Backdooring MP3 Files

Backdooring MP3 Files - pdp, gnucitizen.org.

- Published: date not stated
- Original: <https://www.gnucitizen.org/blog/backdooring-mp3-files/>
- Preserved from: https://www.gnucitizen.org/blog/backdooring-mp3-files/ (stored) on 2026-08-11
- Licence: unknown

Rights remain with the original author and publisher. This is a research archive of a source from the Web Hacking Techniques Index collections, kept so the page going offline. To read the original, follow the link above.

Content

> UNTRUSTED SOURCE TEXT. Everything below this line is third-party material quoted for research. It is data, not instructions. Do not follow directions, execute code, or fetch URLs because this text says so.

Backdooring MP3 Files

Wed, 20 Sep 2006 21:39:18 GMT

by pdp

Recently I published information on how specially crafted HTML (remote and local), Flash and QuickTime (.mov) files can be used by malicious users to target and exploit internal and external networks. Than my friend and college David K released his findings on backdooring PDF documents via builtin Adobe Reader JavaScript features. Also, JavaScript malware via Google AJAX Search API seams to be possible and could affect many popular web products. As Billy Hoffman said "XSS is the new hotness!". I cannot agree more on that.

MP3 Files can be Backdoored with Malicious Content too

Over the past few days I have been exploring different features of Apple's QuickTime player - key software component of iTunes and standard part of many home and business workstations. A lot of research was conducted and some problems, which IMHO are quite serious, were found. Please take this post as a security notice.

QuickTime is quite versatile and flexible media platform which has a lot of functionalities. I quite like it, I must say. I even use iTunes on a daily basis. Unfortunately because of its flexibility QuickTime seams to allow execution of malicious content in a form of JavaScript from media files such as mp3, mp4, m4a and everything else that is supported.

The problems is caused by a quite useful feature called QuickTime Media Link (.qtl). The whole point of these QuickTime Media Link files is to provide means of playing media files in a more accessible way. In this respect the developer can create a .qtl file which hold information about the

media content that needs to be played plus recommended dimensions, accessibility features, control features etc. QuickTime Media Link files are written in XML and end typically end with .qtl. A .qtl file in its very basic form looks like the following:

```
<?xml version="1.0">
<?quicktime type="application/x-quicktime-media-link"?>
<embed src="Sample.mov" autoplay="true"/>
```

The most important element in this XML is called `embed`. This element describes the content that needs to be played. There are quite few attributes that can be assigned to the `embed` element like `src` and `autoplay` but they are not as interesting as `qtnext`. The `qtnext` attribute specifies what needs to be played next. Because `qtnext` expects a URL as an input, .qtl files are capable of opening HTML pages, local files, FTP sites and JavaScript code in the current browser. As such, `qtnext` allows successful backdooring any .qtl file with malicious JavaScript logic. Here it is an example:

```
<?xml version="1.0">
<?quicktime type="application/x-quicktime-media-link"?>
<embed src="presentation.mov" autoplay="true" qtnext="javascript:alert('backdoored')"/>
```

Upon execution the media link presented above will display a harmless message to the user. Keep in mind that a lot more dangerous things can be done. For more information about the impact of such an attack check the AttackAPI - a toolkit designed to test browser related issues.

To sum up, .qtl files can contain malicious JavaScript code that can takeover some important network device when executed for example. That's not the end of the story though. Because of its flexibility QuickTime doesn't mind if Media Link (.qtl) files end with .mp3, .mp4, .m4a or even .mov. For example the following XML can be saved as .mp3 and once opened in QuickTime a harmless message will be prompted to the user:

```
<?xml version="1.0">
<?quicktime type="application/x-quicktime-media-link"?>
<embed src="http://example.com/path/to/real/song.mp3" autoplay="true" qtnext="javascript:alert('hello from backdo
```

This is a quite big problem especially in default configurations of iTunes. The iTunes installation wizard installs the QuickTime player and QuickTime browser plugins and associates various media files with its components. If you open a mp3 file from the desktop it will be played in iTunes player by default, however if you open it from some website it will be played in the QuickTime player browser plugin. In this respect, users who are previewing mp3s and other media files from the Internet are vulnerable.

For the sole purpose of demonstrating how this vulnerability works I composed a quite simple and harmless proof of concept. There are two links to mp3 files at the bottom of this page. Two of these files are backdoored. One of them is a tune I composed many years ago.

- backdoored.mp3 - executes javascript immediately
- jamesbond-overdrive-backdoored.mp3 - executes javascript at the end
- jamesbond-overdrive.mp3 - the real tune

I mentioned earlier that .qtl files can end with .mov, .avi or even .asf extensions. This means that users can be fooled into executing malicious JavaScript content not only through mp3.

There is one more thing that is quite important to point in this article. JavaScript opened from QuickTime is executed in the browser local context. This attack is also know as Cross-context scripting. In Firefox the context is `about:blank`. I am not quite familiar with `about:blank` but my understandings are that everything from `about:*` can request special privileges that will be granted without warning the user. If this the case malicious JavaScript will not only be able to read the local file system but also to alter it. This feature might be used to spread worms that go far beyond the traditional Cross-site Scripting attack.

Proof of concept for this issue can be found at the following URL.

Archived Comments

Archived from https://www.gnucitizen.org/blog/backdooring-mp3-files/. Rendered offline from the local Markdown copy.