# Web application: Phishing attackers

In the context of the web application, defensive security is applied in order to avoid attacks to be successful. This article proposes some ideas in order to be more "proactive" by trying to detect attacks preparation and take measure against the attacker before any attack be launched.

What you will learn...

* some ideas to try to detect profiling phase against your application,
* some counter measures that can be launched against an attacker in order to bother him in his task.

What you should know...

* HTTP request structure.

This article is provided with a Java project in which examples of implementation described into the article are showed.

## Reminder about profiling phase of an attack

Except on TV, attacking a web application (or anything) always start by a phase in which the target is deeply analysed in order to gather as much information as possible about it (web server software, application framework, application software version and type, operating system…). We will name this phase "profiling" and this one takes a big place into the attack timeframe. The profiling can be performed in a "passive" or "active" way. The "passive" way is performed using public information and/or navigating on the target application without doing any suspicious behaviour (browsing like a normal user), the objective here is to not be detected. The "active" way is performed by having behaviour on the target application than can, perhaps, generate alerts depending on the monitoring in place (example: sending HTTP request with an invalid parameter value in order to see how the application behave).

## How can we detect that my application is currently under profiling?

### "Passive" profiling

Using this method, it's difficult to distinguish a normal user than an attacker. An idea is to analyse the number of application functionalities visited by user in a representative time frame. Normal user will only use a part of the application or will use the entire application but on an extended period (one week, one month…) but an attacker will visit as much functionalities as possible in a representative time frame. Take all the web application that you use in your personal/professional life, afterward count how many functionalities do you use and for how long? Interesting track to explore it is not?

Now turn to implementation. The idea is to uniquely identity a HTTP request sender in order to trace is set of requests. There no "silver bullet" method here because IP can be spoofed and HTTP request can be easily forged but using information for these sources we can still catch a panel of attacker. As a skilled and motivated attacker cannot be stopped, the goal here is to raise the skill level required to play with your application.

We will use the information below to identify sender:

* sender IP address,
* HTTP request headers: "Accept", "Accept-Encoding", "Accept-Language", "Connection", "User-Agent".

We will use storage to keep information below:

* digest of the HTTP request sender information above (used as unique ID),
* identifier of the application functionality visited (URI for example),
* last visit date time.

We also need to know the list of functionality exposed by the application in order to perform comparison. This one can be stored into the same storage than information above.

We will assume here that the representative time frame is two weeks.

For each HTTP request, we will store the hit and next check if, for the last two weeks, the visitor has visited all the application functionalities. If it's the case then:

1. We send all current request information (we send information here because in the store we only keep a digest) to a monitoring system in order to generate an alert and launch a review of the sender information in order to decide if aggressive defensive measure should be taken against him,
2. We clean the store with the information of this sender (in order to avoid duplicate alert). Optionally it's possible to move information to archive storage type in order to perform global statistic processing for the application but it's not the goal here.

See class "*com.googlecode.righettod.pdec.PassiveDetectionFilter*" in example project.

### "Active" profiling

When an attacker use this method it's little much easier to detect the profiling behaviour. There several point into the application that can be checked for abnormal behaviour. One of them is the invalid value submitted into application form, indeed we can analyse (during input validation step) the number of invalid values submitted by a user into a functionality (bank transfer form for example). If the count of error is superior to an acceptable limit (three for example) we can assume that the visitor is playing with the application. This method is efficient (I have had the opportunity to test is efficiency through several public web applications) but can it lead to false positive in the case of some lubberly user.

There another more subtle way to detect profiling by literally "phishing the attacker". In this method, we put some "honeypot" into the application. "Honeypot" are represented here into the form of a special custom cookie or custom HTTP header that sounds to be very interesting to the attacker. These cookies/headers are good place because a normal user will not modify theses (they are managed by the browser). An attacker will probably try to modify the value in order to check if the application behaves in different way and it’s at this point that we can detect the profiling.

Now turn to implementation, we will focus here on cookies area. The idea is to find, according to application context and functionalities provided, an interesting name and value for a cookie. We assume here that we want to detect profiling without requiring user to be authenticated and we also assume that there no real application session opened with the user. Like for "Passive" profiling, we will use some information from incoming HTTP request to uniquely identity a HTTP request sender in order to trace is set of requests. Expected values will be hardcoded string in order to not impact application performance.

The table below lists the name and value of the fake cookies that will be issued at first visitor request:

Tableau 1: Name and value of the cookies that will be issued at first visitor request

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Description | Value | Life time |
| verbose\_mode | Simulate a flag that can enable development mode (verbose) of the application. | false | One day |

See class "*com.googlecode.righettod.pdec.ActiveDetectionFilter*" in example project.

## OK we have detected a profiling phase then how my application can defend itself?

There several way to apply counter measures against an attacker in order to bother him in his task and we can classify the counter measure according to their level of invasion on the client.

**Level 1:** (no invasion, fully legal, simply block connection to application)

* close TCP/IP connection,
* backlist IP address.

**Level 2:** (low invasion, not fully legal depending in which data you access, gather information about attacker from is computer)

* run client application through browser plugin (for example Java applet signed by the company, flash animation…) and gather information like geographically location, photo, browser cookies, system environment variables or any others personal information in order to obtain location/identity of the attacker.

**Level 3:** (high invasion, not legal but I think that an attacker will not go to justice against you, apply counterattack measures against the attacker in order to take it offline)

* run client application through browser plugin (for example Java applet signed by the company, flash animation …) and try, executing many random system commands, to break the stability of the operating system or network configuration of the computer of the attacker. The downside here is that it will put the attacker very angry and he can focus on your application after restoring is computer in order to get is revenge but the right side is that it will show to attackers that you are aggressive if someone plays with you!

## Conclusion

In this article we have shown some ideas that can be technically implemented in order to try to detect suspicious user. We have also shown some methods, more or less radical, in order to slow down thereof.

For sure I know that the idea proposed seems to be uncommon, aggressive or a little bit crazy in corporate environment (like a web banking) but I'm personally convinced that if we can detect a profiling phase and send sign, to the originator, indicating clearly to him that "we know what is currently doing" we must be able to stop attack before that it can cause damage…

Only the time and feedback from the projects on which I (and will) work will point to me if I'm right or wrong ;-)

## About the author

Dominique RIGHETTO is a passionate about web application security working for CGI Luxemburg as web application security specialist. He spends his free time learning about web application security and software development technologies in order to increase security in the field of web application.



Web: http://www.righettod.eu - Mail: dominique.righetto@gmail.com