Script-based malware detection in Online Banking Security Overview

Authors:

Jakub Kałużny (jakub.kaluzny@securing.pl)

Mateusz Olejarka (mateusz.olejarka@securing.pl)

SecuRing

e-mail: info@securing.pl

1. ABSTRACT

Online banking applications are particularly exposed to malware attacks. In order to prevent stealing from customer accounts, banks have invested in malware detection mechanisms. These programs are not installed on clients' computers but rather implemented server-side or by including some JavaScript code on protected websites. We have tested such solutions which are using different detection methods. To name a few:

- behavioral patterns,
- web injects signatures,
- user input analysis.

Our research points out clearly that even products sold as a "100% malware proof solutions" have serious implementation errors and it is only a matter of time when malware creators start targeting their guns against these vulnerabilities, effectively bypassing or abusing these countermeasures. Is it a road to failure or is there still time to improve these solutions?

In this document we present security analysis of those solutions from attacker point of view and recommendations for improvement.

2. INTRODUCTION

2.1. Scope of this work

The goal of our research was to perform a security analysis of script-based malware detection and prepare a list of improvements which can be implemented by vendors in order to make these products more effective and secure. Solutions that were taken into consideration contain a client site part (JavaScript code) related to either profiling of user behavior and browser fingerprinting or checking web injects signatures related to malware presence on the users machine. Above mentioned mechanism, combined with a server-side API gathering script output, creates a new type of web fraud detection.

In this paper we do not show exact vulnerabilities we've found in details and all vendors have been notified in a responsible disclosure manner. The aim of this paper is to share our knowledge and improve anti-malware solutions.

If you wish to get additional information please contact us by email. After your positive verification we will able to provide more detailed information and support.

2.2. How does banking malware work?

Our study shows that most of banking malware is sold as "of the self" software. Creators of such software develop malware code and prepare it to target specific bank. Afterwards, usually they obfuscate the executable in order for malware operators not to be able to modify it by themselves. In this context, "targeting" means configuring malware to inject a JavaScript code into HTTP responses from bank's server to user browser. This injection is called "web inject" and is the easiest way to modify browser behavior on a website (for example to steal user's credentials or to switch account number of a transaction). As malware is an executable on user's PC, it has many ways to hook directly into user browser. Operator, after receiving bank's-personalized malware, parameterizes it, specifically choosing i.e.:

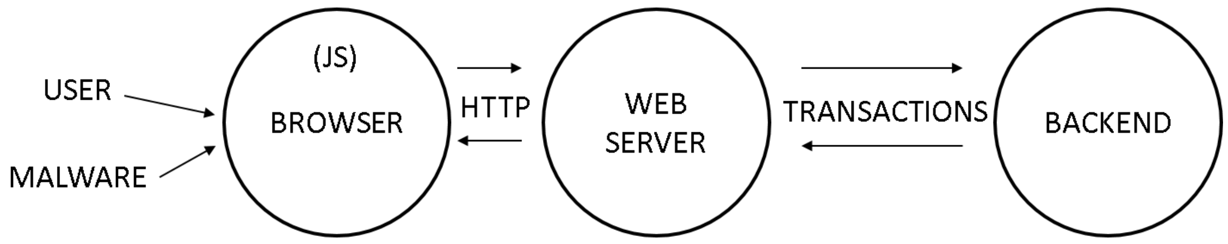
- which accounts will be attacked,
- how much will malware steal,
- where the money will be later sent.

Some of banks use a second channel to authorize transaction. As malware is able to modify transaction details "on-the-fly" and change server responses as well, following authorization methods are ineffective and malware can bypass them:

- TAN codes,
- hardware token generators

- SMS without transaction details

Bear in mind, malware can modify all server responses, so even after sending the transaction, it can modify transaction history on client side so that client doesn't get suspicious. We have observed many malware attacks and we can say that malware will fail only if a secured second channel is a dedicated device which receives transaction key details and the user is aware enough to compare received details with non-editable source (e.g. paper invoice). Why non-editable? We have seen malware which modified account numbers in all digital invoices on PC, so that comparing data from these invoices was worthless. On the diagram below we present a simplified data flow, on which we will base our analysis.



Theoretically, a malware on an infected machine can be totally invisible for the user, by tampering both HTTP requests send by a user's web browser and responses received from the banking application. To be more specific, it can modify images on websites and invoices on hard drive, as well as transfer user's screen to malware operator creating a remote desktop.

However, as we mentioned before, a significant number of malwares use "web injects". It is a piece of HTML or JavaScript code, which is added by malware to the banking website when user opens it. Sometimes there are additional form fields added to the login page or JavaScript code designed to pass user credentials to some external address controlled by malware operator. This changes are completely invisible for the user. The aim is simple – to steal user's credentials or authorization codes for later use or to change on-the-fly transaction details

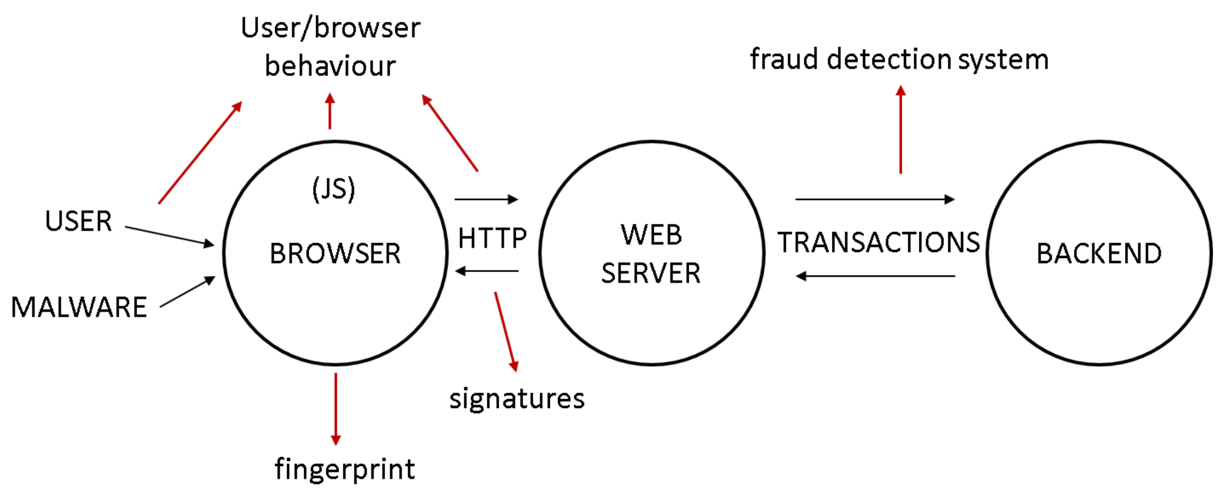
Another way to modify transaction on-the-fly is to tamper computer's memory. For example the Banatrix [1]malware family looks in the memory of the browser for a string which matches a bank account number (26-digits) and changes it to attacker's chosen one.

[1] http://www.cert.pl/news/8999/langswitch_lang/en

2.3. What is script-based malware detection?

As mentioned before, the main concern for preventing malware from stealing money is to detect web injects or changing account numbers in memory. It can be done by adding a special JavaScript code to a bank's website, which will analyze DOM tree client-side, looking for signs of web injects or collect some data about the client environment.

2.4. Observed detection techniques



A simplified diagram above shows observed detection techniques in analyzed solutions. All of them were JavaScript-based and a combination of three input data were used to gather info about possible malware infection:

- browser fingerprint,
- HTTP response data,
- browser behavior.

Later, some of solutions used a fraud detection system to analyze transaction log and look for suspicious entries.

2.4.1. Signatures

In most cases, JavaScript code contains web inject signatures check in a form of a regular expression, which contains a part of web inject itself (for example JavaScript function name which is added as a part of web inject). The code checks whether the string is present in DOM tree (obviously it should not be present in original server response).

We have seen following attributes checked by a signature:

- function name,
- type of JS object,
- JS object name,
- constant string.

When all checks are performed, the result is send to the server. Next, depending of the signature check result, further steps are performed. If the result is negative (no malware found), the user is allowed to proceed. Otherwise (a positive result, malware found), the system generates an alert and additional actions are performed. For example user is alerted using an out-of-bound channel (telephone, sms message) and his session in the application is terminated.

2.4.2. Browser fingerprint

Second observed method of detecting malware is to fingerprint the browser looking for the uniqueness of its configuration and the presence of suspicious settings or add-ons.

Output from the JavaScript code is later send to the server for analysis and for the anti-fraud engine to decide whether given user behavior is suspicious.

2.4.3. User/browser behavior

Humans use browser differently than bots and this can be measured as well. Speed of moving the mouse, clicking buttons and typing in forms can be a good attribute to tell a human and bots apart. We have seen aggregated data about user/browser behavior being sent before login action.

2.4.4. Fraud Detection System

Most advanced method is to build a fraud detection systems which can use previously mentioned methods output and also take into account data collected by other banking system (user IP, location, average transfer amount).

2.5. Limits

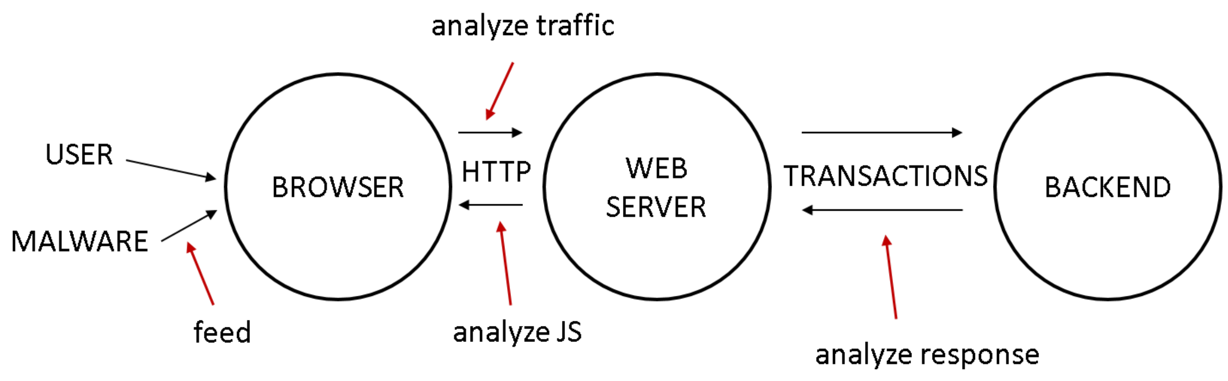
Each of these methods has it pros and cons. None of them will detect each and every malware. However, we believe that every malware countermeasure raises the bar for malware creators and operators.

All of these methods are very dynamic and malware signatures can be changed very quickly. Nonetheless, behavioral systems are still based on the old HTTP – HTML – JS stack and can therefore contain architecture and implementation errors.

3. ANALYSIS OF POTENTIAL ATTACK VECTORS

3.1. Our approach

We fed tested solutions with malware samples, looking for detection mechanisms and interesting requests, we analyzed HTTP communication, obfuscated JavaScript and in some cases the information available in backend part of the anti-malware solution.

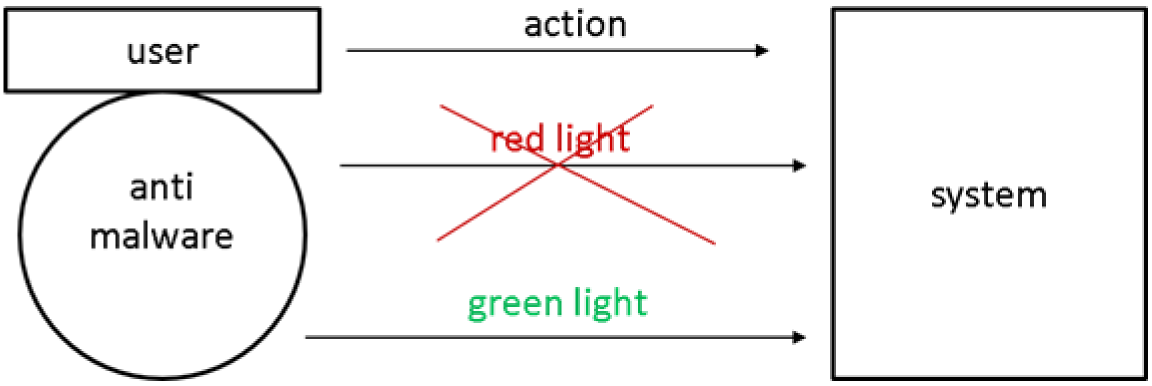


3.2. Potential attack vectors

3.2.1. Red light vs green light

Our first idea was to disable an anti-malware solution because of a simple architecture problem. Most of analyzed solutions used a “red light” system, so they were sending an alert only after positive malware identification. Thus, a malware which wants to disarm protecting solutions does not need to hide or analyze and tamper solution’s output, but just turn it off.

We advise that a “green light” should be used instead, so that a simple reverse inference could work: no signal – no go. These data should be taken into consideration by bank’s backend system in order to determine whether a “no green-light” signal means there is a malware on user’s PC or whether it just proofs that something went wrong, and ask user i.e. to restart his browser.

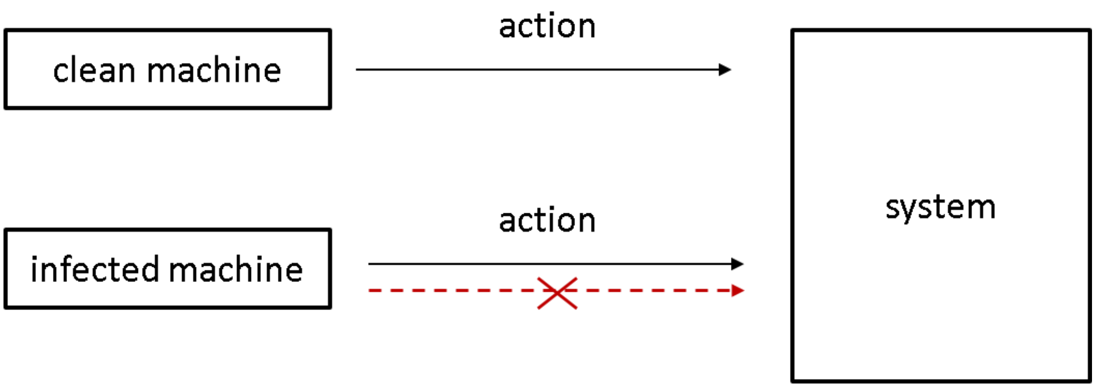


3.2.2. Try cutting off the traffic

Next scenario we considered was to cut communication to particular URL address (related either to initialization phases or to reporting phase). We were really amazed by the results.

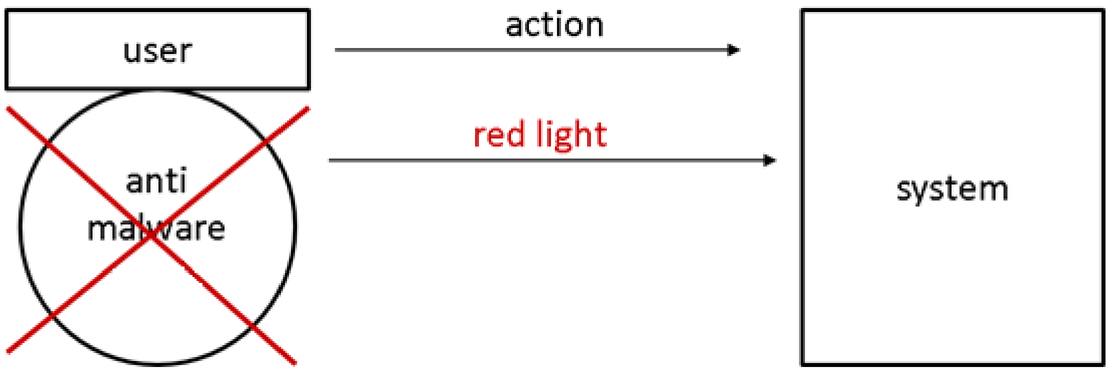
In most of the cases it was enough to disable an anti-malware solution. Simply drop a HTTP request to a given URL or redirect it into nowhere. Both at initialization step of the solution and in the reporting step as well.

First failure could be connected with possible integration error (anti-malware didn't get info from the web application that new user session appeared from the web application), although second one should definitely be reported (new user session, no signature check result send for that session) after preconfigured timeout.



3.2.3. By too fast login

This case was pretty interesting. A very fast login (but still achievable by man) on an infected machine will not result an alert raised. This happened because anti-malware JavaScript was not fully loaded and it didn't execute. Because the solution didn't predict such situation, malware wasn't detected.



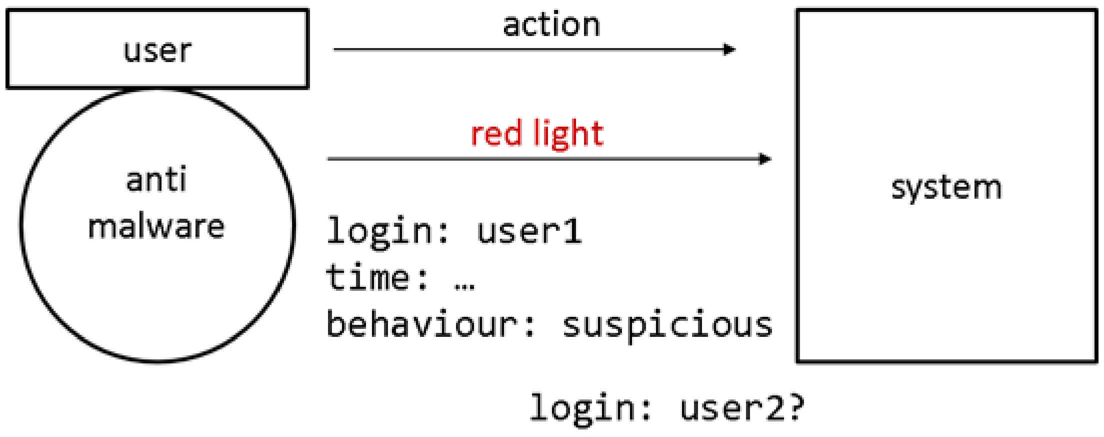
3.2.4. JavaScript deobfuscation

On the client side it is a struggle between web injects and anti-malware scripts. The signature check is done on the clients side, so therefore signatures itself must be known on the client side as well. We believe that the signatures and code responsible for the verification and sending back the result should be obfuscated to make the understanding of the mechanisms harder for an attacker.

Here arises the question, if we are able to do it bullet-proof in JavaScript. The answer is no. Due to its nature, each client site JavaScript based solution given enough resourced can be analyzed and possibly bypassed. Therefore it should be well concealed in the web application, and not so obvious to find/isolate from other JavaScript code. The obfuscation algorithm should use a pattern of different obfuscation techniques, and to change this pattern in each time it is initialized (the JavaScript code is returned to the client side).

3.3. By tampering alerts

Apart from bypassing solution itself, one interesting case let us to abuse the whole reasoning backend system as well. The case was as follows: after spotting the malware, the anti-malware JavaScript script sends alert to backend system. It contains login name, date and few other details encrypted with RSA public key hardcoded in JavaScript. Tampering these values let us to abuse the system, raise false alerts and create a decoy for other attacks.


user
action
red light
anti malware
system
login: user1
time: …
behaviour: suspicious
login: user2?

4. RECOMMENDATION FOR IMPROVEMENT

In this chapter we present our thoughts of improving existing anti-malware solution based on the possible attack scenarios we've discussed earlier in paragraph 3.2.

4.1. Architecture

4.1.1. Initialization phase

Anti-malware solution should be integrated in the application in a way that the application sends notification the moment new user session is created (even before user authenticates). This should allow the anti-malware solutions to detect the situation, when there is a new user session, but fraud detection JavaScript module was not initialized. This should generate "yellow light and warning" in backed anti-fraud systems for given user session identifier.

4.1.2. Reporting phase

This is similar to one described above. The anti-malware solution is initialized on the client side, but the HTTP request reporting results isn't send. After preconfigured timeout this situation should generate even more important warning for given user session in backend anti-fraud systems (mechanism initialized, but no results send back).

We believe that anti-malware solution should always report results (green light), not only when something unusual was found on the client's side (red light).

Another issue is how to treat report which arrives from the application. Even if there is encryption involved, it is done on the client side, hence the public key used to encrypt given message is known on the client side as well (it hast to be placed inside the JavaScript code). Based on this such message can be send by an attacker as well. All additional fields send from the user browser (except the results) should be treated as untrusted, for example username which is already known on the server side, based on the users session.

The report shouldn't be the only factor, when deciding if a particular user session originates from infected machine. It should be considered together with all other data collected about this user session (for example if originating IP address matches previous ones used for given user login, whether user uses some language, e.t.c.).

4.2. Obfuscation and concealment

4.2.1. Obfuscation

In case of client side JavaScript designed to detect and report malware presence on the users machine the obfuscation is a must. The logic and the signatures embedded in the JavaScript code should be obfuscated to make it hard and a resource consuming task for an attacker to understand and mitigate. And based on our experience this is usually a case.

There is no perfect solution here, because of two things:

- is it still a fight between JavaScript and web injects and the battlefield is the user browser, both sides have equal rights there
- the JavaScript code is known on the client side, so sooner or later someone will notice it and start to wonder

Therefore we recommend to use an obfuscation algorithm, which:

- uses many different JavaScript obfuscation techniques to obfuscate the code
- changes the selected obfuscation techniques pattern each time the code is sent back to the browser

Simple obfuscation using one technique may be good for starters, but be prepared to raise the bar higher in the future.

4.2.2. Concealment

Another thing to take into consideration is to hide the client side part of the anti-malware solution in the application. Try to make it look like something else entirely. Make it hard to isolate it from the other parts/scripts of the protected application.

The obfuscated code can attract attention, when the rest of the JavaScript code in the application is not even compressed, but when it is put inside some legitimate compressed script it can decrease the chance the attacker will find it quickly.

4.3. Updates and patches

In this kind of software it is crucial to have the ability to deploy updates and patches quickly. When the customer is under a malware attack and the anti-malware is not stopping it, the time of your reaction is very important. Automate all elements of deployment of the new version, that can possible be automated. Automate the way in which the patch is transferred to the client as well.

4.4. Service level agreement

One of the most important thing, when utilizing such a solution, it's not a malware detection rate which matters most but how quick an anti-malware solution can adapt itself to an ongoing attack. Especially how fast new signatures stopping active, running malware attack can be deployed?

Keep it in mind when thinking about the service level agreement between you and a vendor.

4.5. Researcher cooperation

Last but not least. Setup a way for the web security researchers to contact you and answer their mails. Most of us just want to have a little safer online world. Do not be afraid to cooperate!

5. SUMMARY

5.1. Before you buy

If you think about using such solution there are few things to consider. Start with requesting detailed technical information. Marketing material won't say much about underlying architecture and concepts. Ask for technical people to do the presentation and be prepared to ask them technical questions.

If afterwards you are still interested, then request a live demo on an application you wish to protect. This will cover two things:

- You will see if it is difficult to integrate particular solution with your application and infrastructure.
- You will see how the output from the anti-malware software looks like and how it could be added to your security monitoring routine. Try to understand pros and cons of each solution, advantages and limits.

5.2. Bigger picture

Anti-malware solutions cannot be considered as only precaution against malware. It is just another layer of defense, which helps your security team in keeping you and your users safe. Have a strategy, gather your team of specialist, gather knowledgebase, keep track on the news and keep an eye on the dark side. Anti-malware should be well integrated into your security monitoring and anti-fraud systems, collect stats about the user’s behavior, calculate risk connected with each user session and transaction. Be adaptive.

5.3. In-house solution

Creating such solution as in house project sounds reasonable as well. The potential attackers would probably start trying to bypass the anti-malware solution with biggest adoption on the market first.

5.4. Trust but verify

Last but not least it is important to remember, that it is another piece of software added to your existing application. As such it can contain security vulnerabilities, either architectural or implementation errors. So either prepare test scenarios and execute them against your test environment or hire properly skilled consultant to do that for you.

5.5. Contact

Jakub Kałużny
e-mail: jakub.kaluzny@securing.pl

Mateusz Olejarka
e-mail: mateusz.olejarka@securing.pl)

http://www.securing.pl
e-mail: info@securing.pl
Jontkowa Górka 14a
30-224 Kraków
tel./fax.: (12) 425 25 75