Analysis of Netflix's security framework for '*Watch Instantly*' service

Pomelo, LLC Tech Memo
March – April 2009

Table of Contents

A brief introduction and methodology

Netflix's video-on-demand service offers its subscribers the possibility to watch over 12,000 titles online through their Internet connection. It works on a PC/Mac with a browser-based player created by Netflix (written in Microsoft's Silverlight), or via a Netflix-ready external player such as a Roku Box, an Xbox 360, the Samsung Blu-ray player, or any other Netflix-enabled device.

Both kinds of players (browser-based and external) communicate with a set of Netflix servers that enforce security constraints, provide playback information, and stream the video files to the player. That communication is based on the HTTP and HTTPS protocols, and the message exchange can be observed at different times during playback and be used to make inferences about the kinds of mechanisms that Netflix must have in place in order to provide a secure video experience.

Our analysis was centered on the browser-based playback experience. We created a new Netflix account and observed the message exchange using Firefox and Tamper Data, a Firefox plug-in that traps HTTP and HTTPS requests and responses to/from the server and displays their content, headers, and other useful information[1].

We also studied briefly the behavior of external players. Precisely because those interactions occur outside of a web browser, we were unable to use any high-level tools such as Tamper Data to analyze the message exchanges. We had to rely instead on Unix packet sniffers (mainly tcpdump and Wireshark), which provide less detailed information and, in particular, make it much harder to decrypt HTTPS messages (especially since in these cases the key is locked inside the device!). Even though we were unable to analyze the messages as deeply as for the browser-based players, we still were able to detect many similarities and few differences between browser-based and non-browser based players, and we detail them at the end of this report.

[1] Thanks to Adam Judson, the creator of Tamper Data, for his tool and for the help he provided during this exercise.

Security requirements, control mechanisms, and distributed architecture

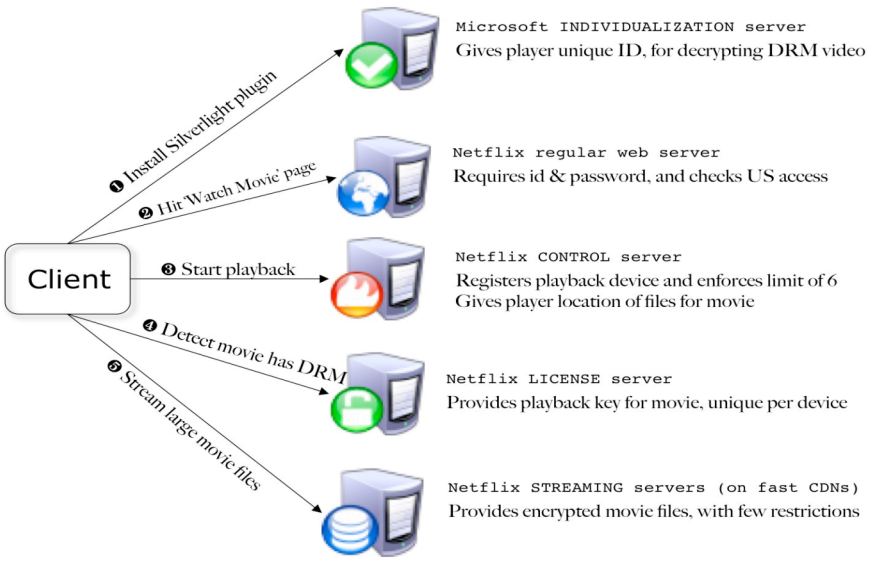
Netflix concerns, from a security point of view, are:

- That only users with the correct (*unlimited*) plan can access content
- That users don't share their account information with others
- That the video is only accessible from within the US, due to licensing restrictions
- That the video content cannot be redistributed and played at a later time

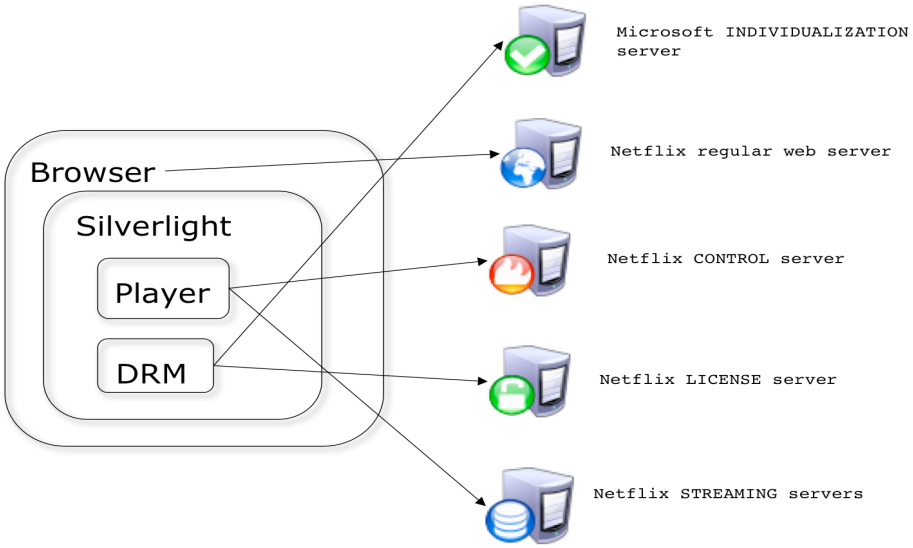
Those restrictions are enforced by a set of coordinated mechanisms:

- By requesting user authentication before playing the video,
- By allowing a maximum of six playback devices (browsers, Roku, etc.) per account,
- By rejecting requests that come from IP addresses outside of the US range,
- By encrypting the video content, and
- By providing unique decryption keys per movie and device

The security checks take place over the duration of the playback experience, and delegated to several different Netflix components and servers, each with distinct responsibilities. The following diagram shows the steps required to play a video on Netflix (slightly out of order, for clarity), the different servers involved in each process, and their respective responsibilities.



Security responsibilities are also distributed on the client side, between the browser (which stores and sends cookies), the Netflix player (which follows the playback and security protocols described above), the Microsoft Silverlight plug-in in which the player is implemented (which prevents cross-site scripting and makes sure that the player only talks to Netflix servers), and a proprietary Microsoft DRM component (which gets an individualized key for each player and coordinates with the License server to acquire keys that are unique to the movie and the player).



This architecture provides several advantages:

- Because all playback devices are registered and associated with an account, Netflix can prevent abusive users from sharing their credentials with others; if they do so, their account will quickly run out of the maximum of allowed devices. (Note that here a device corresponds to a user on a PC/Mac, or a Roku, Blu-ray, etc. external device. Two browsers on the same machine/user are one device, but two users on the same machine are considered to be two separate devices.)

- The encryption of the video files ensures that only players with knowledge of the key and the decryption algorithm (proprietary Microsoft technology) can view them.

- If one of those decryption keys is compromised then the security threat is minor, because the keys are individualized and can only be used by one player.

- Thanks to this distribution of tasks, different servers and client components can focus on the particular constraints that they have to enforce, and assume that other constraints are checked somewhere else. At the same time, and because information is shared between the different servers (through cookies and extra headers in the HTTP requests), security on each of these components can be tightened as desired, if/when Netflix detects abuses or security breaches at a particular point.

- Because all communication takes place using well known HTTP and HTTPS protocols, Netflix has been able to easily provide their video-on-demand service to non-browser devices (such as Roku or Xbox 360) simply by implementing a player on the client side. All these non-browser players communicate with the same servers and follow more or less the same protocol as the browser-based player.

- Since all authentication and authorization checks are performed before playback begins, and since the content of files is encrypted, the streaming of the video files can be easily delegated to CDNs, gaining increased bandwidth without compromising security.

- Finally, because the playback experience is controlled through Netflix's proprietary player, the enforcement of security constraints can be hidden almost completely from users. All they are required to do is log into the site using their account information. The registration and authentication of the device, and the acquisition of playback licenses, all happen behind the scenes.

Step by Step Walkthrough of a typical playback session

From a logical point of view, we can identify five different conceptual phases in the successful playback of a movie:

1. User Authentication,
2. Device Authorization,
3. Instruction Fetching,
4. License Acquisition, and
5. Playback.

We give here a brief overview of those phases, and include a list of the HTTP messages that are exchanged by clients and servers during each phase. For a complete and detailed trace of those HTTP messages we refer the reader to an Appendix distributed separately as a text file.

User Authentication

User Authentication involves making sure that the viewer is indeed a Netflix subscriber and has the right playback privileges. It is mostly dealt with at the browser level, via cookies. The usual mechanisms apply here: users login to their accounts and server sets the cookies that it needs (`netflixShopperID` and `netflixShopperSecret`). At this point Netflix also verifies that the user is visiting from within the United States, by checking the IP address of the request (in fact, Netflix won't display the Play Movie page if the request is not coming from the allowed geographical area).

The process at this stage is the following:

1. Browser requests a movie page

```
GET http://www.netflix.com/WiPlayer?movieid=xxx
```

The server redirects to the login page if the necessary cookies are not present in the request, and also checks at this point that the request is coming from a valid geographical area. If everything is OK, it renders the Movie page.

2. Javascript on the browser checks that Silverlight plug-in is installed. If not, it shows and error message and prompts user to download and install it. Let's assume it is installed.

3. Browser downloads the Silverlight player from

```
GET http://www.netflix.com/pages/watchNow/ \
            player/silverlight/SLPlayer.xap
```

4. Silverlight player notifies that it's ready to play, by sending a message to
```
POST http://www.netflix.com/SilverlightEvent
```

Device Authorization

Device Authorization is necessary to identify the device that the user is playing from, and to ensure that the limit of six is not exceeded. It happens once the Netflix user has been authenticated, and it's handled by the Silverlight application (the player) by coordinating with a Netflix Control server (appropriately named `https://agmoviecontrol.netflix.com`) via a series of HTTPS POST requests. The Netflix servers verify that the device is already authorized or, if it's a new device, that the user is not exceeding the maximum number of devices. The cookies created in the previous step are used here, to establish a relationship between the device and the Netflix user's account.

The steps involved are:

5. The player requests cross-site interaction with the Netflix controller by downloading the client access policy file from
```
GET https://agmoviecontrol.netflix.com/clientaccesspolicy.xml
```

6. The browser verifies identity of Netflix controller with Verisign
```
POST http://ocsp.verisign.com/
```

7. The player begins interaction with the Netflix Controller server by sending a 'ping' command
```
POST https://agmoviecontrol.netflix.com/nccp/controller
Server responds OK
```

8. The player checks whether the device is new (unregistered) or whether the registration of this device has expired, by looking for information stored under the Silverlight plug-in directory (on OS X, this happens to be stored in `~/Library/Application Support/Microsoft/Silverlight`)

- If the device is new or its registration has expired, the player sends a command to the Netflix controller and tries to register or renew the registration:
```
POST https://agmoviecontrol.netflix.com/nccp/controller
POST-DATA: 'register', or
           'authenticationrenewal' with ID of device
```

- If the server responds '`success`', processing continues. The device is now registered for playback, and the registration lasts for a certain period of time (approx. 12 hours, we think)

- The server also checks that the ID of the device, if being renewed, actually does correspond to the Netflix user account. If not, it returns a `'Auth Renew called using different customer's shopper ID'` error message, and forces the player to send a new `'register'` command with a new device ID. Another possible error from which the player can recover by sending a new `'register'` command is `'Sequence Number out of order'`

- The server may also return `'Device Limit Reached'`, in which case playback obviously cannot continue: the account already has six registered devices.

If everything goes well and the device is registered and authorized, the server sends back a response message along with a `CTicket` header that the client will use from now on in all messages with the server. This `CTicket` must contain (or, rather, reference) information about both the account and the device, which means that from now on the Netflix servers will know exactly who it is talking to.

Instruction Fetching

Instruction Fetching is the step in which the player gets information from the Netflix servers about how to play the movie, and where to get the files. It happens after the device has been authorized; the player exchanges a few more messages with the Netflix Control server and requests among other things the list of URLs where the streaming files are, which it promptly starts downloading. Below are the messages exchanged at this phase:

9. The player requests information about the movie and about the user:

```
POST https://agmoviecontrol.netflix.com/nccp/controller
HEADER: X-CTicket=AQAAAA...
POST-DATA: 'moviemetadata', with movie ID

POST https://agmoviecontrol.netflix.com/nccp/controller
HEADER: X-CTicket=AQAAAA...
POST-DATA: 'usermoviemetadata', with movie ID
```

The server returns generic information about the movie (title, cast, play time, etc), and information about previous viewings of this movie. The player then uses that information to display controls, and also to skip to the point in the movie where the user left in a previous viewing.

10. The player now requests information about the Streaming Servers, and where to find the streaming files for the movie, via a request to

```
POST https://agmoviecontrol.netflix.com/nccp/controller
HEADER: X-CTicket=AQAAAA...
POST-DATA: 'authorization' (a misnomer), with movie ID
```

The server returns a long *Manifest* message with information about the location of the streaming files. A portion of that response looks like this:

```
<nccp:authorization movie_id="60027274">

    [Information about CDNs, and their relative weight]
    [...]

    <nccp:downloadable>

    <nccp:downloadableid>1811192541</nccp:downloadableid>
         <nccp:size>1067043503</nccp:size>
         <nccp:bitrate>1500</nccp:bitrate>
         <nccp:contentprofile>
                playready-vc1ap-none
         </nccp:contentprofile>

         <nccp:downloadurls>

                 <nccp:downloadurl>
                  <nccp:expiration>1238103632</nccp:expiration>
                  <nccp:cdnid>4</nccp:cdnid>
                  <nccp:url>
                    http://netflix-
                      274.vo.llnwd.net/s/s1/541/1811192541.wmv
                      ?p=55&amp;e=1238103632&amp;
                      h=c2ff68a253b2602e9f905479db3e0d3d
                  </nccp:url>
                 </nccp:downloadurl>

                 [ … download URLs for other CDNs…]
         </nccp:downloadurls>
         <nccp:resolution>
                 <nccp:width>720</nccp:width>
                 <nccp:height>404</nccp:height>
         </nccp:resolution>

       </nccp:downloadable>
       [... more <downlodable> elements for different bitrates]

</nccp:authorization>
```

11. The player starts downloading the streaming files from the locations specified in the *Manifest*, using the CDN recommended by the Netflix controller. (It first has to request cross-site scripting permissions, by issuing a GET request of the `clientaccesspolicy.xml` file on the corresponding CDN's root). For example, the player will request the following file:

```
GET http://netflix-480.vo.llnwd.net/s/s0/548/1527065548.wmv \
    /range/0-23?\
    p=55&e=1238027489&h=b10cffcc1a9dfe4591e18aa6fb9c2e86
```

The files are encoded in .wmv and .wma format[2], and are served by L3, Limelight, or Akamai. (A parameter in the *Manifest* response from the Netflix Control server specifies which one of the possible CDN to use).

The video for each movie is stored in a single .wmv file (or, to be more precise, one .wmv file for each bitrate in which the movie has been encoded). To stream it, the player sends with its requests an extra HTTP header, `Range-bytes=`, specifying the offset within the movie file that it needs to play (on OS X, Safari doesn't support the header, and so the range is included in the URI, as shown above). It is up to the CDN then to use the OS calls to seek within the large file and return the requested bytes.

At this point there are almost no security checks. The CDNs return the files to anyone who asks for them (even, for example, if the requests come from outside the US). There is only one check: the URIs from the *Manifest* file have an expiration date, as shown in the URI above (the `e` parameter, which has different names and formats depending on the CDN from which the file is delivered). That expiration date is protected by a hash parameter (the `h` parameter above), and is enforced by the CDNs: an attempt to retrieve those files after the expiration date returns `Bad Request`.

License Acquisition

License Acquisition is essential to control DRM-encoded video and audio files. It is the last security step before playback can begin, and it is mostly handled by the Microsoft PlayReady DRM component of the Silverlight plug-in, in collaboration with the Netflix License Server (which is also served from `http://agmoviecontrol.netflix.com`). As the device tries to play the first section of the video files downloaded in the previous step, the player realizes that the content is encoded and that therefore it needs to request a license.

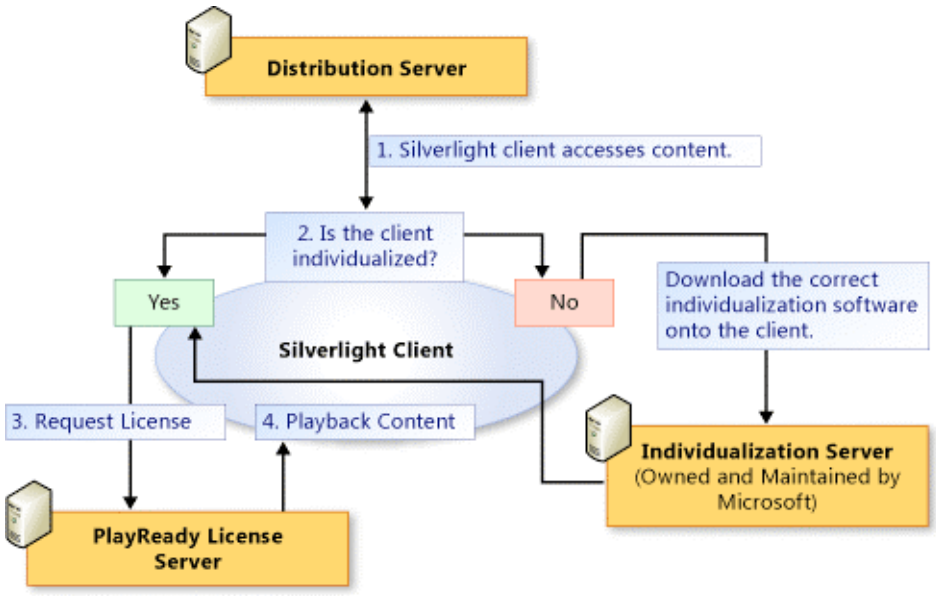
[2] There is a third kind of file, .bif, which contains information for the frames that the player displays as the user tries to fast-forward or rewind during playback.

The **License Acquisition** process is a bit tricky, and it is at the core of the security model, so it is important to understand well how it works and it manages to provide keys that are unique to a given device

The process is mostly handled by the proprietary PlayReady DRM component in the Silverlight plug-in. Here is how Microsoft explains the functioning of this technology (except from "Using Silverlight™ DRM, Powered by PlayReady®, with Windows Media® DRM Content"):

> *The following steps must be taken for a Silverlight application to acquire and consume protected content, whether the content is protected using WMDRM or PlayReady.*
>
> *Step 1: The Silverlight client application makes a request to the distribution server for a piece of content. [...] The distribution server receives the request and sends the content, encrypted, to the client.*
>
> *Step 2: The client reads the clear header of encrypted content and determines that the content is encrypted. To decrypt the content, the client must receive a key from a license server located by a license acquisition URL (LAURL). The client attempts to get a license by sending a license challenge to the LAURL.*
>
> ***If this is the first time the user has attempted to access protected content on this machine, individualization must occur before requesting a license. Individualization is the process of acquiring the individualization component, a software component embedded into the Silverlight plug-in, to handle requesting licenses and protecting sensitive data used in the decryption process. To individualize, the client will send a request to the Microsoft Individualization Service and is returned the individualization component. [At that point, the player can proceed to the next step and request the license from the license server]***
>
> *Step 3: The license server receives a request for a license from the PlayReady client, performs the necessary authentication checks to verify identity, executes business logic to ensure that the user is authorized to consume the requested content, and issues a license to the client with the appropriate usage rights and restrictions.*
>
> *Step 4: The client uses the license to decrypt the content.*

The following diagram (also from Microsoft's documentation) gives a conceptual overview of the process



In Netflix's case,

- the Distribution Server in the diagram above is one of the streaming servers on the edge of the CDN,
- the Individualization Server is Microsoft's server (which is located at `http://services.silverlight.microsoft.com/`), and the
- PlayReady License Server is the same as the Netflix Control server (`https://agmoviecontrol.netflix.com/nccp/controller`).

In concrete terms, these are the messages exchanged at this point:

12. The PlayReady DRM component first checks whether the current user/machine has been '*individualized*', by looking for a directory named DRM/ under the Silverlight plug-in directory.

As explained above, if the machine hasn't been individualized yet, the PlayReady component requests an individual packet by communicating with a Microsoft Individualization Server, by passing a `ClientInfo` parameter that it computes based on some information that depends on the client machine (the parameter is always the same on the same machine, regardless of which user is logged in at the time). For example, it might send

```
POST http://services.silverlight.microsoft.com/ \
          Silverlight_RTM/default.jolt?Individualize
POST-DATA: ClientInfo=AAAARgAAAAEAAAAHAAAAAQAAADJRA6csu74ALv[…] ==
```

The response from the Microsoft server is approximately 500KB, including an individualized key for the machine but also the actual PlayReady component library. The result is saved in a directory named DRM/ under the Silverlight plug-in directory.

13. The PlayReady DRM component tries to get a license from the Netflix License server. It sends a message requesting a license for the corresponding movie, and requesting that the response be encrypted using a public key that the component generates.

```
POST https://agmoviecontrol.netflix.com/nccp/controller
HEADER: X-CTicket=AQAAAA...
POST-DATA: 'license', with stream ID
```

14. The server finds the license for the requested element, encrypts it using the specified key, and sends it back.

15. The PlayReady DRM component now decrypts the license using its private key, and uses that license to decrypt the streaming file

In summary: the request to the Netflix License Server includes the public key of the individualized component, and the response from the server returns the license key *encrypted using the individualized component's public key*. This means that only that particular player can decode the message (using its private key), to extract the *original* license key for the movie and then use it to unlock the encrypted content.

We have noticed that both the request for the license and response from the server are constantly changing (even for the same machine, user, device ID, and movie). This leads us to believe that the Silverlight plug-in is generating a new Public/Private key pair every time it requests a new license, using perhaps a combination of its individualization code, the device ID as recorded by Netflix, and a timestamp.

Playback

Playback can finally take place once these three groups of constraints have been enforced. The player talks directly to the Netflix Streaming servers, which in turn assume that all security concerns have already been taken care of by the other components. Those servers will give content to whoever knows the correct URIs (but only within a pre-specified window of time), knowing that because what they deliver is encrypted, only authorized devices with the proper license will be able to play it. An important advantage of this model is that the

streaming servers can be dumb, sitting at the edge of CDNs and delivering content without almost any intelligence in them.

The messages exchanged are:

16. Once the content has been decrypted, control returns to the Silverlight player, which starts the playback, and continues downloading. The player notifies that it's happy by sending another message to

```
POST http://www.netflix.com/SilverlightEvent
```

At this point all the handshake has been performed, and the player continues to download the streaming files and to show the movie. It also contacts the Netflix servers regularly (every minute or so) and sends playback data and other (encrypted) information. The message is :

```
POST https://agmoviecontrol.netflix.com/nccp/controller
HEADER: X-CTicket=AQAAAA...
POST-DATA: 'logblob', with logging data AND sequence number
```

External devices

As we said before, the playback session is almost identical for browser-based and external playback devices. Here is what happens when the user plays a movie from an external device such as a Roku player, an Xbox 360, or a Samsung Blu-ray player:

1. The first part is establishing the relationship between the box and the user's account. After it boots, the box realizes that it is not authorized and it requests a code from the Netflix server. The server gives it a unique 4-6 alphanumeric string and the player shows on the TV screen. At that time, the box starts pinging the Netflix server and sending that code to it. The user has to go with their web browser to Netflix (`/activate`, after logging in) and enter the code, which the Netflix server stores in its database. At some point the Netflix server will get one of those pings from the box and it will find the corresponding code in the database, and it can therefore establish the connection between the box and the user's account. If, for example, the user enters the code in the browser but turns off the box, there is no box sending the pings and the registration cannot complete --the pings are necessary.

2. If after power-on the box detects that it was already registered, it either accepts the registration information (if it hasn't timed out), or it renews it from the Netflix server. This is the same as from the browser. This is where it checks that the device is still active (if not, it goes back to step 1 above), and that the user doesn't already have six devices on his/her account.

3. What follows is a series of messages between the box and the server to retrieve account information and get the movies from the user's Instant Watch queue. Also, as one moves through the movies in the queue, the box retrieves the high-definition box art and information about the particular movie (files, where the user left watching, etc.). This is different from the way it happens from the browser, where all the relevant messages are exchanged once the movies is already selected. We believe, however, that the information is the same.

4. Once the movie starts playing, the behavior is similar to that on the browser. The actual content files are downloaded, and when the box starts playing it realizes that the files are DRM-protected, talks again to the Netflix server, and gets a license to play the movie. The content files are exactly the same as for the Silverlight player: .wmv files, served from the same location from the same CDNs, and also requested via HTTP using the Range-bytes header. There is no individualization process here, because (we assume) the box is already individualized from factory --i.e., it comes with its own key. There's no need to talk to the Microsoft server.

5. Finally, there's also a regular message sent to the server every 30 seconds or so, with logging information. It is the same as from the browser.

In summary, most of the behavior is similar to that on the browser:

- The device gets information from the Netflix control server
- The device needs to be registered and connected to an account, so that Netflix can enforce the corresponding constraints
- The content files are exactly the same, live on the same place, and are requested using the same HTTP methods
- The content files are DRM-protected, and are also decrypted with a key from the Netflix control server.

There are some differences:

- The registration process is different, of course
- The messages to get information about movies is done once for all the movies in the queue
- There is no Microsoft server for individualization
- The Netflix control server is no longer their usual server located at `https://agmoviecontrol.netflix.com`, but instead a new server at `https://moviecontrol.netflix.com`. This "private" API probably exists so that it's more practical to isolate the external devices that talk to it. These devices likely have stronger security, such as protected (physically) keys to sign messages. Netflix can provide better stability ensuring that only authentic Netflix-capable devices are the only HTTP clients that can pass through into their web services stack. The browser software stack is far more exposed.

We'd like to point out again that the messages from/to the Netflix control server travel via HTTPS, and are therefore encrypted. When we were examining the same messages from the browser, we had tools to look at the actual requests **after they had been processed and decrypted** by the browser. With an external box all we can do is just sniff raw packets, and without the box's private key we have no way to decrypt them. In practical terms this means that we don't know the exact nature and content of the messages; we are just inferring that they behave in similar ways to what we observed in the browser[3].

[3] In particular, we don't know whether the license key to play the movie is unique to each box, unique to each device kind (Blu-ray, Roku, etc.), or global for everything that talks to `https://moviecontrol.netflix.com`. We believe that it's the first case (unique to each box), because they obviously have that capability for the case of the individualized Silverlight player.

Summary of important URIs and security measures

Important URIs

URI	Purpose
`http://www.netflix.com/activate`	Authorizes external device
`http://www.netflix.com/Player/unregister_device`	De-authorizes a device
`http://www.netflix.com/WiPlayer?movieid=xxx`	Page to watch a movie
`http://www.netflix.com/SilverlightEvent`	Player POSTs here information about its state
`https://agmoviecontrol.netflix.com/ \ nccp/controller`	Location of Netflix Control server. Also Netflix License server
`http://services.silverlight.microsoft.com/ \ Silverlight_RTM/default.jolt?Individualize`	Location of the Microsoft Individualization Server
`http://netflix-480.vo.llnwd.net/ \ s/s0/548/1527065548.wmv`	Example location of a movie file

What happens if you try to break the security?

If you…	Then…
Share account credentials	Only up to 6 devices will be allowed
Visit from outside the US	The movie page will refuse to load
Playback from an unauthorized device	The device will be authorized automatically
Attempt to authorize more than 6 devices	Authorization will be rejected, and movie won't play
De-authorize a device after playback starts	Device will play movies for 12 hours, then be re-authorized (up to 6 devices)
Access video files directly (via web)	Content will be DRM protected, and only a player with a valid license will play
Somehow acquire the DRM license key	The key will be valid for just a single device; no other device will accept it