
# Inglourious Hackerds: Targeting Web Clients

## Black Hat Briefings
## Washington DC, USA 2010

**OUDOT Laurent, CEO & Founder, TEHTRI-Security**

TEHTRI-Security
Technology-Ethical-Hacker-Trust-Robust-Information-Security

## Abstract

This tiny paper focuses on offensive technologies abusing web clients. We propose to look at technical security issues related to multiple Internet Web Clients.

As we all know for years, while those tools are widely used to crawl the Net and retrieve information, there might exist many scenarios where evil attackers can abuse them.

By studying the protocols (HTTP, etc), and by doing some kind of fuzzing operations, TEHTRI-Security was able to find multiple security issues on many handled devices and workstations.

This would take more than just a talk to disclose all our new offensive concepts and our tricks, but here we will try to share interesting things related to how evil attackers can become anonymous and create cover channels based on web clients, or like how to own or crash most famous current web clients and devices.

# Finding vulnerabilities

## Fuzzing

To quote many security experts, fuzzing is like art, because you have to create a strong technical basis, and then to sort the results and to find how to improve your research and to get real results. When you think about fuzzing against web clients, you have many possibilities of attacks.

For our research, we based our first tests on HTTP issues, and contents issues. For example, though it might sound useless, what would happen with a typical HTML web page containing weird stuff, etc?

Thanks to our humble platform, we were able to find 0days against many devices and products involved in HTTP flows, as we will see further.

## Penetration tests

Fuzzing is not enough. You also need to do things manually, and to go beyond automatic stuff that your offensive robots would not find themselves. It's like for the chess game, computers cannot replace humans (yet?). So we decided to apply the methods and concepts used against the infrastructures of our customers during penetration tests.

You have two kinds of pentesters. Those who will proudly call themselves certified something, and who will apply rules, books, methods, etc with a great serious. And those who will try to discover unknown paths in your wires and systems (+ 0days…).

Both might be extremely useful. When you audit a classical IT architecture, with no home based stuff, etc, maybe the corporate classical pentester will help a lot.

But when you want to find unknown vulnerabilities against IT products, moving like a more creative pentester might really help, because you'll want to check how far you can push pressure on some parts, etc. Everything is a problem of behavior, and the more you check things, the more you'll be able to see if those things are like they were supposed to be.

Thanks to those methods, we also found 0days in some products, as we'll see in next chapters.

# Client Side Attacks

## Becoming anonymous by abusing web clients

We would like to introduce a new kind of class of client side attack. The goal of those attacks is to abuse web clients, talking HTTP, in order to ask them to do things for you, like moving information from a point to another. Thanks to that, those web clients will do the dark job for you, and you'll remain anonymous. They are our sleeping partners, involved in our means.

How can we abuse HTTP clients for such a purpose? Let's take a simple example. You have a web site A, and another web site B. You want to transfer data between sites, from time to time, with asynchronous behavior. And you want it to be anonymous and automatic.

One way to achieve this goal is to think about vulnerabilities with a kind of control of a client, through XSS vulnerabilities, etc.

Let's imagine that on site A, you put a line like this:

```
<IMG SRC="http://site-b/foo.asp?exchange=data-to-send">
```

A client coming here will then ask site B for this picture (HTML tag IMG). On site B, a request is made to "/foo.asp" with a GET argument added "exchange=data-to-send".

There will be issues with the limitation of size for the GET arguments, and there will also be issues with stealthiness. For example, as it's a GET, on site B there will be tons of lines of logs with the data. Moreover, if the web client coming on site B is using a proxy, there will be lines of logs in the proxy itself too.

For the size and GET issues, we could use a POST argument, by abusing a client compatible with javascript, so that we automatically ask to launch an HTTP request, exactly like with CSRF/XSS attacks. But playing with javascript requires javascript enabled, and it's a bigger interaction than just putting an IMG tag and a web page (of course, IMG is not the only possibility).

For the lines of logs in the proxy, this might be a problem because of the HTTP field called REFERER. If the web client is configured with referrers enabled (which is the default for everybody), the proxy logs will contain the address of site A as a referrer. And this is something you might want to avoid in some situations.

This also means that when you have web clients of paranoid people, coming to your web site, with no Referer, you can dynamically generate a web page with many URL to crawl (through GET or POST), so that they can do nasty things for free for you. So, when

security staff claim that deleting Referer is a nice trick to improve the security of their clients, we could argue that this might sometimes be used by attackers against them too.

Now let's move to another trick. We could call it the loop attack, or something like that. The goal is to abuse an incoming web client, in order to generate an indirect communication between you (site A) and your friend (site B). You'll communicate from site A to site B, with the help of the web client.

When you decide that you can abuse a client, you dynamically generate a web page claiming that this resource has move somewhere else. This can be done with HTTP Redirections (check the protocol if needed)., like with 302, 301, etc, HTTP messages.

Here is an example of a response sent back from Site-A, to an incoming web client:

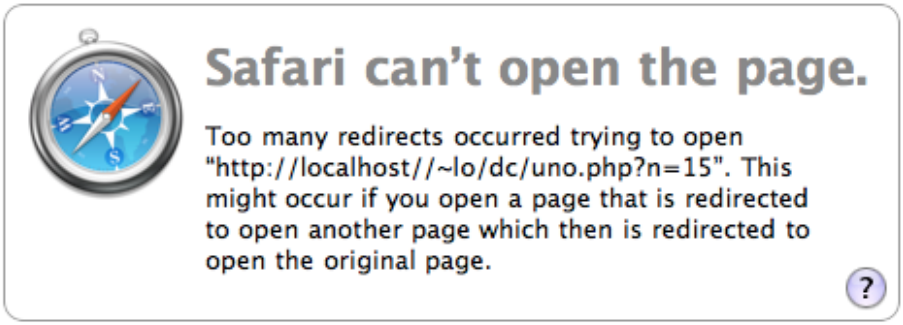
```
HTTP/1.1 302 Found
Date: Thu, 06 Jan 2011 17:31:28 GMT
Server: Apache/2.2.15 (Unix) mod_ssl/2.2.15 OpenSSL/0.9.8l DAV/2 PHP/5.3.3
X-Powered-By: PHP/5.3.3
Location: http://site-b/communication.php?data=XXXXXXXXXX
Content-Length: 0
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Content-Type: text/html
```

Then, the web client will follow the link (unless it's a non standard web client, like default behavior of "curl", etc, which do not follow such redirections by default). By moving the URL given by Site-A (red line in the upper box), the web client will go to site-B and ask for: "/communication.php?data=XXXXXXXXXX"

Now, if Site-B directly answers that the resource is not there, and redirect the client to Site-A (for example), we have created a ping-pong link between site-A and site-B, with the possibility to send data between both servers, through the GET arguments (at least).

This loop attack, which could be classified in client-side attacks, is really interesting when you consider the results given here as an example: depending of the versions/vendors/etc the number of loops allowed would be different. By knowing this number, and by checking the User-Agent sent by the browser, a malicious attacker can dynamically generate a web page with the needed fake redirections up to the maximum allowed. This process can be repeated as much as needed. If the web client supports Javascript, etc, you can also play with reloading resources at the end of a loop transaction.

If you add too much loops for a specific web client, you might generate an error message on the client side, which is not a good idea from a stealth point of view. Here is an example of such a warning for an end-user with a recent MacOSX/Safari browser:




With some tests on some web browsers, we tried to check how much redirections we were able to generate, by incrementing a counter between site-A and site-B, each time the were able to exchange a request thanks to a web client. Here are the results:

| Web Browser | Max Number of loop messages allowed |
|---|---|
| MS Internet Explorer 8 | 9 |
| Apple Safari | 15 |
| Google Chrome | 19 |
| Mozilla Firefox | 19 |
| … | … |
| Internet Explorer 6 | 99 (!) |

As you can see, when you get an incoming Web client with Internet Explorer 6 reaching your web site, you can use this web client to send around 50 messages to a remote web site, and you'll also get around 50 replies from that server. And it's free :-)

Back to the lower layers, especially in the HTTP protocol, what are the fields that could be carried anonymously by our new friendly web browsers? By default, as we just glanced at, we can easily use GET arguments, which might be limited by its size. We also saw that the POST argument might be used, like with XSS attacks, etc.

It might also be able to play with the referrer field, by crafting the HTTP ping-pong with different styles.

From our tests, we also found that some web browsers will carry the Cookies from a web site to another, when they follow a 302 HTTP redirection. This sounds like vulnerability, and of course, cookies created by site-A should not be sent to site-B just because the site-A created a 302 redirection.

Beyond the scope of this tiny humble white paper, and during the BlackHat Washington DC talk, we will disclose such a vulnerability, which can be used as a zero-day exploit for those who would like to play with the vulnerable web browser.

## Exploiting web clients

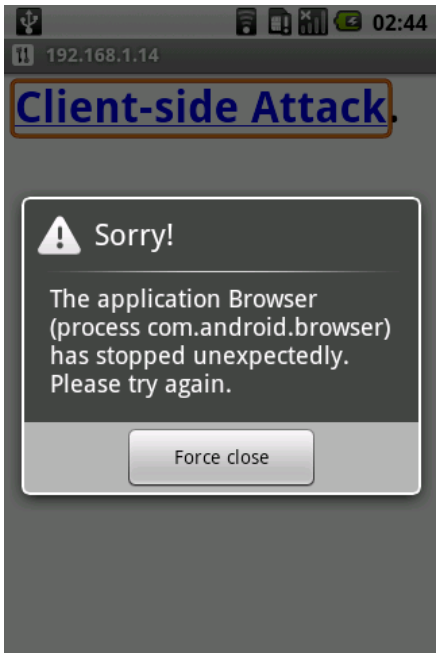
During our talk at Blackhat DC 2011, we will also disclose vulnerabilities based on HTML tricks that can lead to execution or crash against an incoming web client.

Let's check some results or details that can be shared before the event through this white paper.

### Playing with Google Android

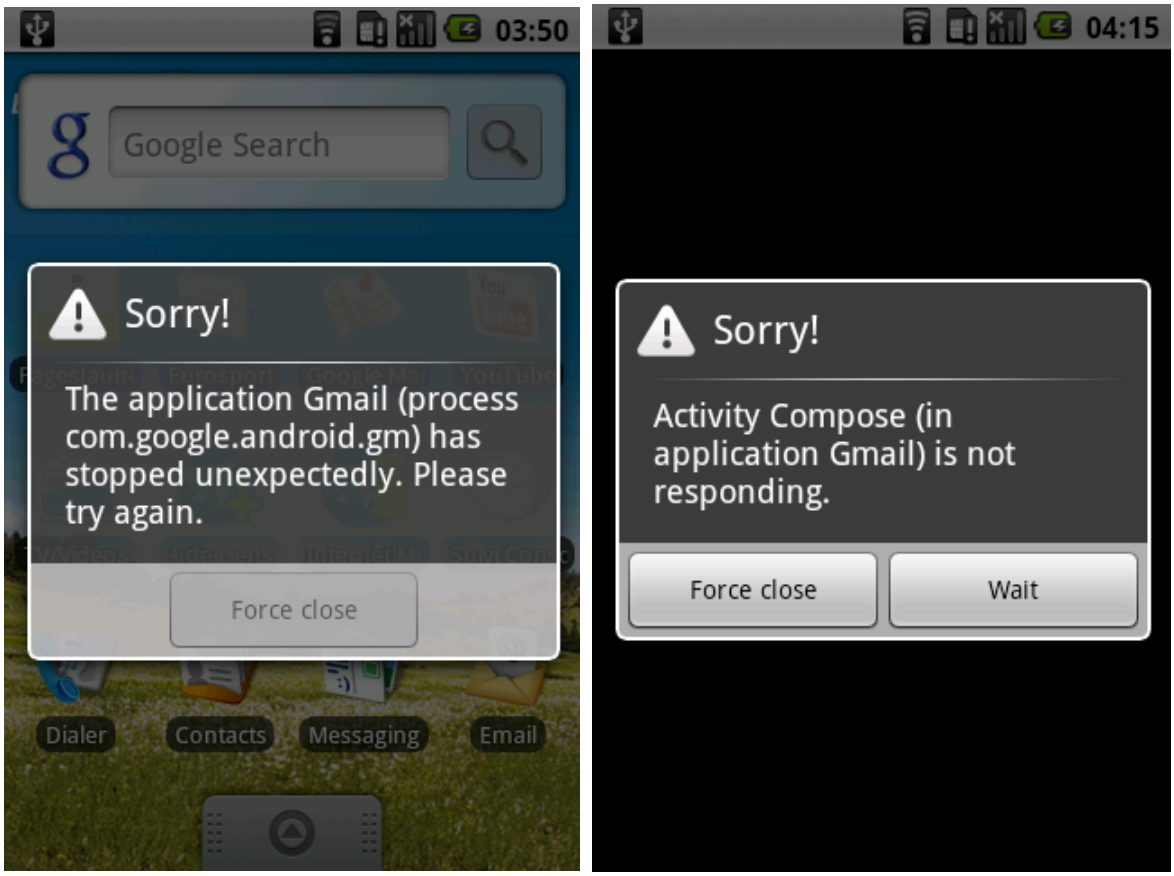
Name: Browser (com.android.browser)
Firmware tested: 1.5
Kernel version tested: 2.6.27
User-Agent tested: Mozilla/5.0 (Linux; U; Android 1.5; en-us; GT-I5700 Build/CUPCAKE) AppleWebKit/528.5+ (KHTML, like Gecko) Version/3.1.2 Mobile Safari/525.20.1

*Example of client-side attack against com.android.browser*



```
04-03 05:12:49.328: INFO/ActivityManager(1847): Starting activity: Intent {
action=android.intent.action.VIEW categories={android.intent.category.BROWSABLE}
data=http://192.168.1.14/webfuzz/handled/android/verified/h.php
comp={com.android.browser/com.android.browser.BrowserActivity} }
...
04-03 05:13:10.135: ERROR/JavaBinder(6505): !!! FAILED BINDER TRANSACTION !!!
...
04-03 05:14:24.418: ERROR/dalvikvm-heap(6505): Out of memory on a 8388612-byte allocation.
...
04-03 05:14:24.453: DEBUG/AndroidRuntime(6505): Shutting down VM
04-03 05:14:24.453: WARN/dalvikvm(6505): threadid=3: thread exiting with uncaught exception
(group=0x4000fe70)
04-03 05:14:24.453: ERROR/AndroidRuntime(6505): Uncaught handler: thread main exiting due to
uncaught exception
04-03 05:14:24.468: ERROR/AndroidRuntime(6505): java.lang.OutOfMemoryError
04-03 05:14:24.468: ERROR/AndroidRuntime(6505):     at
android.text.StaticLayout.generate(StaticLayout.java:138)
04-03 05:14:24.468: ERROR/AndroidRuntime(6505):     at
android.text.StaticLayout.<init>(StaticLayout.java:97)
04-03 05:14:24.468: ERROR/AndroidRuntime(6505):     at
android.text.StaticLayout.<init>(StaticLayout.java:54)
04-03 05:14:24.468: ERROR/AndroidRuntime(6505):     at
android.text.StaticLayout.<init>(StaticLayout.java:45)
04-03 05:14:24.468: ERROR/AndroidRuntime(6505):     at
android.widget.TextView.makeNewLayout(TextView.java:4757)
04-03 05:14:24.468: ERROR/AndroidRuntime(6505):     at
android.widget.TextView.onMeasure(TextView.java:4984)
04-03 05:14:24.468: ERROR/AndroidRuntime(6505):     at
android.view.View.measure(View.java:7115)
04-03 05:14:24.468: ERROR/AndroidRuntime(6505):     at
android.view.ViewGroup.measureChildWithMargins(ViewGroup.java:2875)
04-03 05:14:24.468: ERROR/AndroidRuntime(6505):     at
android.widget.LinearLayout.measureChildBeforeLayout(LinearLayout.java:888)
04-03 05:14:24.468: ERROR/AndroidRuntime(6505):     at
android.widget.LinearLayout.measureVertical(LinearLayout.java:350)
04-03 05:14:24.468: ERROR/AndroidRuntime(6505):     at
android.widget.LinearLayout.onMeasure(LinearLayout.java:278)
04-03 05:14:24.468: ERROR/AndroidRuntime(6505):     at
android.view.View.measure(View.java:7115)
04-03 05:14:24.468: ERROR/AndroidRuntime(6505):     at
android.view.ViewGroup.measureChildWithMargins(ViewGroup.java:2875)
04-03 05:14:24.468: ERROR/AndroidRuntime(6505):     at
android.widget.LinearLayout.measureChildBeforeLayout(LinearLayout.java:888)
04-03 05:14:24.468: ERROR/AndroidRuntime(6505):     at
android.widget.LinearLayout.measureVertical(LinearLayout.java:350)
04-03 05:14:24.468: ERROR/AndroidRuntime(6505):     at
android.widget.LinearLayout.onMeasure(LinearLayout.java:278)
04-03 05:14:24.468: ERROR/AndroidRuntime(6505):     at
android.view.View.measure(View.java:7115)
04-03 05:14:24.468: ERROR/AndroidRuntime(6505):     at
android.view.ViewGroup.measureChildWithMargins(ViewGroup.java:2875)
04-03 05:14:24.468: ERROR/AndroidRuntime(6505):     at
android.widget.FrameLayout.onMeasure(FrameLayout.java:245)
04-03 05:14:24.468: ERROR/AndroidRuntime(6505):     at
android.view.View.measure(View.java:7115)
04-03 05:14:24.468: ERROR/AndroidRuntime(6505):     at
android.view.ViewGroup.measureChildWithMargins(ViewGroup.java:2875)
04-03 05:14:24.468: ERROR/AndroidRuntime(6505):     at
android.widget.FrameLayout.onMeasure(FrameLayout.java:245)
04-03 05:14:24.468: ERROR/AndroidRuntime(6505):     at
android.view.View.measure(View.java:7115)
...
04-03 05:14:24.468: ERROR/AndroidRuntime(6505):     at
android.os.Handler.dispatchMessage(Handler.java:99)
04-03 05:14:24.468: ERROR/AndroidRuntime(6505):     at android.os.Looper.loop(Looper.java:123)
04-03 05:14:24.468: ERROR/AndroidRuntime(6505):     at
android.app.ActivityThread.main(ActivityThread.java:3948)
04-03 05:14:24.468: ERROR/AndroidRuntime(6505):     at
java.lang.reflect.Method.invokeNative(Native Method)
04-03 05:14:24.468: ERROR/AndroidRuntime(6505):     at
java.lang.reflect.Method.invoke(Method.java:521)
04-03 05:14:24.468: ERROR/AndroidRuntime(6505):     at
com.android.internal.os.ZygoteInit$MethodAndArgsCaller.run(ZygoteInit.java:782)
04-03 05:14:24.468: ERROR/AndroidRuntime(6505):     at
com.android.internal.os.ZygoteInit.main(ZygoteInit.java:540)
04-03 05:14:24.468: ERROR/AndroidRuntime(6505):     at dalvik.system.NativeStart.main(Native
Method)
```

*Example of client-side attack against com.android.gm (Gmail)*



### Playing with Apple Safari on Mac OS X

Here is another example of result of exploit example against Safari on MacOSX.

```
Process:         Safari [3254]
Path:            /Applications/Safari.app/Contents/MacOS/Safari
Identifier:      com.apple.Safari
Version:         5.0.2 (6533.18.5)
Build Info:      WebBrowser-75331805~1
Code Type:       X86-64 (Native)
Parent Process:  launchd [3178]
OS Version:      Mac OS X 10.6.4 (10F569)
Report Version:  6
Crashes Since Last Report:           9739
Exception Type:  EXC_BAD_ACCESS (SIGSEGV)
Exception Codes: KERN_PROTECTION_FAILURE at 0x00007fff5df61b38
Crashed Thread:  0  Dispatch queue: com.apple.main-thread

Thread 0 Crashed:  Dispatch queue: com.apple.main-thread
0   com.apple.CFNetwork               0x00007fff859a9686 URLResponse::...
1   com.apple.Foundation              0x00007fff80e7c352 ...
2   com.apple.WebCore                 0x00007fff84d0d26b
WebCore::ResourceResponse::platformLazyInit() + 475
3   com.apple.WebCore                 0x00007fff84d86b91
WebCore::ResourceResponseBase::httpStatusCode() const + 17
4   com.apple.WebCore                 0x00007fff84d91e25
WebCore::ApplicationCacheHost::maybeLoadFallbackForResponse…
…
7   com.apple.Foundation              0x00007fff80e7c0cb
_NSURLConnectionDidReceiveResponse + 123
8   com.apple.CFNetwork               0x00007fff8599facc URLConnectionClient::...
9   com.apple.CFNetwork               0x00007fff85a0649c
URLConnectionClient::ClientConnectionEventQueue::...
…
11  com.apple.CFNetwork               0x00007fff8598d78f
URLConnectionClient::processEvents() + 121
12  com.apple.CFNetwork               0x00007fff8598d56c
MultiplexerSource::perform() + 160
13  com.apple.CoreFoundation          0x00007fff88f3be91 __CFRunLoopDoSources0 +
1361
14  com.apple.CoreFoundation          0x00007fff88f3a089 __CFRunLoopRun + 873
15  com.apple.CoreFoundation          0x00007fff88f3984f CFRunLoopRunSpecific + 575
16  com.apple.HIToolbox               0x00007fff83c1a91a RunCurrentEventLoopInMode
+ 333
…
21  com.apple.Safari                  0x00000001000165d8 0x100000000 + 91608
…
Thread 1:  Dispatch queue: com.apple.libdispatch-manager
…
Thread 2:  WebCore: IconDatabase
…
Thread 3:  Safari: SafeBrowsingManager
…
Thread 4:
0   libSystem.B.dylib                 0x00007fff814712fa mach_msg_trap + 10
1   libSystem.B.dylib                 0x00007fff8147196d mach_msg + 59
2   com.apple.CoreFoundation          0x00007fff88f3a3c2 __CFRunLoopRun + 1698
3   com.apple.CoreFoundation          0x00007fff88f3984f CFRunLoopRunSpecific + 575
…
Thread 5:
0   libSystem.B.dylib                 0x00007fff814b4dce select$DARWIN_EXTSN + 10
1   com.apple.CoreFoundation          0x00007fff88f5be92 __CFSocketManager + 818
2   libSystem.B.dylib                 0x00007fff814aa456 _pthread_start + 331
3   libSystem.B.dylib                 0x00007fff814aa309 thread_start + 13
…
Thread 0 crashed with X86 Thread State (64-bit):
  rax: 0x000000000...
```

### More exploits

At the end, we found security issues against multiple products.

| Product / Vendor | Vulnerability found | Status |
|---|---|---|
| Apple iPhone | **YES** | **CVE-2010-1752** |
| Apple iPod | **YES** | **CVE-2010-1752** |
| Apple iPad | **YES** | **CVE-2010-1752** |
| Apple Safari MacOSX | **YES** | **CVE-2010-1752** |
| Apple Safari Windows | **YES** | **CVE-2010-1752** |
| BlackBerry rim_bb_browser | **YES** | **Almost totally patched** |
| Google Android | **YES** | **Not considered as a security issue** |
| HTC Windows | **YES** | **No response from HTC** |

More info will be disclosed in our slides of this international event.

## Solutions & Conclusions

We tried to show how easy it might sometimes be to find 0days against web clients, etc. This is a proof that on many products, the vendors do not have the time to focus on security as much as they would like, because they are under the business pressure, with lot of things to handle (calendar, objectives, RoI, etc). Does it mean that we'll leave in a world of vulnerable systems? Sure. And the future will become more and more funny, when you look at the number of new interesting technologies.

To quote Obiwan Kenobi, the first conclusion of this humble paper would be: "*Your eyes can deceive you*". With this sentence, we want to remind the reader that cyber warfare is full of unknown techniques and possibilities, and that everything is moving and changing, everyday. You should not trust the vendors and consultants who claim that everything is secure and perfect, that this product is better, etc.

You should just know that trusting is just a temporary mean to take the right decision, and that everything can change, even in a short future. The trouble is that in a corporate environment, you cannot apply big changes on infrastructures each week, just because there is a 0day in the dark market against your products, etc.

So what should we do? Keep analyzing the situation, and keep improving security, not only protection but also containment, with in depth defenses, so that your enemies will be blocked or will move slower somewhere, etc. Sometimes you'll be able to detect them too, or to understand how they moved, etc. Checking your security on a regularly basis, and improving your situation will definitely help too.

It's just the well-known life cycle of IT Security. Plan, Do, Check, and Act. That's more complex than what is said during trainings and in books, and sometimes it might cost a lot, especially when you are unlucky (bad choices in the past, highly sensitive activity with lot of enemies, etc). But it's the price to avoid those unglourious *hackerds*. Take care.