Dr.-Ing. Mario Heiderich, Cure53
Rudolf Reusch Str. 33
D 10367 Berlin
cure53.de · mario@cure53.de

Pentest-Report PyCIRCLean 05.-06.2017

Cure53, Dr.-Ing. M. Heiderich, BSc. F. Fäßler, BSc. D. Weißer, MSc. N. Krein, Dipl.-Ing. A Inführ

Index

Introduction

"PyCIRCLean is the core Python code used by CIRCLean, an open-source USB key and document sanitizer created by CIRCL. This module has been separated from the device-specific scripts and can be used for dedicated security applications to sanitize documents from hostile environments to trusted environments. PyCIRCLean is currently Python 3.3+ compatible."

From https://github.com/CIRCL/PyCIRCLean/blob/master/README.md

This report documents the findings of a Cure53 assessment of the PyCIRCLean suite. The project, which was carried out over the course of eight days in May and June of 2017, covered the Python library and the implementing CIRCLean tool (i.e. PyCIRCLean deployed on a Raspberry Pi). Five testers from the Cure53 team completed this assessment which, ultimately, yielded nine security-relevant findings.

As for the test's methodology, it was decided that white-box approach should generally be used. Specifically, the Cure53 testers followed the white-box procedures with reference to being granted access to the source code subjected to audit. However, the tests relied on emulators (i.e. QEMU) for the creation of the necessary emulation setup.

The detection of any features pertinent to malicious files constituted the core focus of the test. This clearly stems from the implementation logic, which seeks to ensure secure handling within the library. To fulfill the primary goal, Cure53 invested into crafting files that contain malicious data in a slightly obfuscated manner. By this logic, an exemplary created file would be recognized by LibreOffice and Microsoft Office. Conversely, the PyCIRCLean would not assume the very same file to be safe.

Additional efforts were dedicated to close observation of the general file communication and analysis process. The investigations zoomed in on identifying general and file-unspecific bypasses and weaknesses. Moreover, the code was particularly checked for the presence of issues leading to privilege escalation or Remote Code Execution issues. Given the time constraints and scope of the project, the coverage and aims did not attempt to meticulously examine the code of all libraries included in the tool.

The report proceeds with a case-by-case discussion of the findings. Detailed technical descriptions are paired with mitigation and fix advice. At the end, concluding remarks about the general security level of both the CIRCLean and the PyCIRCLean are offered. The Cure53 team further shares key security-relevant recommendations for future releases in the final section.

Scope

- **CIRCLean**
  - https://github.com/CIRCL/Circlean/
- **PyCIRCLean**
  - https://github.com/CIRCL/PyCIRCLean
  - Python library used by CIRCLean (the USB sanitizer) and others

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. *PCL-01-001*) for the purpose of facilitating any future follow-up correspondence.

PCL-01-001 Bypass: LibreOffice file bypass via File Structure Renaming *(High)*

The PyCIRCLean library parses LibreOffice document files to detect malicious behavior. After unzipping the document to retrieves its internal file structure, PyCIRCLean checks for occurrences of certain file names. This is aimed at pinpointing names which can indicate malicious behavior, for instance structure names starting with the *"object"* string. It was discovered that an attacker could modify the LibreOffice file's structure in specific way. This modification means that the file is still parsed as valid by LibreOffice, but no longer contains any malicious file names, thus bypassing detection.

Steps to reproduce:

- Create a LibreOffice Write file and embed an "*object*" in it;
- Save the file as *filename.odt;*
- Unzip the file;
- Rename the folder "*Object 1*" to "*Asdf 1*";
- Rename all occurrences of "*Object 1*" in all files to "*Asdf 1*";
- Zip all files again and open it in LibreOffice;
- LibreOffice "repairs" the file and permits a *"save"* action;
- The new file no longer contains any folder with the name "*Object 1*" but still parses and displays the embedded *object* in the LibreOffice correctly.

Affected File:
https://github.com/CIRCL/PyCIRCLean/blob/f5cc3d7533e2efa5d79924000f5e51e8e09c6b55/bin/filecheck.py#L353

Affected Code:
```
def _libreoffice(self):
       """Process a libreoffice file."""
       # As long as there is no way to do a sanity check on
              the files => dangerous
       [...]
if fname.startswith('script') or fname.startswith('basic') or
                   fname.startswith('object') or fname.endswith('.bin'):
       self.make_dangerous('Libreoffice file containing executable code')
```

It is recommended to review whether detecting malice by scanning for certain structures and object names is actually the most optimal strategy. Given that LibreOffice happily accepts renamed objects, a better practice could be envisioned. A deeper inspection into the document is necessary to determine potentially harmful objects.

Note: A PoC for this bypass is available at:
https://cure53.de/exchange/04321789437243/bypasses.zip

PCL-01-002 Bypass: PDF XFA structure not detected *(High)*

To be able to identify malicious PDFs, the PyCIRCLean library is trying to detect certain suspicious keys inside the PDF structure. The focus is on keys evoking a sense of risky behavior. However, it was discovered that the currently used approach does not check for the occurrence of any form of XML Forms Architecture (XFA) structure[1]. XFA lets PDFs define certain elements like *buttons* and similar, while it also permits JavaScript execution and invoking of the FormCalc language[2].

Affected File:
https://github.com/CIRCL/PyCIRCLean/blob/f5cc3d7533e2efa5d79924000f5e51e8e09c6b55/bin/filecheck.py#L358

Affected Code:
```
if oPDFiD.encrypt.count > 0:
            self.make_dangerous('Encrypted pdf')
        if oPDFiD.js.count > 0 or oPDFiD.javascript.count > 0:
            self.make_dangerous('Pdf with embedded javascript')
        if oPDFiD.aa.count > 0 or oPDFiD.openaction.count > 0:
            self.make_dangerous('Pdf with openaction(s)')
        if oPDFiD.richmedia.count > 0:
            self.make_dangerous('Pdf containing flash')
        if oPDFiD.launch.count > 0:
            self.make_dangerous('Pdf with launch action(s)')
        # C53: no check for a XFA structure
        if not self.is_dangerous:
            self.add_description('Pdf file')
```

It is recommended to include *oPDFiD.xfa.count* to detect malicious XFA structures[3]. It must be noted that *oPDFiD* is only able to detect this structure if the scanned PDF is decompressed beforehand. This can be achieved by using the *qpdf* tool set.

Note: A PoC for this bypass is available at:
https://cure53.de/exchange/04321789437243/bypasses.zip

[1] https://en.wikipedia.org/wiki/XFA
[2] http://help.adobe.com/en_US/livecycle/es/FormCalc.pdf
[3] https://github.com/DidierStevens/DidierStevensSu...e5a6d3957e9719ef228/plugin_triage.py

Dr.-Ing. Mario Heiderich, Cure53
Rudolf Reusch Str. 33
D 10367 Berlin
cure53.de · mario@cure53.de

PCL-01-003 Bypass: DOCX Bypass via OLE File Recovery (*High*)

It was discovered that a malicious DOCX file can bypass the PyCIRCLean library by relying on the Microsoft Office *File Recovery* feature[4]. As already noted in PCL-01-001, the first step to create a bypassing file is to unzip the file - in this case a DOCX file. Then one shall change the content of the unpacked files before repacking and distributing it. By modifying the "*word/_rels/document.xml.rels"* XML file, the defined objects like "*OLE objects*" are no longer detected by PyCIRCLean. Still, Microsoft Office retains the ability to recover the file properly and may freely display the *OLE object*[5] afterwards.

Steps to reproduce:

1. Create a DOCX file, which embeds an *OLE object;*
2. Modify the *word/_rels/document.xml.rels* structure:

```
<Relationship Id="cure53" Type="cure53" Target="NULL"><Relationship
Id="rId5"
Type="http://schemas.openxmlformats.org/officeDocument/2006/relationships
/oleObject" Target="embeddings/oleObject1.bin"/></Relationship>
```

3. Open the modified office file in Microsoft Office;
4. An Dialog will popup stating that "*The document cannot be opened*";
5. A second dialog opens with a question: "*Do you want to repair the file? Yes/No?*";
6. After clicking on *Yes*, the file will be rendered correctly and the *OLE object* is displayed.

It is recommended to work with a *canary* here and drop a file in the *root* directory of the destination USB key. This must take place as soon as any Microsoft Office file is copied. In particular, this file should contain information describing the procedure, namely that the user needs to stop the process as soon as Microsoft Office is trying to recover the file. Only this way it is possible for the users to protect their system. Warning the user seems to be the only way to handle this bypass as it is unlikely that PyCIRCLean will be able to emulate the entire *file recovery* process and somehow prevent all possible bypasses.

The *recovery process* available in Microsoft Office is surprisingly creative as far as obtaining one's lost files is concerned. There are many routes to recovering files, which in turn means that making a determination as to whether an Office file is safe, broken or manipulated is a nearly impossible feat for any library.

Note: A PoC for this bypass is available at:
https://cure53.de/exchange/04321789437243/bypasses.zip

[4] https://support.microsoft.com/en-us/help/316951/how-to-recover-a-lost-word-document
[5] https://msdn.microsoft.com/en-us/library/office/ff838421.aspx

PCL-01-004 Design: Malicious USB key can perform a TOCTOU attack (*High*)

The current implementation of CIRCLean application is suffering from a "*Time of check to time of use"* (TOCTOU) issue[6]. This flaw allows a malicious USB key to trick PyCIRCLean into copying malicious files.

USB storage devices run firmware which responds to the commands sent by the host. Upon scanning the device for malicious files, the modified firmware of a malicious USB key could return "*safe"* content. But once PyCIRCLean copies the files off the USB key, the malicious firmware can instead return *evil* file content. As a result, malicious files get copied to the supposedly safe USB key.

To mitigate the issue the files should be copied from the USB key prior to being sanitized. To achieve this, files need to be either copied to the destination and then deleted and renamed if they are malicious, or, alternatively, they can first be copied to a safe location like */tmp.* Only from that safe location, they could be subjected to further processing.

Note. Due to the limited amount of time allocated to this test, Cure53 has not confirmed this issue practically and relies on public research on the topic[7]. Specifically, the relevant research concerns "*When USB memory sticks lie*"[8] work in this context.

PCL-01-005 Design: Malicious USB key can directly access Safe stick (*High*)

Building on the PCL-01-004, a comparatively more general issue is the fact that all devices are connected to the same *bus*. To a user, this appears as if each USB device had a dedicated slot, but in reality the USB devices are all connected to a single shared *bus*. This means a malicious USB key with a modified firmware could try to spoof USB packets for the safe USB key and write malicious files into it directly.

> *"[...] USB, as the names stands, is a bus interconnect, which means all the USB devices sharing the same USB controller are capable of sniffing and spoofing signals on the bus."*[9]

There is no quick fix for this problem. Unfortunately, the issue cannot be easily mitigated on RaspberryPi due to the complex entanglement of the elements. Notably, even if the process would be changed so that the unsafe USB key is never plugged into the *bus* at the same time that the safe USB key is connected, the critical items are then the SD card and the Ethernet ports. The SD card and the Ethernet ports are connected to the

[6] https://en.wikipedia.org/wiki/Time_of_check_to_time_of_use
[7] https://www.usenix.org/sites/default/files/conference/protected-files/michele_woot12_slides.pdf
[8] http://www.h-online.com/security/news/item/29C3-When-USB-memory-sticks-lie-1775193.html
[9] http://theinvisiblethings.blogspot.de/2011/06/usb-security-challenges.html

system via the same *bus* again, meaning that a malicious USB firmware potentially has access to the network traffic and the SD card.

Note: Due to time constraints it has not been possible to conduct resource-intensive research into this matter. Cure53 relies on the publicly available knowledge in this realm, referring to both the aforementioned quote from the *"USB security challenges"* work and the PCL-01-004 documentation.

PCL-01-007 Bypass: DOCX can trigger DoS through endless parsing *(Medium)*

The PyCIRCLean library uses the *officedissector* library[10] to analyze malicious Microsoft Office files. It was discovered that the former library takes advantage of an XML parser to analyze certain file structures inside the MS Office file. As soon as an XML structure contains an XML External Entity[11], the parser is trying to retrieve the specified remote file. Although it was not possible to steal local files or issue HTTP requests without larger effort, this behavior can be used to cause a Denial of Service. The DoS can be achieved by specifying */dev/random* from the local file system as the source for the entity to resolve to.

The XML parser will endlessly wait for the device to return a character. This cannot occur, so that the process will never finish and remains ongoing.

Steps to reproduce[12]:

- Create a DOCX file which embeds an *OLE object;*
- *Save* the created *Office* file;
- Modify the *word/_rels/document.xml.rels* structure:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<!DOCTYPE test [
<!ENTITY test SYSTEM "file:///dev/random">
]>
[...]
&test;<Relationship Id="rId5"
Type="http://schemas.openxmlformats.org/officeDocument/2006/relationships
/oleObject" Target="embeddings/oleObject1.bin"/>
```

- Place the modified Microsoft Office file on a USB key;
- Let PyCIRCLean analyze the USB key;
- The used *officedissector* library will process the entity, load endlessly. It will cause a Denial of Service as no files are copied to the destination USB key.

[10] https://github.com/grierforensics/officedissector
[11] https://www.owasp.org/index.php/XML_External_Entity_(XXE)_Processing
[12] https://drive.google.com/open?id=0B-LVB5x1knFJUnZEMUZ1TXpfMDA

It is recommended to patch the *officedissector* library and disable the support for XML External Entities in the utilized *lxml* library[13]. This ensures that a malicious Microsoft Office file is no longer able to point to a local file resource and cause a Denial of Service or even a worse problem.

Note: A PoC for this bypass is available at https://cure53.de/exchange/04321789437243/bypasses.zip

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.

PCL-01-006 Firmware: Symlinked files outside evil key are copied *(Info)*

It was found that even though PyCIRCLean marks symlinks as dangerous, it still follows symlinks on the malicious source key. This takes place even when the symlinks in question point to files outside a given key's file system. As a consequence, the file is copied instead of the symlink.

Proof of Concept:

1. Create a symlink to */etc/passwd: ln -s /etc/passwd link;*
2. Execute PyCIRCLean;
3. Inspect the file copied to the destination USB key:

```
$ file src/link1
src/link1: symbolic link to /etc/passwd
$ file dst/DANGEROUS_link_DANGEROUS
dst/DANGEROUS_link_DANGEROUS: ASCII text
$ head -3 dst/DANGEROUS_link1_DANGEROUS
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
```

The described process can lead to the inaccurate files being copied. In other words, files which are not on the source USB key may be copied, potentially leading to a false impression about the data being sanitized or not. It is recommended not to follow symbolic links and depend on the action of copying the link itself. If this is not feasible, it could be otherwise guaranteed that the link does not point outside of its source folder.

[13] http://lxml.de/index.html#documentation

Dr.-Ing. Mario Heiderich, Cure53
Rudolf Reusch Str. 33
D 10367 Berlin
cure53.de · mario@cure53.de

PCL-01-008 Firmware: Specially crafted Filesystem leads to DoS *(Info)*

It was found that a specially modified file system on the source USB key can lead to an infinite error loop when a directory listing is obtained. In effect, listing or copying files would become impossible.

An attacker could prepare an *evil* USB key with two partitions, one having a broken *ext3* file system and another one with a NTFS filesystem[14] containing *evil* files. Obtaining data cannot be accomplished in this context as the application would just freeze at one point. If the *finder* of the evil USB key is not particularly patient and eager to find out what is on the drive, s/he might plug the key into their Windows PC where only the NTFS partition is mounted.

As this is a bug in the driver, the CIRClean team once again has limited options as to how to proceed. One idea might be to use *fsck.ext3* to check the filesystem before mounting it, however this is just another tool is likely to suffer from similar issues.

As this attack relies on social engineering, using the DoS of CIRClean to get a user somewhat frustrated enough use the malicious USB key out of curiosity, a technical solution again does not exist. It is recommended to improve the awareness and policy around using CIRClean. One idea would be to place a file on the destination key before the source key is even touched, and issuing an informational warning. This could read like the following:

> *"The copying process is still running. If you are seeing this file, an error occurred or the process has been interrupted. This might be caused by a broken file system on the source key"*.

After the copying process has ended, the file should be removed.

PCL-01-009 Instrumented Blacklist offers insufficient coverage *(Info)*

PyCIRCLean, among other checks, makes use of a blacklist with file extensions and MIME-types to identify potentially malicious files at a very early stage of processing.

Affected File:
https://github.com/CIRCL/PyCIRCLean/blob/master/bin/filecheck.py#L52

Affected Code (result depends on version):
```
>>> mimetypes.guess_all_extensions("application/octet-stream",strict=False)
['.obj', '.dll', '.so', '.exe', '.bin', '.o', '.a', '.deploy', '.msu', '.msp']
```

[14] https://en.wikipedia.org/wiki/NTFS

While multiple checks are in place and the library does not rely on the blacklist alone, the added list of entries is incomplete and needs to be expanded to incorporate a rather significantly longer list of different file types. Depending on the version of *libmagic* in use, the bypass potential varies in magnitude. Certain test setups used during this assessment allowed for CHM[15] and URL files to go through, for instance.

It is recommended to seek inspiration and benefit from the blacklist employed by Google Chrome[16]. This list has grown over many years and provides considerably better coverage. While blacklists are rarely a tool to be recommended and a whitelist would be harder to circumvent, it is understood that whitelists are not optimal for this project's context. Therefore, it is acceptable for a blacklist to be employed, yet its better maintenance must be ensured.

Strategically, it might also be recommended to offer certain whitelist profiles for certain use-cases and let the *user/administrator* decide as to which issues they should specifically protect against and how.

Conclusions

While the results of this Cure53 penetration test and source code audit of the PyCIRCLean project are generally good, they must be seen through the lens of the pre-existing conditions in which the tools like the CIRCLean operate more broadly. The five Cure53 testers, who completed this assessment over the course of eight days, identified nine security issues and can offer valuable comments on the issues that must be taken into account as the project moves forward.

Among the nine findings, six were classified as actual security vulnerabilities - that is actual design problems and bypasses. The remaining three comprised minor weaknesses. The biggest downside is that the most problematic discoveries were rooted in the software that usually handles the malicious files, notably MS Office and LibreOffice, rather that originate in the tested library. This is because the main products are overly tolerant and lack ways for imposing stricter mechanisms and rules (see e.g. PYC-01-001 and PYC-01-003). The tested library should therefore cease to rely on certain assumption, for example that LibreOffice successfully steers clear from embedding potentially malicious *OLE objects* in the case of them being given completely different name labels.

After the test, certain lessons may have been learnt, yet in many cases the knowledge itself does not equal the possibility to deploy solutions. As is the case with the aforementioned behaviors of LibreOffice, the PyCIRCLean still faces tough choices in

[15] https://en.wikipedia.org/wiki/Microsoft_Compiled_HTML_Help
[16] https://cs.chromium.org/chromium/src/content/browser/download/download_stats.cc?dr=CSs&l=78

terms of how to proceed. This especially holds in the context where it cannot be envisioned that the overly tolerant software changes the problematic behaviors. Instead, the PyCIRCLean maintainers may develop adequate reactions and appropriate mitigations, even if their efforts are partial and complex.

Providing further feedback and observations about the tested tool and library, it has to be underlined that the CIRCLean instruments an interesting concept by moving sanitization to a dedicated affordable device instead of using Antivirus on a Work PC or Laptop. Yet the proposed mechanism cannot replace the original. What is even more evident is the fact that neither can operate successfully without user awareness. In other words, identifying weak links in the process effectively points to the users. They must be educated or encouraged enough to follow reasonable security guidelines to stay safe and actually use the CIRCLean suite consequently. All efforts are in vain if a user decides to plug a malicious USB stick directly because of a DoS, an error on CIRCLean, or just because they do not know better. Another area of interest pertains to blacklists, which are, per a general rule, prone to bypasses executed with obscure file types. Future versions of the PyCIRCLean should consider dependence on whitelists covering certain common scenarios and user-stories. An option to deploy a whitelist would let users limit the attack surface considerably. Similarly, yet another currently observable risk originates from the omission of the low-level attacks on the USB protocols in the threat model. The CIRCLean cannot grant protections against this class of vulnerabilities and if a device is actively attacking the tool, there is close to nothing that can be done on the affected layer available to the tool itself.

In conclusion, CIRCLean is a best-effort approach, not unlike the Antivirus schemes. Consequently, it can never aim at guaranteeing full protection and security on its own. It however might just be what is needed to protect against commonly reported attacks like a USB key dropped in a parking lot with a Trojan executable on it, or a USB stick being being passed around to exchange slides. If the maintainers are ready for the endless game of cat and mouse, their approach might actually become conducive to tackling this and the CIRCLean could then become a successful project. On the final note, it should be emphasized that a dedicated attacker with the knowledge about the tool used by the targeted victim will most certainly be able to find ways around the offered protections. In other words, even the best efforts to have a nearly ideal setup thwarting a majority of the attacks will not eradicate all of the risks that the CIRCLean is up against.

Cure53 would like to thank Raphaël Vinot of CIRCL for his excellent project coordination, support and assistance, both before and during this assignment. We would like to further express our gratitude to the Open Technology Fund in Washington D.C., USA, for generously funding this and other penetration test project, as well as enabling us to publish the results.