Abbildung 1: The World of Warcraft client downloading a patch.

Tutorial

# Implementing in-client patching for World of Warcraft

*„stoneharry“*
Bernd *„schlumpf“* Lörwald

29. April 2012

## 1 What is this tutorial about?

This tutorial allows you to use the built-in World of Warcraft updating functionality, so that you can send custom patches to the client based on their client version. This is shown in figure 1.

## 2 How to construct a patch

The patching process allows you to simply send a MoPaQ file to the client, which can include arbitrary files and a list of commands being executed as soon as the MoPaQ is downloaded called *prepatch.lst*

Typically, such a MoPaQ file includes a binary *installer.exe* to be executed as well as a *prepatch.lst* file saying

```
delete some_no_longer_needed_file
extract some_new_file
extract installer.exe
execute installer.exe
```

This extracts the updater, and then runs it. The updater handles the updating process and then deleting the *wow-patch.mpq*. *wow-patch.mpq* is what the client calls the MoPaQ file downloaded from the server, and is checked for and ran if found upon logging in.

The *prepatch.lst* can include the commands

- execute, which executes an arbitrary file given.
- extract, which extracts a file from the MoPaQ.
- delete, which deletes the named file.

The file needs to be saved with windows-style line endings (\r\n). Each line can be at most 260 characters long.

## 3 Sending the client the patch.

When logging into the server, the server receives the client's build number. Depending on the client's build, it is able to do different things.

You need to make it so that if the client build is lower or equal to 12 340 (patch 3.3.5a), the server will check for updates, and send them to the client.

The patching process works by the patch being selected, and then the server telling the client that it is about to send a patch and how big the patch is. The client accepts this, and tells the server where to start sending from – the full patch if not started being sent before. This means that if the client is disconnected during the transfer for whatever reason, it can resume from where it left off. The server will keep sending 1 500 byte chunks of the patch until the client has the full patch. The client will thus have a *wow-patch.mpq* in the World of Warcraft directory now, and when the restart button is pressed it will try to open it and execute the contained *prepatch.lst*.

The source file modifications required to get this process to work on server side are listed below:

## 3.1 ArcEmu

All of the edits described below take place in the ArcEmu-Logonserver project.

In *main.cpp* we have this code snippet:

```
 1 while(mrunning.GetVal())
 2 {
 3   if(!(++loop_counter % 20))                // 20 seconds
 4     CheckForDeadSockets();
 5
 6   if(!(loop_counter % 300))                 // 5 mins
 7     ThreadPool.IntegrityCheck();
 8
 9   if(!(loop_counter % 5))
10   {
11     sInfoCore.TimeoutSockets();
12     sSocketGarbageCollector.Update();
13     CheckForDeadSockets();                  // Flood Protection
14     UNIXTIME = time(NULL);
15     g_localTime = *localtime(&UNIXTIME);
16   }
17
18   PatchMgr::getSingleton().UpdateJobs();
19   Arcemu::Sleep(1000);
20 }
```

Each second this code snippet is called. We are interested in line 16. A job is created when a patch is sending. Using this code, it would send 1 500 bytes (1 chunk) each second. This would take a very, very long time to send 100 MB. By reducing the sleep time you can make it so that it sends at a much faster rate. You would have to increase the wait time on the other checks by adding to the % value checks.

A much more efficient way to handle it would be to add the UpdateJobs check to a new thread. However, in reality the logonserver uses very, very little CPU and most machines can run this without any issues, which is bad logic but is a quick implementation:

```
const int cycles_per_second (1000);

while(mrunning.GetVal())
{
  if(!(++loop_counter % (20 * cycles_per_second)))  // 20 seconds
    CheckForDeadSockets();

  if(!(loop_counter % (300 * cycles_per_second)))   // 5 mins
```

```
    ThreadPool.IntegrityCheck();

  if(!(loop_counter % (5 * cycles_per_second)))
  {
    sInfoCore.TimeoutSockets();
    sSocketGarbageCollector.Update();
    CheckForDeadSockets();                                // Flood Protection
    UNIXTIME = time(NULL);
    g_localTime = *localtime(&UNIXTIME);
  }

  PatchMgr::getSingleton().UpdateJobs();
  Arcemu::Sleep(1000 / cycles_per_second);
}
```

The following code snippet from *AuthSocket.cpp* shows how the patch is selected for the client:

```
if(build < LogonServer::getSingleton().min_build)
{
  char flippedloc[5] = {0,0,0,0,0};
  flippedloc[0] = m_challenge.country[3];
  flippedloc[1] = m_challenge.country[2];
  flippedloc[2] = m_challenge.country[1];
  flippedloc[3] = m_challenge.country[0];

  m_patch = PatchMgr::getSingleton().FindPatchForClient(build, flippedloc);
  if(m_patch == NULL)
  {
    LOG_DETAIL("[AuthChallenge] Client %s has wrong version. Server: %u, Client:
        %u", GetRemoteIP().c_str(), LogonServer::getSingleton().min_build,
        m_challenge.build);
    SendChallengeError(CE_WRONG_BUILD_NUMBER);
    return;
  }
}
```

The logic is that if the client version is less than the server version, then to find the patch for the client. If the patch is found, then to send it to them, else to return a wrong build error.

In ArcEmu the patches are formatted with the pattern *LocaleBuild.mpq*. These go in a folder called *ClientPatches/* in the same folder as your world executable. For example, for a enGB client running the build 12 340 (patch 3.3.5a) that you would like to update, you would name the patch *enGB12340.mpq*.

The next change happens in *AutoPatcher.cpp*: Replace the whole method named `Patch * PatchMgr::FindPatchForClient(uint32 Version, const char * Locality)` with the following listing. You will also need to correct the header file to match the new signatures. Also, you need to add a call to `InitializePatchList ()` to `PatchMgr::PatchMgr()`.

Listing 1: Corrected version of Patch* PatchMgr::FindPatchForClient()

```
void PatchMgr::InitializePatchList()
{
  Log.Notice ("PatchMgr", "Loading Patches...");

  const size_t path_length (MAX_PATH * 10);

  char base_path[path_length];
  char absolute_filename[path_length];

#ifdef WIN32
  char file_pattern[path_length];

  if (!GetCurrentDirectory (sizeof (file_pattern), file_pattern))
    return;

  strcpy (base_path, file_pattern);
  strcat (file_pattern, "\\ClientPatches\\*.*");

  WIN32_FIND_DATA fd;
  HANDLE fHandle (FindFirstFile (file_pattern, &fd));
  if (fHandle == INVALID_HANDLE_VALUE)
    return;
#else
  strcpy (base_path, ".");

  struct dirent ** list (NULL);
  int filecount (scandir ("./ClientPatches", &list, 0, 0));
  if (filecount <= 0 || list== NULL)
  {
    Log.Error("PatchMgr", "No patches found.");
    return;
  }
#endif

#ifdef WIN32
  do
#else
  while (filecount--)
#endif
  {
#ifdef WIN32
    snprintf (absolute_filename, sizeof (absolute_filename),
        "%s\\ClientPatches\\%s", base_path, fd.cFileName);
#else
    snprintf (absolute_filename, sizeof (absolute_filename),
        "%s/ClientPatches/%s", base_path, list[filecount]->d_name);
#endif

    uint32 srcversion;
    char locale[5] = "****";
    if (sscanf(fd.cFileName,"%4s%u.", locale, &srcversion) != 2)
    {
      Log.Notice ("PatchMgr", "Found incorrect patch file: %4s %s", locale,
          fd.cFileName);
      continue;
    }

#ifdef WIN32
    HANDLE hFile (CreateFile (absolute_filename, GENERIC_READ, 0, NULL,
        OPEN_EXISTING, FILE_ATTRIBUTE_ARCHIVE, NULL));
    if(hFile == INVALID_HANDLE_VALUE)
      continue;
#else
    const int file_descriptor (open (absolute_filename, O_RDONLY));

    if (file_descriptor <= 0)
    {
      LOG_ERROR("Cannot open %s", absolute_filename);
      continue;
    }

    struct stat stat_buffer;
    if(fstat(file_descriptor, &stat_buffer) < 0)
    {
      LOG_ERROR("Cannot stat %s", absolute_filename);
      continue;
    }
#endif

    Log.Notice ("PatchMgr", "Found patch for %u locale `%s`.", srcversion,
        locale);

    Patch* patch (new Patch);

#ifdef WIN32
    DWORD sizehigh;
    DWORD size (GetFileSize (hFile, &sizehigh));
#else
    unsigned int size (stat_buffer.st_size);
#endif
    patch->FileSize = size;
    patch->Data = new uint8[size];
    patch->Version = srcversion;
    for(size_t i (0); i < 4; ++i)
      patch->Locality[i] = static_cast<char> (tolower (patch->Locality[i]));
    patch->Locality[4] = '\0';
    patch->uLocality = *(uint32*) (patch->Locality);

#ifdef WIN32
    const bool result (ReadFile (hFile, patch->Data, patch->FileSize, &size,
        NULL));
#else
    size = read (file_descriptor, pPatch->Data, size);
    const bool result (size > 0);
#endif
    ASSERT (result);
    ASSERT (size == patch->FileSize);

```

```
#ifdef WIN32
    CloseHandle (hFile);
#else
    close (file_descriptor);
#endif

    MD5Hash md5;
    md5.Initialize();
    md5.UpdateData (patch->Data, patch->FileSize);
    md5.Finalize();
    memcpy (patch->MD5, md5.GetDigest(), MD5_DIGEST_LENGTH);

    m_patches.push_back (patch);

#ifndef WIN32
    free (list[filecount]);
#endif
  }
#ifdef WIN32
  while (FindNextFile (fHandle, &fd));

  FindClose(fHandle);
#else
  free (list);
#endif
}

const Patch* PatchMgr::FindPatchForClient(uint32 Version, const char * locale) const
{
  const char lower_case[4] = {tolower (locale[0]), tolower (locale[1]), tolower (locale[2]), tolower (locale[3])};
  const uint32 ulocale (*(uint32*)lower_case);

  const Patch * fallbackPatch (NULL);
  for( std::vector<Patch*>::const_iterator patch_it (m_patches.begin())
     ; patch_it != m_patches.end()
     ; ++patch_it
     )
  {
    const Patch* patch (*patch_it);
    if(patch->uLocality == ulocale)
    {
      if(patch->Version == Version)
        return patch;

      if(fallbackPatch == NULL && patch->Version == 0)
        fallbackPatch = patch;
    }
  }

  return fallbackPatch;
}
```

This changes it so that upon this function being called, it gets the correct patch and returns it.

Next go to the function `bool PatchMgr::InitiatePatch(Patch * pPatch, AuthSocket * pClient)` and remove the last assignment in the line `init.name[0] = 'P'; init.name[1] = 'a'; init.name[2] = 't'; init.name[3] = 'c'; init.name[4] = 'h'; init.name[5] = '\0';`, so that it becomes

```
init.name[0] = 'P';
init.name[1] = 'a';
init.name[2] = 't';
init.name[3] = 'c';
init.name[4] = 'h';
```

Inside *AutoPatcher.cpp*, you also find this definition, which has the size of `char name[];` off by one. Correct the size to be 5 instead of 6.

```
struct TransferInitiatePacket
{
  uint8 cmd;
  uint8 strsize;
  char name[6];
  uint64 filesize;
  uint8 md5hash[MD5_DIGEST_LENGTH];
};
```

Some of the code above might not work on other platforms than Windows. You may want to adjust it where needed.

### 3.2 TrinityCore

A patch for TrinityCore is provided by schlumpf here. It was based on an older version, so you might need to adjust parts of it.

## 4 Patching the client to verify the patch

For the client to verify and attempt to install a patch, you would have to sign your patch with Blizzard's private key, which is sadly non-public. Therefore you need to disable the client verifying that the patch is signed by Blizzard. The following code is for the OSX version of World of Warcraft Mists of Pandaria, so your experience on Windows will differ.

As soon as you click the `restart` button after downloading the patch, the code seen in listing 2 gets executed. As you can see, the part responsible for failing is in line 17 to 21. If `SFileAuthenticateArchiveEx()` returns a value of `authresult` less or equal to 4, patching will be aborted. Therefore, one needs to either change the `if` to always be true and therefore be executing the patch, or `SFileAuthenticateArchiveEx()` to always verify the archive. You can do the latter either by changing modulus and exponent to your own ones – which would be good, seeing as your client can't be hijacked by others than and be forced to execute malicious patches – or by changing SFileAuthenticateArchiveEx() which only is a wrapper for Blizzard :: Mopaq::SFileAuthenticateArchiveEx(), which sets up a RSA / SHA-1 signature structure which is then comparing the actual signature of the MoPaQ with the signature in the given *signaturefile*.

Changing SFileAuthenticateArchiveEx() instead of CGlueMgr::PatchDownloadApply() has the advantage of also enabling custom surveys, which can be streamed to the user on login and should therefore be chosen to be patched. You can see the C++ version of SFileAuthenticateArchiveEx() in listing 3 and the assembler version in listing 4. As you easily can see, the if needs to be removed and ∗authresult = authresult_temp; needs to be changed into ∗authresult = 5;. As it is easier just rewriting that function than modifying it, I suggest patching it to be looking as seen in listings 5 and 6.

Listing 2: void CGlueMgr::PatchDownloadApply()

```
void CGlueMgr::PatchDownloadApply()
{
  int reason_for_failure = 5;

  char old_cwd[PATH_MAX];
  OsGetCurrentDirectory (sizeof (old_cwd), old_cwd);
  OsSetCurrentDirectory (OsFileGetDownloadFolder());

  m_deleteLocalPatch = false;

  Blizzard::Mopaq::HSARCHIVE__* archive = NULL;

  if (SFileOpenArchive ("wow-patch.mpq", 100, 0, &archive))
  {
    Blizzard::Mopaq::AuthResult authresult;
    SFileAuthenticateArchiveEx ( archive, &authresult
                               , &modulus, sizeof(modulus)
                               , &exponent, sizeof(exponent)
                               );

    if (authresult > 4)
    {
      if (PatchDownloadExecutePrepatch (archive))
      {
        SFileCloseArchive (archive);
        archive = NULL;

        if (m_deleteLocalPatch)
          OsDeleteFile ("wow-patch.mpq");

        OsSetCurrentDirectory (old_cwd);

        QuitGame();
        return;
      }
      else
      {
        reason_for_failure = 6;
      }
    }

    SFileCloseArchive (archive);
    archive = NULL;
  }
  else if (SErrGetLastError() == 2)
  {
    reason_for_failure = 4;
  }

  PatchFailed (reason_for_failure, 0);
  OsDeleteFile ("wow-patch.mpq");

  OsSetCurrentDirectory (old_cwd);
}
```

Listing 3: bool SFileAuthenticateArchiveEx()

```
bool SFileAuthenticateArchiveEx ( Blizzard::Mopaq::HSARCHIVE__ *archive
                                , Blizzard::Mopaq::AuthResult *authresult
                                , const unsigned char *modulus
                                , unsigned int modulus_length
                                , const unsigned char *exponent
                                , unsigned int exponent_length
                                )
{
  Blizzard::Mopaq::AuthResult authresult_temp;

  bool result (true);

  if (!Blizzard::Mopaq::SFileAuthenticateArchiveEx ( archive, &authresult_temp
                                                   , modulus, modulus_length
                                                   , exponent, exponent_length
                                                   , "ARCHIVE"
                                                   )
     )
  {
    SErrSetLastError(Blizzard::Mopaq::SFileGetLastError());
    result = false;
  }

  *authresult = authresult_temp;
  return result;
}
```

Listing 4: Assembler version of bool SFileAuthenticateArchiveEx()

```
_SFileAuthenticateArchiveEx proc near
  authresult      = Blizzard__Mopaq__AuthResult ptr -0Ch
  authresult_temp = Blizzard__Mopaq__AuthResult ptr  0Ch

```

```
55                          push     ebp
89 E5                       mov      ebp, esp
83 EC 38                    sub      esp, 38h
C7 44 24 18 FC 38 1E 01     mov      dword ptr [esp+18h], offset aArchive ; "ARCHIVE"
8B 45 1C                    mov      eax, [ebp+1Ch]
89 44 24 14                 mov      [esp+14h], eax   ; exponent_length
8B 45 18                    mov      eax, [ebp+18h]
89 44 24 10                 mov      [esp+10h], eax   ; exponent
8B 45 14                    mov      eax, [ebp+14h]
89 44 24 0C                 mov      [esp+0Ch], eax   ; modulus_length
8B 45 10                    mov      eax, [ebp+10h]
89 44 24 08                 mov      [esp+8], eax     ; modulus
8D 45 F4                    lea      eax, [ebp+authresult]
89 44 24 04                 mov      [esp+4], eax     ; authresult_temp
8B 45 08                    mov      eax, [ebp+8]
89 04 24                    mov      [esp], eax       ; archive
E8 84 99 00 00              call     Blizzard::Mopaq::SFileAuthenticateArchiveEx
; *authresult = authresult_temp;
8B 4D F4                    mov      ecx, [ebp+authresult_temp]
8B 55 0C                    mov      edx, [ebp+authresult]
89 0A                       mov      [edx], ecx
; result = true;
BA 01 00 00 00              mov      edx, 1
; if (!Blizzard::Mopaq::SFileAuthenticateArchiveEx (...))
84 C0                       test     al, al
75 0F                       jnz      short return_now
E8 8E E4 FF FF              call     Blizzard::Mopaq::SFileGetLastError
89 04 24                    mov      [esp], eax
E8 86 D3 E5 FF              call     _SErrSetLastError
; result = false;
31 D2                       xor      edx, edx

                        return_now:
89 D0                       mov      eax, edx
C9                          leave
C3                          retn
_SFileAuthenticateArchiveEx endp
```

Listing 5: proposed patch for bool SFileAuthenticateArchiveEx()

```
bool SFileAuthenticateArchiveEx ( Blizzard::Mopaq::HSARCHIVE__ *archive
                                , Blizzard::Mopaq::AuthResult *authresult
                                , const unsigned char *modulus
                                , unsigned int modulus_length
                                , const unsigned char *exponent
                                , unsigned int exponent_length
                                )
{
  *authresult = 5;
  return true;
}
```

Listing 6: Assembler version of proposed patch for bool SFileAuthenticateArchiveEx()

```
_SFileAuthenticateArchiveEx proc near
  authresult       = Blizzard__Mopaq__AuthResult ptr -0Ch

55                          push    ebp
89 E5                       mov     ebp, esp
83 EC 38                    sub     esp, 38h
; *authresult = authresult_temp;
B9 05 00 00 00              mov     ecx, 5
8B 55 0C                    mov     edx, [ebp+authresult]
89 0A                       mov     [edx], ecx
; result = true;
B8 01 00 00 00              mov     eax, 1
C9                          leave
C3                          retn
_SFileAuthenticateArchiveEx endp
```

## 5 Applying the patch

The executable run from the MoPaQ is where you update the client. Myself, I wrote a quick application in C++ that renames the *WoW.exe* (as a backup) then writes files from the *wow-patch.mpq* into a *patch.mpq* and deletes the *Cache/* folder as well as the *wow-patch.mpq*. It also writes a new executable *WoW.exe*, which has an updated build number and then starts the new *WoW.exe*. Now the client should be fully up to date and be accepted by your server's check and proceed to log in without additional patching.

## 6 Updating the client version

There are different locations, where the build number is referenced in the client. There is a string variant to be written onto the login screen. This one is supplied by `int Script_GetBuildInfo(lua_State *)`. You can easily find the location to modify via just searching for the number given on the login screen as string. You will come up with two locations: One where the build number is stored and another one where the version as well as build-date is. These are all only for logging and to show the user which version he has and should be set by you to help the user identify the version. The actually relevant build number for patching is in `RealmConnection::HandleAuthChallenge()`.

Offsets specific to build 12 340 (patch 3.3.5a) can be found on OwnedCore.

I would always advise making a back up of your WoW executable before attempting to modify anything. You should use incrementing numbers above 12 340, to be able to supply incremental patches. Every patch should have a different build number, of course.