**Multiplayer Assembly Package  
Tutorials  
Server Controlled Events: EXP Modifiers**

This tutorial will teach you how to control certain aspects of your container systems by means of a single line server message.

**What is it?**When you play the “latest and greatest” Call of Duty game, you are almost always likely to come across a double experience weekend. Players are thrilled by events in which aspects are suddenly switched on/off by a source to make their difficult tasks easier. This tutorial will go in depth on how to run things like double exp events using the MAP, while keeping the system safe and secure from potential hacks.

**Pre-Tutorial Work**Back to the fun Pre-Tutorial stuff. Remember that good ol’ RSA Certificate Generation page you had to visit to install the pack? Well, you will want to head back there now as we will need another RSA certificate.

I will be working with the Client-Side RP system here, so if you are using the server-side example alone your results may be different.

**C++ Work**The first thing we need to do is add the new certificate. Open the header file for your EXP system and add these lines to the “private” section:

F32 EXPMulti;

const char \*EMKey();  
 bool certVerify(std::string message, std::string signature);

Then in the “public” section add these definitions:

F32 getEXPMultiplier();

void setEXPMultiplier(F32 amount, const char \*rsaSig);

Now, open up the .cpp file, and paste this somewhere near the top:

const char \*expControl::EMKey() {

return "Replace Me";

}

Obviously replace the “Replace Me” with the public key hex of the new RSA key.

Scroll down to the constructor of the class and right underneath where it defines plEXP = 0; add this line:

EXPMulti = 1.0f;

Next we need to add our certVerify function which will serve as the test for the messages sent by the server.

bool expControl::certVerify(std::string message, std::string signature) {

std::string dec, fin;

fin.assign(EMKey());

fin.append("h");

Integer rsaPub(fin.c\_str()), rsaexp("65537");

std::string holder;

try {

RSASSA\_PKCS1v15\_SHA\_Verifier verifier;

verifier.AccessKey().Initialize(rsaPub, rsaexp);

cryptoPackage->HexDecode(signature, holder);

StringSource(message+holder, true,

new SignatureVerificationFilter(

verifier, NULL,

SignatureVerificationFilter::THROW\_EXCEPTION

) // SignatureVerificationFilter

); // StringSource

}

catch(CryptoPP::Exception e) {

return false;

}

return true;

}

Now we will add the methods for setting and getting the EXP modifier:

F32 expControl::getEXPMultiplier() {

return EXPMulti;

}

void expControl::setEXPMultiplier(F32 amount, const char \*rsaSig) {

//verify the signature

char number[32];

dSprintf(number, sizeof(number), "%.1f", amount);

std::string weWant = "MAPEXP";

weWant.append(number);

std::string valid = cryptoPackage->pgdHash(weWant);

if(certVerify(valid, string(rsaSig))) {

Con::printf("\* EXP Multiplier Set to: %f", amount);

EXPMulti = amount;

}

else {

Con::errorf("\*\*\* EXP Multiplier Set Failed, Certificate Verification Denied.");

}

}

Next we will modify the gain experience function to detect the EXP modifier and use it properly. Scroll down to addEXP() and where you see plEXP += amount; change it to read: plEXP += amount \* EXPMulti;

Lastly, we will add our new Torque Script functions for setting and getting the EXP Modifier:

DefineEngineFunction(setEXPModifier, void, (F32 newMod, const char \*sig),, "(int, string)") {

experience->setEXPMultiplier(newMod, sig);

}

DefineEngineFunction(getEXPModifier, F32, (),, "returns the current EXP Modifier") {

return experience->getEXPMultiplier();

}

**PHP Work**Now, create a new PHP file on your web-server. Place it somewhere that you will remember.

<?php

//RSA.php is included with the package…

include\_once("path/to/RSA.php");

//functions

function XXZHash($string) {

$interior = sha1($string);

$exterior = hash('whirlpool', $interior);

return $exterior;

}

function EM\_private() {

$key = "You know what to put here xD”;

return $key;

}

//Not really used here, but keep it in case you need to debug.

function EM\_public() {

$key = "Same Here”;

return $key;

}

function obtainEXPMod() {

$modifier = 1.0; //yay EXP mod!

$hash = XXZHash("MAPEXP".$modifier."");

$rsa = new Crypt\_RSA();

$rsa->loadKey(EM\_private());

$rsa->setSignatureMode(CRYPT\_RSA\_SIGNATURE\_PKCS1);

$signature = $rsa->sign($hash);

$sigFinal = bin2hex($signature);

die("$"."MAP ".$modifier." ".$sigFinal);

}

obtainEXPMod();

?>

Change the fields as necessary. The clever coder will try to change my above code to use a MySQL database to handle the EXP modifier, but I leave that to you. =)

**TS Work**The last thing we need to do is to have our game client communicate with the server to obtain the message regarding the current EXP rate. Place this somewhere on the client side, and then every time the client logs in, call downloadEXPModifier(); and once you have that set, you’re done!

//Weapon EXP Modifier Code

//Connect to PGD to download the EXP Mod Rate

//Script Kiddies Be Warned, this system is secured via RSA-4092 Signature Verification... so don't even bother..

function downloadEXPModifier() {

if(isObject(ERDownload)) {

ERDownload.delete();

}

%http = new HTTPObject(ERDownload);

%http.get("www.mywebsite.com:80", “path/to/your/php/file.php");

}

function ERDownload::onConnected(%this) {

//do nothing.

}

function ERDownload::onLine(%this, %line) {

if(strstr(%line, "$MAP") != -1) {

%line = strReplace(%line, "$MAP", "");

%response = stripchars(%line, "$\n");

//test the rate via RSA-signature verification

%rate = getWord(%response, 1);

%sig = getWord(%response, 2);

setEXPModifier(%rate, %sig);

}

if(getEXPModifier() > 1) {

%newRate = getEXPModifier();

//make a GUI to show EXP event is on?

}

}

function ERDownload::onDisconnect(%this) {

%this.delete();

}

**Post Tutorial Notes**

Simple Right? Good, sometimes the best approach to methods is a simple one. This method is safe because only the server holds the private key to generate correct messages.

Sending secure server messages can be tedious, but following this approach to doing it will save you the hastle of needing to deal with pesky hackers later on!