goals in the future. Most of these results are vulnerable code snippets that did not provide an easy way to be called. Conclusively, while a vulnerability is present, an exploit might not always be possible.

CSB-01-004 Web: ZIP-bomb leads to Denial of Service of Cuckoo service (*High*)

While uploading malicious files through the *upload* function published on the Cuckoo dashboard under *"Submit a file for analysis",* it was unveiled that a crafted ZIP-bomb causes the Cuckoo service to become unresponsive. This item eventually crashes the entire instance.

Excerpt from */var/log/kern.log*:

```
Jul  3 15:32:35 kookoo kernel: [17406077.996867] Out of memory: Kill process
30397 (python) score 937 or sacrifice child
Jul  3 15:32:35 kookoo kernel: [17406077.996903] Killed process 30397 (python)
total-vm:50637752kB, anon-rss:31369088kB, file-rss:8kB
```

In order to ensure that the uploaded ZIP file will not deplete all available system resources upon extraction, it is recommended to implement a ZIP parser that has the capability to analyze and verify all uploaded ZIP archives.

CSB-01-006 Sandbox: File dropping detection bypass via ADS (*Info*)

Normally, the Cuckoo Sandbox detects if the application drops files to the local filesystem. However, it was discovered that this detection can be bypassed. First an application creates a new folder which ends with ". .". Afterwards, it can drop any files into this newly created item without it being detected by the Cuckoo Sandbox. The reason behind this is that the Windows API is removing ". ." for certain system calls in case it is found in a specified path. This most likely confuses the detection.

It must be mentioned that the Sandbox properly detects that an alternative data stream is used.

File:
Bypass.bat

Code:

```
echo 123 > "test3. .::$INDEX_ALLOCATION"
cd "test3. .::$INDEX_ALLOCATION"
echo "secret" > dropped.exe
dir
```

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

It should be taken into consideration to detect if a file *path/filename* ends with ". .". As no benign application creates these kinds of resources, a warning could be displayed to the user as soon as this behavior is detected.

CSB-01-007 Event: Detection bypass on Windows startup folder installation (*Info*)

In case the analyzed file drops a file in the startup folder, the GUI will display a red event notification about this behavior in the report summary. It was discovered that NTFS directory links can confuse the detection and therefore this event is not displayed in the report summary. It must be mentioned that the Sandbox still properly detects that a file is dropped, it is only that the warning is not displayed.

File:
bypass.bat

Code:
```
mklink /J test "C:\Users\%USERNAME%\AppData\Roaming\Microsoft\Windows\Start
Menu\Programs\Startup"
echo 123 > "test\drop_wupp4446.hta"
dir "C:\Users\%USERNAME%\AppData\Roaming\Microsoft\Windows\Start
Menu\Programs\Startup"
```

Currently, the Cuckoo Sandbox seems to have some problems to properly detect NTFS directory junctions. It could be taken into consideration to detect NTFS junctions and include an additional event which is displayed in the report summary.

CSB-01-010 Web: Archive-bombs lead to Denial-of-Service (*Medium*)

While uploading malicious files through the *upload* function published on the Cuckoo dashboard under the "*Submit a file for analysis*" option, it was discovered that a crafted archive-bomb (e.g. RAR archive) causes the Cuckoo service to become unresponsive. The archive files in question are comparatively small files[2] which once extracted take up over 100GB of disk space. This can quickly lead to the exhaustion of the available space and deny any further analysis.

Excerpt from the console:
```
root@kookoo ~ # ps auxw | grep rar
/home/cure53/cuckoo/local/lib/python2.7/site-packages/SFlock-0.3.5-
py2.7.egg/sflock/data/zipjail.elf /tmp/cuckoo-tmp-cure53/tmp82YvHZ/100GB.rar
/tmp/tmprYcEVY [...]
```

[2] https://bomb.codes/bombs

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

```
/usr/bin/rar x -mt1 -p- /tmp/cuckoo-tmp-cure53/tmp82YvHZ/100GB.rar
/tmp/tmprYcEVY

root@kookoo ~ # du -sh /tmp/cuckoo-tmp-cure53/tmp82YvHZ/100GB.rar
50M    /tmp/cuckoo-tmp-cure53/tmp82YvHZ/100GB.rar
root@kookoo ~ # du -sh /tmp/tmprYcEVY/100GB
89G    /tmp/tmprYcEVY/100GB
```

In order to ensure that the uploaded compressed archives will not deplete all available system resources upon extraction, it is recommended to implement a configurable upper boundary. This will kill the process and cleanup the disk once the threshold is reached[3]. Furthermore, it should be considered to employ *cgroups* to further limit the available memory and CPU footprint that a process can create.

Conclusions

The results of this Cure53 2018 assessment of the Cuckoo Sandbox are rather positive. While the tested scope is not completely free from vulnerabilities and fully secure, the results did not yield any issues that could be seen as a major compromise.

Despite significant efforts from the Cure53 team of eight senior testers, who spent twenty-five days investigating the scope in June and July of 2018, the Cuckoo Sandbox compound held well against various attack approaches. Among the unveiled ten findings, no paramount defect like Remote Code Execution, privilege escalation or virtual machine escape has been verified. This should be seen in context of an intensive and extensive coverage as Cure53 tested a large array of potentially vulnerable routes. Substantial attack surface has been covered, further contributing evidence that the analyzed platform exhibits a praiseworthy level of maturity.

To shed light on some of the most vital findings, the first up for fixing should be the handling of decompression and different file-types inside of the *guest* virtual machines. Room for improvement has been identified particularly with the verbatim application of filesystem names. The partial parsing of PDFs discussed in CSB-01-005, would certainly benefit from a more complete implementation. On the host machine, it could be advantageous to consider an augmented segregation of network spaces with the goal of achieving a more effective isolation of the *guest* virtual machines.

In another realm, the web user-interface and the programming interface could be improved with the implementation of proper authentication. Mitigation of request forgery, evidenced in CSB-01-002 and CSB-01-003, should be seen as a priority. The remaining risks did not translate to significant patterns and can be viewed as information on

[3] https://bomb.codes/mitigations#archives

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

defense-in-depth strategies to consider. All in all, the total number of issues is acceptable and the severities, with the exception of three flaws ranked as *"High",* point to the majority of modern security risks being handled well by the Cuckoo Sandbox compound.

Cure53 believes it vital to underline that online communications with the Cuckoo development team were excellent, characterized by fluency and lack of hesitation in expressing interests and grasping complexity behind some of the findings. All questions posed by the Cure53 testers during the project were answered swiftly and conclusively. Feedback about the general direction of the tests was furnished accordingly and at different stages of the project. The testing team is impressed with the relaxed professionalism and dedication of the development team.

From a technical standpoint, Cure53 has no major concerns about the Cuckoo Sandbox project's design and deployment. The entire range of dependencies, from system provisioning to test access was either directly implemented or promptly solved. The project documentation proved to be clear and complete so that answers to pressing questions might be quite easily derived. With the mitigation of the reported fixes, the project will definitely display even higher standards of security. To conclude, Cure53 finds the Cuckoo Sandbox entities to be rather mature and recommends the project for dynamic malware behavior analysis in separated networks and on dedicated machines.

Cure53 would like to thank Jurriaan Bremer, Ricardo van Zutphen, Koen Houtman and Alwin Peppels of Hatching[4] for their excellent project coordination, support and assistance, both before and during this assignment.

[4] https://hatching.io/