

Michael Allar

How to Build a Completely Free Version of the Unreal Marketplace With JavaScript and Automated Crowdsourcing

Version 1.0
December 18, 2015

Building a Completely Free Version of the Unreal Marketplace

Overview

This paper outlines with some engineering and some crowd contributions, one could feasibly build a custom marketplace frontend that would allow downloading of any Unreal Marketplace asset without credentials from Epic's servers.

The Marketplace Asset Delivery Process

Before getting into how this process can be manipulated, this paper will provide a brief overview of how assets are delivered for reference later. The entire process of an end-user getting an asset is as follows:

1. Authorizing with Epic's webservers to retrieve an OAuth token that has permission to download marketplace assets.
2. Pulling down a list of marketplace assets a user can buy or download. In Epic's backend, the 'store' information is regarded as a Catalog Offer and the marketplace asset digital good itself is referred to as a Catalog Item. Pulling down this information can be done in two ways, using the catalog API or by using a simplified API meant for easier parsing by web clients. This requires an OAuth token.
3. Once assets are pulled down and a user wishes to download a purchased asset, the client requests a list of manifest files using the Catalog API regarding the asset. This list of manifest files serves as a list of entry points for the user to download one of many versions of the marketplace asset. This requires an OAuth token.
4. The client then pulls down the appropriate manifest file for the asset they selected. This step and all further steps **no longer require** an OAuth token.
5. The manifest file is then parsed and a list of files and their URLs are made. This involves reversing a simple hash to build the required URL to download an asset 'chunk'.
6. The client downloads all of these asset chunks. An asset chunk is a binary file that contains multiple smaller binary files packed together.
7. When all asset chunks are downloaded, the client then extracts the asset files according to descriptions of each file found in the manifest file. At this point the client has a usable copy of the asset file.

With these steps outlined, this paper will now dissect how someone outside of Epic could piece enough information together to perform this process for themselves using their own credentials. Once we can do this with proper credentials, we will look at how to do this without credentials.

Step 1: Performing OAuth Authorization as an Outsider

Performing successful OAuth authorization requires knowing:

1. The Token and Exchange API endpoints
2. An acceptable Base64 'basic Authorization header' for the Token endpoint
3. A valid username and password set of credentials

To get this information, one simply has to capture web traffic that they are sending out during an Epic Launcher log in. The easiest way to do this is with an HTTPS proxy tool. This paper uses Fiddler4, freely available at <http://www.telerik.com/fiddler>

With Fiddler installed, some HTTPS traffic is visible but some requests are still missing. This is because the Launcher appears to intentionally not respect a user's internet proxy settings, perhaps for this very reason. This could cause issues with users behind proxies however, and a simple Google search for the terms "ue4 launcher proxy" reveals a troubleshooting wiki article for this very problem.

https://wiki.unrealengine.com/Troubleshooting_Launcher_Problems

Building a Completely Free Version of the Unreal Marketplace

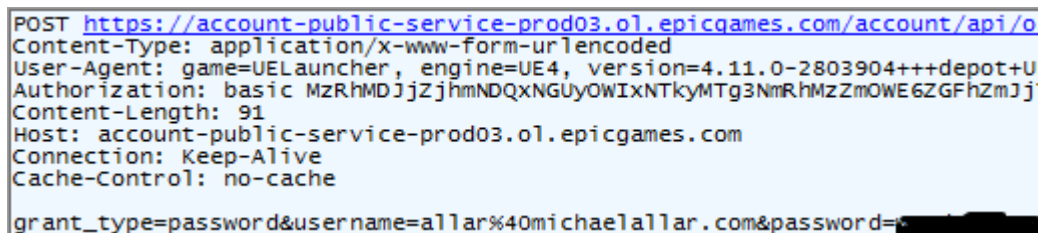
Step 1: Performing OAuth Authorization as an Outsider (Continued)

This wiki article shows that running the launcher with the command argument “-http=wininet” will allow the launcher to respect a user’s proxy settings on Windows, and thus sending traffic through our HTTPS proxy Fiddler for capturing.

Using Wireshark, a common packet sniffing tool, we can find that requests going through our proxy from the launcher is using the TLS1.2 protocol, so it is important to set up your HTTPS proxy / Fiddler to not intercept just SSL3 and TLS1.0 traffic, but TLS1.2 traffic as well.

To get a clean capture from a fresh OAuth process, one can delete the “Saved” folder found in a user’s “AppData\Local\EpicGamesLauncher” folder as this will remove all cached credentials and asset data.

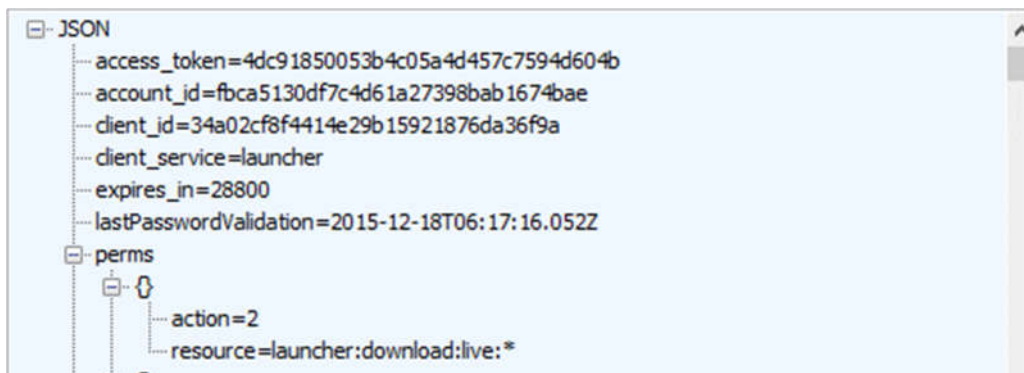
Providing that we already have access to a valid username and password, the OAuth endpoints and the Authorization header required becomes immediately clear.



```
POST https://account-public-service-prod03.ol.epicgames.com/account/api/o
Content-Type: application/x-www-form-urlencoded
User-Agent: game=UE4launcher, engine=UE4, version=4.11.0-2803904+++depot+U
Authorization: basic MzRhMDJjZjhmNDQxNGUyOWIxNTkyMTg3NmRhMzZmOWE6ZjGFhZmJj
Content-Length: 91
Host: account-public-service-prod03.ol.epicgames.com
Connection: Keep-Alive
Cache-Control: no-cache

grant_type=password&username=allan%40michaelallan.com&password=[REDACTED]
```

Figure 1 An example of Fiddler showing an HTTPS request to Epic



```
{
  access_token=4dc91850053b4c05a4d457c7594d604b
  account_id=fbca5130df7c4d61a27398bab1674bae
  client_id=34a02cf8f4414e29b15921876da36f9a
  client_service=launcher
  expires_in=28800
  lastPasswordValidation=2015-12-18T06:17:16.052Z
  perms: {
    action=2
    resource=launcher:download:live:*
  }
}
```

Figure 2 An example of Fiddler showing the JSON response from Epic

Using Fiddler, we find that our endpoints are:

1. <https://account-public-service-prod03.ol.epicgames.com/account/api/oauth/token>
2. <https://account-public-service-prod03.ol.epicgames.com/account/api/oauth/exchange>

And that our token authorization header is (decoded from Base64):

34a02cf8f4414e29b15921876da36f9a:daafbccc737745039dffe53d94fc76cf

Which is clearly the launcher’s client_id along with some secret key of some kind. Knowing how this secret key is made is not required, as passing the encoded Base64 version of this string is enough to pass authorization.

Knowing this information, we can easily run through this process ourselves. Here is a JavaScript snippet from the custom marketplace frontend’s project code completing this process using node.js and the NPM package ‘request’.

Building a Completely Free Version of the Unreal Marketplace

Step 1: Performing OAuth Authorization as an Outsider (Continued)

Getting an OAuth token with valid credentials

```
// Go through Epic's OAuth chain using a username and password
// cb_status is a callback with string parameter of current OAuth status and
// bool of whether complete. (status, bComplete)
epic_api.prototype.OAuthViaPassword = function (user, pass, cb_status) {
    var opts = {
        uri: 'https://account-public-service-
prod03.ol.epicgames.com/account/api/oauth/token',
        headers: { Authorization: 'basic
MzRhMDJjZjhmNDQxNGUyOWIxNTkyMTg3NmRhMzZmOWE6ZGFhZmJjY2M3Mzc3NDUwMzlkZmZlNTNkO
TRmYzc2Y2Y=' },
        form: { grant_type: 'password', username: user, password: pass,
includePerms: true }
    };

    request.post(opts, function(error, response, body) {
        if (response.statusCode == 200) {
            global.epic_oauth = JSON.parse(body);
            if (cb_status != undefined) {
                cb_status('Got OAuth token, exchanging for code', false);
            }
            module.exports.OAuthExchange(cb_status);
        } else {
            if (cb_status != undefined) {
                cb_status('OAuth Via Password failed: ' +
JSON.stringify(response, null, ' '));
            }
        }
    });
}
```

Building a Completely Free Version of the Unreal Marketplace

Exchanging an OAuth token for a valid OAuth bearer token

```
// cb_status is a callback with string parameter of current OAuth status and
// bool of whether complete. (status, bComplete)
epic_api.prototype.OAuthExchange = function(cb_status) {
    var opts = {
        uri: 'https://account-public-service-
prod03.ol.epicgames.com/account/api/oauth/exchange',
        headers: { Authorization: 'bearer ' + global.epic_oauth.access_token
    }
    };

    request.get(opts, function(error, response, body) {
        if (response.statusCode == 200) {
            var json = JSON.parse(body);
            global.epic_oauth.code = json.code;
            if (cb_status != undefined) {
                if (global.epic_SSO === undefined)
                {
                    cb_status('Got OAuth exchange code. Getting SSO.',
false);
                }
                else
                {
                    cb_status('Got OAuth exchange code. Skipping SSO.',
true);
                }
            }
            // Grab our SSO token
            if (global.epic_SSO === undefined) {
                cb_status('Successfully Authorized!', true);
                global.epic_api.GetSSOWithOAuthCode(cb_status);
            }
            // renew our token before it expires
            global.setTimeout(module.exports.OAuthExchange, 250 * 1000);
        } else {
            if (cb_status != undefined) {
                cb_status('OAuth renew failed: ' + JSON.stringify(response,
null, ' '), false);
            }
        }
    });
}
```

Building a Completely Free Version of the Unreal Marketplace

Step 2: Fetching Marketplace Data as an Outsider

In order to download a marketplace asset, we need an asset's Catalog ID and its Asset ID. To create a user-friendly frontend, we will need details such as an assets name, price, and display art. This data often is given together at the same time using one of three methods.

The first method is to use the catalog API which requires an OAuth bearer token retrieved from the authorization process. This API uses two endpoints, one for getting marketplace offers and one for getting information about the digital goods associated with those offers:

1. <https://catalog-public-service-prod06.ol.epicgames.com/catalog/api/shared/namespace/ue/offers?status=SUNSET%7CACTIVE&country=US&locale=en&start=0&count=1000&returnItemDetails=false>
2. <https://catalog-public-service-prod06.ol.epicgames.com/catalog/api/shared/namespace/ue/items?status=SUNSET%7CACTIVE&country=US&locale=en&start=0&count=1000>

With these endpoints, one could explore other namespaces, offers of different status, or getting item details by changing the query strings in the above URIs. With these endpoints and some programming to correlate the data, you can get any available data you'd like from the marketplace ranging from product descriptions, key images, seller information, and most importantly, both Catalog IDs and Asset IDs which will come into play later.

The second way to get comprehensive marketplace data. Doing this will require repeated calls as a single call only retrieves 25 assets at a time, but no OAuth or SSO credentials are required and significantly less logic is needed to use the asset data as it doesn't require correlating of Catalog items and offers.

<https://www.unrealengine.com/assets/ajax-get-categories>

This API endpoint requires a POST with form data such as:

Body	
Name	Value
category	assets/environments
start	0

Once posted, you will get enough asset data to show the asset beautifully on a frontend. This information also contains an asset's Catalog ID and Asset ID without requiring an OAuth token or SSO credentials.

The third way to get asset data and the later needed Catalog ID is to simply scrape all of the marketplace asset pages. If you open any asset page, such as <https://www.unrealengine.com/marketplace/generic-shooter-sample-project>, all of the asset data needed is returned in the resulting HTML. View the page's source and you'll see the asset data stored as a convenient JSON object in the page's HEAD element. This happens to contain all the information for the viewed asset that would be retrieved via the second API above.

No sample code is provided here as pulling down JSON data from web requests and scraping HTML responses is trivial.

Building a Completely Free Version of the Unreal Marketplace

Step 3: Retrieving a List of Marketplace Asset's Manifests as an Outsider

To start the download process of an asset, the client needs to know where the files to download live. This information is stored in manifest files. To get the list of manifest files available for an asset, we need to call an endpoint which requires the OAuth bearer token retrieved in Step 1 and we must own the asset. If we do not own the asset, we will be greeted with a 'missing entitlement' error, which is common for offer/item/entitlement systems. Because of this, there is no way to retrieve the list of manifest files for an asset without ownership of it. This fact will play into crowd contributions later.

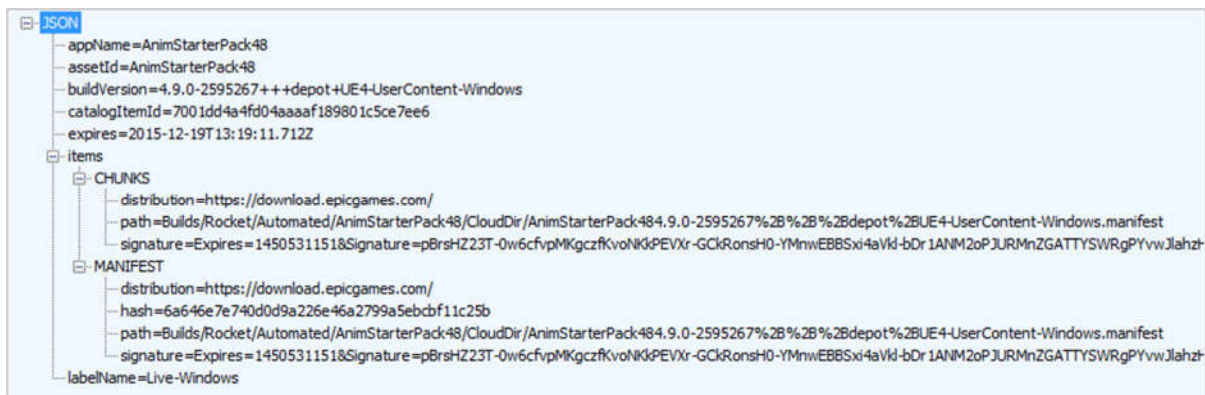
Using the 101 Muzzle Flashes asset as an example, the endpoint for grabbing its manifest list is:

<https://launcher-public-service-prod06.ol.epicgames.com/launcher/api/public/assets/Windows/25601cb5924c4d639a9fbc79dbd0efd6/muzzleflash?label=Live>

Dissecting the last part of this URL...

[25601cb5924c4d639a9fbc79dbd0efd6/muzzleflash?label=Live](https://launcher-public-service-prod06.ol.epicgames.com/launcher/api/public/assets/Windows/25601cb5924c4d639a9fbc79dbd0efd6/muzzleflash?label=Live)

...and comparing this to the asset data we've gotten in Step 2, we can see that this API requires a Catalog ID and an Asset ID. The resulting response from this is:



Depending on the asset there may be multiple manifest files available referring to versions suited for different versions of the engine.

Building a Completely Free Version of the Unreal Marketplace

Step 4: Downloading a Marketplace Asset's Manifest File as an Outsider

Piecing together the distribution and path fields of a MANIFEST property found in Step 3, let us look at the URL for the manifest file for the Animation Starter Pack.

<https://download.epicgames.com/Buils/Rocket/Automated/AnimStarterPack48/CloudDir/AnimStarterPack484.9.0-2595267%2B%2B%2Bdepot%2BUE4-UserContent-Windows.manifest>

One very important thing to note here, if you have been clicking links to API endpoints in this document you should have often been greeted with an unauthorized error of some kind. This manifest URL however requires no authentication to access, and following this link should give you the plaintext JSON manifest. To better look at this JSON file, one can use something like <http://codebeautify.org/jsonviewer> to 'beautify' the JSON and make it easier to read.

Building a Completely Free Version of the Unreal Marketplace

Step 5: Parsing (and Making Sense of) an Asset's Manifest File

Looking ahead to Step 6 where we download binary chunks of an asset, it can be inferred that this manifest file has all the information to know where these chunks live. Grabbing a URL for a chunk and working backwards, we can learn how this manifest file works.

download.epicgames.com	/Builds/Rocket/Automated/AnimStarterPack48/CloudDir/ChunksV3/67/6F2DAEC85B1E1C3E_23F695E782471E021CFB55A1284A5D5C.chunk	930,927
download.epicgames.com	/Builds/Rocket/Automated/AnimStarterPack48/CloudDir/ChunksV3/23/7D39B52EC7642A7F_7139556625407346AFD1B3BF1E18F522.chunk	902,195
download.epicgames.com	/Builds/Rocket/Automated/AnimStarterPack48/CloudDir/ChunksV3/04/2E5ABF47EE5FC0D9_2DD852C32B4725718C61D78C967B8541.chunk	921,281
download.epicgames.com	/Builds/Rocket/Automated/AnimStarterPack48/CloudDir/ChunksV3/44/D7E4ABA0077EF1C8_88E8C811CF4F5F926A05B0B85F6B47B9.chunk	932,293

Looking at the HTTPS traffic for Step 6, we can learn two things: the chunks have static paths and each chunk is roughly 1 Megabyte, similar to Valve's diff and patch solution. Knowing that these paths are static and that the data needed to get these paths is in the manifest file, we have everything we need to figure out how to build these paths for ourselves. Let's take a look at one of these chunk paths:

https://download.epicgames.com/Builds/Rocket/Automated/AnimStarterPack48/CloudDir/ChunksV3/56/1038F1813D4FCBDB_7CB48D15D34AD1AF9766AD8860DA48E6.chunk

Pulling this URL apart, we can see that we need:

1. An App ID (AnimStarterPack48)
2. Some form of 2-character identifier (56)
3. A 16-character hex encoded value of some kind - 8 bytes of data (1038F1813D4FCBDB)
4. A much longer hex encoded value of some kind (7CB48D15D34AD1AF9766AD8860DA48E6)

We already know how to grab the App ID, but we need to figure out where the rest of the URL comes from.

Doing a simple search for the "56" 2-character identifier, we can find this interesting association:

```
"DataGroupList": {  
  "FA8D9E75BA45C4CCB4DFB7A6627D6AD5": "046",  
  "5C34B448874DBE8C7D6D80BC7FE2DADF": "032",  
  "5032B97B6F43344CE4807CAC5CB57253": "094",  
  "DE8E13CE0A4FE572977841A4BB1F50B1": "097",  
  "23F695E782471E021CFB55A1284A5D5C": "067",  
  "2DD852C32B4725718C61D78C967B8541": "004",  
  "88E8C811CF4F5F926A05B0B85F6B47B9": "044",  
  "7139556625407346AFD1B3BF1E18F522": "023",  
  "7CB48D15D34AD1AF9766AD8860DA48E6": "056",
```

Not only do we find our identifier associated to another piece of data, we find that its associated to the 4th piece of data we need: the much longer hex encoded value of some kind.

Searching for this much longer hex string, we find that multiple files are associated with this and that it appears to be a chunk's GUID. We also see that this GUID is associated with this 2-character identifier, a file size value, and a hash value. Taking an optimistic approach and hoping that the 3rd piece of data we need is easily derived from one of these values we can rule out the file size value as it has far too many zeros for it to have enough entropy to easily calculate our 3rd piece of data. The hash value however immediately presents itself with some interesting properties.

Building a Completely Free Version of the Unreal Marketplace

Step 5: Parsing (and Making Sense of) an Asset's Manifest File (Continued)

We know that for this chunk, the 3rd piece of data we need is "1038F1813D4FCBDB". This appears to be a 16-character encoded hex string, or 8 bytes of data.

The hash value for this chunk is "219203079061129241056016", a 24-character string that consists completely of 0-9 characters. Assuming that this is our value, we can also assume that this 24-character string somehow folds into 16-character hex, or 8 bytes of data. Assuming that these 24-characters fold into 8 bytes and that the characters are numeric only, and knowing that any byte of data expressed as numeric text requires 3 ASCII characters or 3 digits (0-255), we can quickly deduce that this hash value is simply 8 bytes represented in decimal form. We can make this hash value more readable by adding a space every 3 characters:

219 203 079 061 129 241 056 016

The fact that none of these values exceed 255 is a definite good sign. Transforming these values into hex gives:

DB CB 4F 3D 81 F1 38 10

Comparing this to our "1038F1813D4FCBDB" we can see that this hash string to URL transform is nothing more than interpreting the hash value as a reversed series of bytes expressed in decimal which is then hex encoded.

Because every chunk URL needs these pieces of data, we can infer that the number of chunks we need is the same number of hash values we have, therefore we can treat the ChunkHashList as the full list of GUIDS of chunks we need to download. We can then correlate all the data needed in the manifest file to build a list of URLs of all the chunks needed for this asset. These chunk URLs (such as the one given above) are static and need no form of authentication to retrieve.

With some simple programming, we can build a list of URLs for these chunks. On the following page is a fully working JavaScript node.js program that is capable of building such a list. For the sake of brevity only a few chunks are listed here, but following all of these links will lead you to a valid chunk file.

https://download.epicgames.com/Buils/Rocket/Automated/AnimStarterPack48/CloudDir/ChunksV3/46/B84338F39E2AF9A7_FA8D9E75BA45C4CCB4DFB7A6627D6AD5.chunk

https://download.epicgames.com/Buils/Rocket/Automated/AnimStarterPack48/CloudDir/ChunksV3/32/F5FCD935CFAA9159_5C34B448874DBE8C7D6D80BC7FE2DADF.chunk

https://download.epicgames.com/Buils/Rocket/Automated/AnimStarterPack48/CloudDir/ChunksV3/94/4328A5D35FBC17CA_5032B97B6F43344CE4807CAC5CB57253.chunk

https://download.epicgames.com/Buils/Rocket/Automated/AnimStarterPack48/CloudDir/ChunksV3/97/68674393189E93FE_DE8E13CE0A4FE572977841A4BB1F50B1.chunk

https://download.epicgames.com/Buils/Rocket/Automated/AnimStarterPack48/CloudDir/ChunksV3/67/6F2DAEC85B1E1C3E_23F695E782471E021CFB55A1284A5D5C.chunk

https://download.epicgames.com/Buils/Rocket/Automated/AnimStarterPack48/CloudDir/ChunksV3/4/2E5ABF47EE5FC0D9_2DD852C32B4725718C61D78C967B8541.chunk

https://download.epicgames.com/Buils/Rocket/Automated/AnimStarterPack48/CloudDir/ChunksV3/44/D7E4ABA0077EF1C8_88E8C811CF4F5F926A05B0B85F6B47B9.chunk

Building a Completely Free Version of the Unreal Marketplace

Step 5: Parsing (and Making Sense of) an Asset's Manifest File (Continued)

```
var http = require('http');
var fs = require('fs');
var rimraf = require('rimraf');
var mkdirp = require('mkdirp');
var Download = require('download');
var zlib = require('zlib');

// Load manifest file into JSON object
var manifestJSON = JSON.parse(fs.readFileSync('AnimStarterPack48.manifest',
'utf8'));

var HexChars = ["0", "1", "2", "3", "4", "5", "6", "7", "8", "9", "A", "B",
"C", "D", "E", "F"];
function ByteToHex(b) {
    return HexChars[(b >> 4) & 0x0f] + HexChars[b & 0x0f];
}

// Converts Epic's chunk hash to chunk URL hex data
function ChunkHashToReverseHexEncoding(chunk_hash) {
    var out_hex = '';
    for (var i = 0; i < chunk_hash.length / 3; ++i) {
        out_hex = ByteToHex(parseInt(chunk_hash.substring(i*3, i*3+3))) +
out_hex;
    }
    return out_hex;
}

// Pads a string with leading zeros or passed in string, i.e. padLeft(4,2) =
"04"
// http://stackoverflow.com/questions/5366849/convert-1-to-0001-in-javascript
function padLeft(nr, n, str){
    return Array(n-String(nr).length+1).join(str||'0')+nr;
}

// Build chunk URL list
var chunks = [];
var chunkBaseURL = 'http://download.epicgames.com/Builds/Rocket/Automated/' +
manifestJSON.AppNameString + '/CloudDir/ChunksV3/';
for ( var chunk in manifestJSON.ChunkHashList )
{
    var hash =
ChunkHashToReverseHexEncoding(manifestJSON.ChunkHashList[chunk]);
    var group = padLeft(parseInt(manifestJSON.DataGroupList[chunk]), 2);

    chunkBaseURL + group + '/' + hash + '_' + chunk + '.chunk';
    chunks.push({guid: chunk, hash: hash, url: chunkBaseURL + group + '/' +
hash + '_' + chunk + '.chunk'});
}

console.log(chunks);
```

Building a Completely Free Version of the Unreal Marketplace

Step 6: Downloading an Asset's Chunks as an Outsider

This is a rather trivial step. Using the URLs generated from the manifest, one can iterate through then and save the files locally. Extending onto the JavaScript program in Step 5, here is an example of downloading all these chunks.

```
// Download chunks
var downloadDir = './downloads/' + manifestJSON.AppNameString + '/chunks/';
// Build download dir
rimraf.sync(downloadDir + '.*'); // Deletes download dir and all old chunks if they exist
mkdirp.sync(downloadDir); // Creates download path recursively if it doesn't exist

var chunkDownloader = new Download();
chunkDownloader.dest(downloadDir);
for (var i = 0; i < chunks.length; ++i) { chunkDownloader.get(chunks[i].url);
}
console.log(chunks.length);
chunkDownloader.run(function(err, files) {
    console.log("Done downloading chunks.");
    decompressChunks();
    extractAssetsFromChunks();
});
```

Step 7: Extracting Asset Files from Chunks as an Outsider

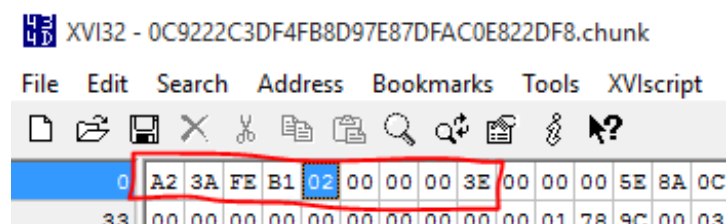
Extracting assets from chunks at first is a rather daunting task, but taking it step by step simplifies it greatly.

Knowing that each asset file in the manifest points to data in the chunk files and has offset and size information, although encoded the same way as the hash string in Step 5, the first step is building an automated way to pull this data out of the chunk files. This requires only simple I/O programming to read parts of a file and write those parts to another file. This might prove easy at first but if one tries to use the resulting .uasset files it is clear that the resulting files are not proper .uasset files at all, which means the chunk files themselves might have some special encoding or serialization happening.

Having done some minor reverse engineering in the past, the following was my thought process of how I was able to crack the chunk file format and successfully extract marketplace asset files. Other outsiders might approach the issue differently but they will generally lead to the same results.

The general first step in reverse engineering a file format such as this chunk format is to look for patterns across files that could help identify data. It is incredibly common among file formats to have some form of standardized header that gives meta data about the stored information, such as how big the file is, how big the header is (thus allowing to know when the 'real data' starts', a format version, and so on.

Upon comparing a good number of the chunk files from the Animation Starter Pack with each other using a hex editor, it was clear that a header pattern was being used. One constant piece of data existed in all chunk files:



Building a Completely Free Version of the Unreal Marketplace

Step 7: Extracting Asset Files from Chunks as an Outsider (Continued)

Looking at the first 9 bytes of these chunk files, one can start taking guesses as to what this data represents. As mentioned before, static header information that would be present across all files that are common in headers of many formats are things such as a 'magic number', a header version, and a header size. With the common knowledge that data types are usually stored as 1, 2, or multiples of 4 bytes, it can be inferred that these 9 bytes are multiple pieces of data. The last bytes being "00 00 00 XX" is an obvious sign that this is a 4-byte int quite a small value. This leaves 5 bytes to the left. The "02" doesn't 'feel' like it belongs with the 4 bytes before it and it would make sense for this byte to represent something on its own, creating a 4-1-4 byte data arrangement.

With the first four bytes being unlike the rest and having the greatest data entropy, I optimistically made the assumption that this is a 'magic number' of some kind that is being used to identify a file as this chunk format. Having heard of the term 'chunk' during the cooking, patching, and the dlc release process of some more deep level Unreal Engine cooker features, I figured that Epic would not create two formats using the term 'chunk'. Because of this, and because of the fact that anyone with the engine can invoke these cooker processes, Epic's code to handle chunk processing must live somewhere in the Engine source code that is exposed to the public. Pressing forward with this, I decided to run a "Find in Files" search for these four bytes: "A23AFEB1". This turned up nothing. Inspired by how the hashing for files was done in the manifest, I decided to reverse these bytes as if they were written with a different endianness. Searching for "B1FE3AA2" led me directly to "CHUNK_MAGIC_HEADER" in Engine\Source\Runtime\Online\BuildPatchServices\Private\BuildPatchChunk.cpp.

After an hour of reading through BuildPatchChunk.cpp, there is enough learnable information to do everything you need to reverse this 'chunking' process. The way the chunk header was read in and what the data meant became immediately obvious and after further digging it was clear that chunk data is being compressed using zlib and that is why the initial naïve attempt at extracting asset files was incredibly garbled. With all this information in mind and using BuildPatchChunk.cpp as a guide, it was possible to create a chunk extractor using JavaScript.

The code on the next page is more code to serve as an addition to the code in Step 5 and Step 6. Node.js has many useful features built in to handle extracting chunk asset data, such as the 'zlib' module which allows uncompressing of the chunk data with a single line of code. The following code iterates through all download chunks, checks to see if they are compressed and decompresses if needed, then overwrites the chunk files with 'just the raw data' without headers or compression. Pulling asset files from this resulting chunk file using the offsets and sizes found in the manifest file then become trivial. This also has the nice bonus of allowing Linux users to download marketplace assets, as the code provided will run on any platform.

Building a Completely Free Version of the Unreal Marketplace

Step 7: Extracting Asset Files from Chunks as an Outsider (Continued)

```
var extractDir = './downloads/' + manifestJSON.AppNameString + '/extracted/';
rimraf.sync(extractDir); // Deletes extract dir and all old files if they
exist
mkdirp.sync(extractDir); // Creates extract path recursively if it doesn't
exist

// Decompresses chunks compressed with zlib
function decompressChunks() {
    // Strip chunk hashes from file names
    var chunkFiles = fs.readdirSync(downloadDir);
    for (var i = 0; i < chunkFiles.length; ++i) {
        fs.renameSync(downloadDir + chunkFiles[i], downloadDir +
chunkFiles[i].substring(17));
    }

    // Decompress chunk files
    chunkFiles = fs.readdirSync(downloadDir);
    for (var i = 0; i < chunkFiles.length; ++i) {
        // Open the file for reading
        var file = fs.openSync(downloadDir + chunkFiles[i], 'r');

        // We need to first read a chunk's header to find out where data
        begins and if its compressed
        // Header details can be found in
        Engine\Source\Runtime\Online\BuildPatchServices\Private\BuildPatchChunk.cpp
        // Header size is stored in the 9th byte (index 8)
        // Whether a file is compressed is always at header byte 41 (index 0)
        var headerBuffer = new Buffer(41);
        fs.readSync(file, headerBuffer, 0, 41, 0);

        var headerSize = headerBuffer[8];
        var compressed = (headerBuffer[40] == 1);

        var stats = fs.statSync(downloadDir + chunkFiles[i]);
        var chunkBuffer = new Buffer(stats['size'] - headerSize);
        fs.readSync(file, chunkBuffer, 0, stats['size']-headerSize,
headerSize);
        fs.closeSync(file);

        if (compressed) {
            fs.writeFileSync(downloadDir + chunkFiles[i],
zlib.unzipSync(chunkBuffer));
        } else {
            fs.writeFileSync(downloadDir + chunkFiles[i], chunkBuffer);
        }
    }
}

function extractAssetsFromChunks() {
    // Iterate through all asset files split into chunks
    for (var i = 0; i < manifestJSON.FileManifestList.length; ++i)
    {
        var fileSize = 0;
        var fileName = extractDir +
manifestJSON.FileManifestList[i].Filename;
        var fileDir = fileName.substring(0, fileName.lastIndexOf('/'));
    }
}
```

Building a Completely Free Version of the Unreal Marketplace

```
    mkdirp.sync(fileDir); // Creates asset file path if it doesn't exist

    // Iterate through all chunks and get total file size
    for (var j = 0; j <
manifestJSON.FileManifestList[i].FileChunkParts.length; ++j) {
        fileSize +=
parseInt('0x'+ChunkHashToReverseHexEncoding(manifestJSON.FileManifestList[i].
FileChunkParts[j].Size));
    }

    var buffer = new Buffer(fileSize);
    var bufferOffset = 0;

    // Read data from chunk files and store them in the buffer
    for (var j = 0; j <
manifestJSON.FileManifestList[i].FileChunkParts.length; ++j) {
        var chunkGuid =
manifestJSON.FileManifestList[i].FileChunkParts[j].Guid;
        var chunkOffset =
parseInt('0x'+ChunkHashToReverseHexEncoding(manifestJSON.FileManifestList[i].
FileChunkParts[j].Offset));
        var chunkSize =
parseInt('0x'+ChunkHashToReverseHexEncoding(manifestJSON.FileManifestList[i].
FileChunkParts[j].Size));

        var file = fs.openSync(downloadDir + chunkGuid + '.chunk', 'r');
        fs.readSync(file, buffer, bufferOffset, chunkSize, chunkOffset);
        fs.closeSync(file);

        bufferOffset += chunkSize;
    }

    fs.writeFileSync(fileName, buffer);
}

rimraf.sync(downloadDir); // Deletes chunk dir
}
```

Building a Completely Free Version of the Unreal Marketplace

Moving from Authorized Downloads to Unauthorized Downloads

Now that being able to complete the entire Marketplace asset delivery process as an authorized outsider is proved to not just completely possible but already executable, approaching downloads of marketplace assets as an unauthorized outsider can now be done.

With the sample code provided an outsider can easily download and use a marketplace asset given a manifest file. Mentioned in the initial step outline, currently anyone can grab a manifest file for a marketplace asset without any credentials given that they know the URL to the manifest file. Let's take another look at the Animation Starter Pack manifest URL. All steps that lead up to building this URL need valid credentials, but all steps afterwards do not.

<https://download.epicgames.com/Builds/Rocket/Automated/AnimStarterPack48/CloudDir/AnimStarterPack484.9.0-2595267%2B%2B%2Bdepot%2BUE4-UserContent-Windows.manifest>

Given this URL, we can see we need the following pieces of data:

1. An asset's app id (AnimStarterPack48)
2. A second reference to an app id of some kind (found to be a variant name for assets with multiple releases but in this case it is AnimStarterPack48)
3. A Perforce revision number the asset was built against

As shown in step 3, an asset's app id is freely available through many endpoints.

Fortunately getting the variant id and perforce revision a release of an app id was built against is much more difficult and the only way to do so that I have found **absolutely 100%** requires user authentication, however this data is **static** and so if one authorized user disseminates this information than **anyone** can proceed to build a proper URL to an asset's manifest file.

This means that if multiple people split the costs to buy assets to get this info or create a custom frontend that would auto-collect this data from its users, **everyone** would have access to this asset directly from Epic's CDN for free. This would render DMCA notices obsolete. Given that a custom frontend for the marketplace is also not just possible but already created and being used by people ([https://forums.unrealengine.com/showthread.php?93657-Custom-UE4-Marketplace-Frontend-\(Full-Source-Available\)&highlight=custom+frontend](https://forums.unrealengine.com/showthread.php?93657-Custom-UE4-Marketplace-Frontend-(Full-Source-Available)&highlight=custom+frontend)), adding a 'Download For Free' button if this information was collected would be trivial.

Because we can complete the entire marketplace delivery process for an authorized user, all it takes to collect the information to allow free downloading of an asset would be for a user to log in with a custom frontend as it is easy to crawl through all the assets a user owns (Step 2 includes a "isOwned" property) and kick off requests for manifest lists. If those manifest lists were collected, knowingly or unknowingly, then the only way to prevent piracy would be to change the backend significantly to impede this process. If this was automated, clients could send the full manifest files themselves and skip the manifest URL building process completely.

Even without a script, users who wanted to contribute to marketplace piracy but without any programming skill can also get the needed information. Asset app ids, variant ids, and perforce revision numbers are visible as plain text in the following locations:

1. Launcher's VaultCache\asset_id\stage\Install\\$resumedata file. (Start and stop a launcher download before completion and it will generate one of these)
2. Launcher's logs. You don't have to look very hard, when you log in it does a dump of EVERY ASSET ID AND PERFORCE REVISION YOU OWN

If you have any questions or would like any more details on anything discussed in this paper, please contact Michael Allar at allar@michaelallar.com

Building a Completely Free Version of the Unreal Marketplace

Possible Ways of Prevention

There are a few things that could be done that would have made the discovery and exploitation of this information impractical enough for anyone to attempt. I have only limited training in web security but I find that these fixes would be the most worthwhile from stopping a piracy attack on the marketplace.

Low-cost fixes with moderate impact:

1. Simply removing references to the needed information of gathering manifest data from the Launcher logs and not having this data stored as plaintext locally would go a long way as this would force people who would want to contribute to piracy to use a scripted client of some kind.
2. Run an obfuscator on asset manifest and manifest list JSONs of some kind, even if it's just running it through a Base64 encoder. Reversing this in the marketplace launcher should be trivial.

High-cost fixes with significant impact:

3. Manifest files should not be plain JSON. Manifest files should always be binary files that are not human readable. If the manifest files were not human readable I would have stopped then and there. They are currently too transparent to not put behind an OAuth security wall.
4. Salt the chunk file hashes with a secret only stored in the Launcher, as the Launcher is the only 'official' way of downloading assets currently. Finding this secret would require deep levels of reverse engineering.

Follow Up and Additional Details

If you have any questions, would like any more details on anything discussed, or raw copies of working code that demonstrates what is shown in this paper, please contact Michael Allar at allar@michaelallar.com