Man-in-the-browser Attack: A Case Study on Malicious Browser Extensions

Sampsa Rauti(✉)

University of Turku, 20014 Turku, Finland
sjprau@utu.fi

Abstract. Man-in-the-browser (MitB) attacks, often implemented as malicious browser extensions, have the ability to alter the structure and contents of web pages, and stealthily change the data given by the user before it is sent to the server. This is done without the user or the online service (the server) noticing anything suspicious. In this study, we present a case study on the man-in-the-browser attack. Our proof-of-concept implementation demonstrates how easily this attack can be implemented as a malicious browser extension. The implementation is a UI-level, cross-browser implementation using JavaScript. We also successfully test the extension in a real online bank. By demonstrating a practical man-in-the-browser attack, our research highlights the need to better monitor and control malicious browser extensions.

1 Introduction

In a man-in-the-browser (MitB) attack, a malicious program can change the structure and contents of web pages, modify data in HTTP messages, or steal sensitive data the user enters in the browser without the user or online service observing anything out of the ordinary [15]. There are several real-world examples of man-in-the-browser malware, such as SpyEye, Zeus, Torpig, URLZone and Silentbanker [4,5].

The attack was originally presented by Augusto Paes de Barros in a talk about new backdoor trends in 2005. The name man-in-the-browser attack was later invented by Philipp Gühring, who also described the attack in more detail and discussed possible countermeasures against it [9]. Today, almost 15 years later, pieces of malware with man-in-the-browser functionality are still a significant threat for many online services. Online banking and web services of financial institutions, for example, are among the most popular targets for man-in-the-browser attacks [6].

This study presents a case study on the man-in-the-browser attack. We demonstrate how easy it is to build a malicious browser extension with man-in-the-browser functionality that stealthily changes the data the user has inputted in the browser. While our implementation is a Chrome extension, it could easily be utilized in Opera or Firefox as well, as the code is written in JavaScript and operates on the UI level. We also successfully test this extension in a real

S. M. Thampi et al. (Eds.): SSCC 2019, CCIS 1208, pp. 60–71, 2020.
https://doi.org/10.1007/978-981-15-4825-3_5

online bank. This study shows that even a simple, easy-to-implement malware can successfully perform a man-in-the-browser attack, bypassing all traditional authentication mechanisms and other security solutions like TLS encryption. By demonstrating a practical man-in-the-browser attack, our research shows that MitB is still a serious threat for web applications and outlines the need to better monitor and control the malicious browser extensions.

The rest of the paper is organized as follows. Section 2 explains how a typical man-in-the-browser attack proceeds. Section 3 describes our proof-of-concept implementation for the attack. Section 4 describes the experiment we performed in a real online bank with our malicious extension. Section 5 discusses the implications and countermeasures of man-in-the-browser attacks. Finally, Sect. 6 concludes the paper.

2 The Attack

Because spying on and altering messages in the network is difficult due to encryption, many attackers are instead looking for an easier opportunity to perform man-in-the-middle attack at the endpoint of communication – the user's infected machine. Man-in-the-browser is a security threat that can be described as a deceitful proxy inside the browser. The goal of the malicious program is either to steal or alter the data exchanged by the user and the web service [20]. This can mean (1) fraudulently altering the contents of web pages before they are rendered (2) modifying the data in incoming or outgoing messages (3) generating additional malicious HTTP requests, or (4) capturing sensitive data and sending it to command and control server [9,22]. A MitB malware can contain some or many of these functionalities. The malicious program usually operates totally silently, without giving the user or the web service any visible clues about its existence.

In this paper, we will take a closer look at a type of man-in-the-browser attack that uses DOM (Document Object Model) modification to quietly alter the data inputted by the user before the data gets transmitted to the server. Such an attack usually proceeds as follows:

1. The user's computer gets infected by malware. Oftentimes, the malware resides in the browser and is implemented as a malicious browser extension.
2. The malware has a list of matching URLs and once the user visits a URL on the list, the man-in-the-browser functionality activates.
3. The malicious program waits until the user logs in and makes a transaction – for instance, the user transfers money from his or her bank account.
4. Before the data is sent to the server, the malware tampers with the request and modifies the data – for example by using the browser's DOM interface to change the bank account number of the receiver.
5. After the values submitted by the user have been modified, the man-in-the-browser malware lets the browser proceed with transmitting the data to the server.

6. The browser then delivers the deceptive HTTP request to the server. The server, however, has no way of telling this falsified request from a real one. It therefore accepts the request, believing this is the real intent of the user.
7. The user is then usually asked to verify the transaction. For instance, an online banking website shows the details of a bank transfer to the user once more so that they can be confirmed.
8. The MitB malware changes any details (e.g. the bank account number) on the displayed page so that they correspond to the original transaction that the user intended to make. The user thinks everything is fine and confirms the transaction.

The user and the online service involved in the exchange have been deceived. Later the user will probably notice that the transaction was altered (e.g. when receiving a reminder letter for the invoice). At this point, the money has probably already been irrevocably lost.

3 Implementation

We studied how a man-in-the-browser attack could be implemented as an extension for the Chrome web browser. Like in the example description of a man-in-the-browser attack in the previous section, we decided to make a malicious browser extension that manipulates the data the user has filled in on a web site before it is sent to the server. Later on, we perform an experiment with our extension by changing the recipient account number when making a transaction in an online bank.

For better security, changing HTTP messages has been made tricky in Chrome by restricting this functionality in the WebRequest API. We circumvent this problem by using an easier method of manipulating the data with the DOM API before it is sent to the server. This way, we do not even need to use any browser specific APIs, the extension will just consist of a few lines of very basic JavaScript.

When using DOM to replace the data given by the user with our own fraudulent data, we have to find a way to do this stealthily so that the user does not notice anything. Simply changing the value of a text field so that the user can easily spot the change does not work, for example. There are many possible approaches to modify the DOM and manipulate data, but in our implementation we used the following one:

1. Find the text field containing the value we want to change.
2. Make a fake copy of this original text field.
3. Make the original field invisible (with CSS).
4. Replace the value of the invisible original text field with a deceptive value.
5. Insert to fake field in the place of the original one.

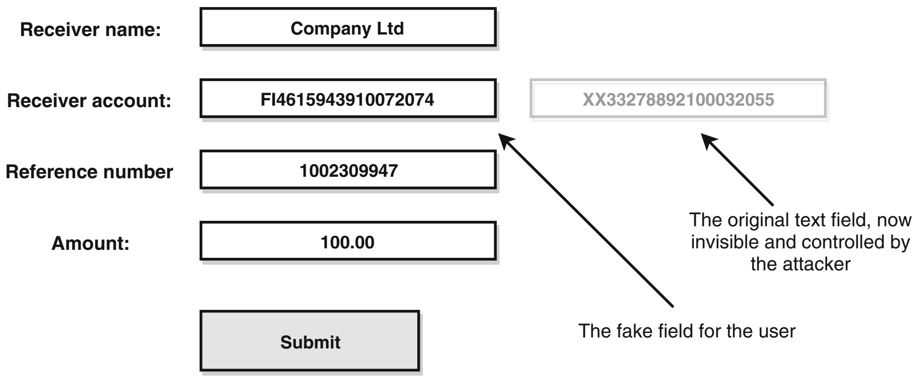


Fig. 1. A man-in-the-browser attack against the online bank application by substituting the receiver account field with a fake one. The user types a value in the fake field, which is never going to be submitted. Instead, the value in the original, invisible text field, controlled by the attacker, is transmitted to the server.

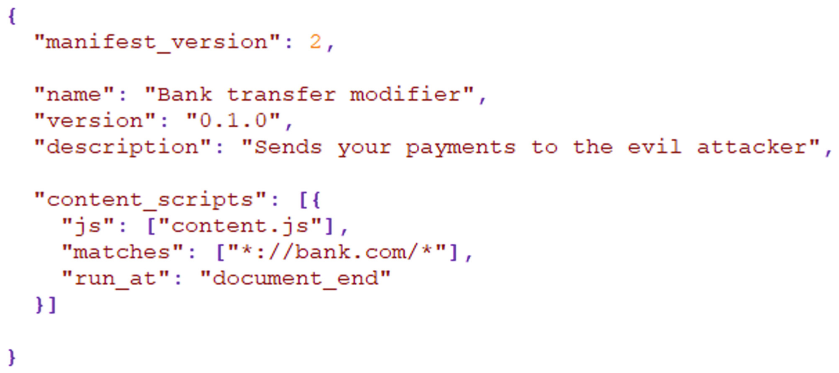
Now, the user will think that the fake field is the real one, and he or she will use this field to input the value in the application. However, what is really going to be sent to the server is the data in the original text field that is invisible and unreachable to the user. Figure 1 illustrates this situation. Of course, the same trick could be used with other text fields as well. For instance, the attacker could easily increase the amount of money that is being transferred.

The whole malicious modification functionality in the extension can be written in about 5 lines of basic JavaScript which uses the DOM API. We will not share the code here, because it is a piece of malware, but anyone with a moderate knowledge of JavaScript could write the extension in just a few minutes. The code is available upon request for research purposes.

There is one more minor thing to take care of: we want the malicious extension to also deceive the user when the verification page is displayed. On the verification page, the extension just searches the element that displays the data which has been sent to the server (e.g. payment information with an account number) and replaces its contents with the original value the user has inputted. This requires just a few lines of code: capture the data written by the user in the fake text field, store it and display it on the verification page instead of the fallacious data that has really been sent to the server. With this, our extension is pretty much finished.

The extension should normally only be installed through Chrome Web Store, but one can also test unpacked extensions by enabling Chrome's developer mode. To load the extension in Chrome, we just have to have two separate files in a folder: a manifest (manifest.json) and a content script (content.js) containing the malicious functionality that was described previously. The contents of the manifest file are shown in Fig. 2.

The main thing to note about the manifest file is the fact that it defines the website or the websites on which the extension activates (matches). The content script, content.js, is injected into the web page once the DOM of the page is complete [7]. Naturally, the name and description of the extension displayed to the user on the extension page of the browser would be changed if we really were building a real malware. Malware extensions are usually Trojans: they trick the user by performing some useful functionality, but at the same time, malicious activities are stealthily performed in the background.



```
{
  "manifest_version": 2,

  "name": "Bank transfer modifier",
  "version": "0.1.0",
  "description": "Sends your payments to the evil attacker",

  "content_scripts": [{
    "js": ["content.js"],
    "matches": ["*://bank.com/*"],
    "run_at": "document_end"
  }]

}
```

Fig. 2. The extension's manifest file.

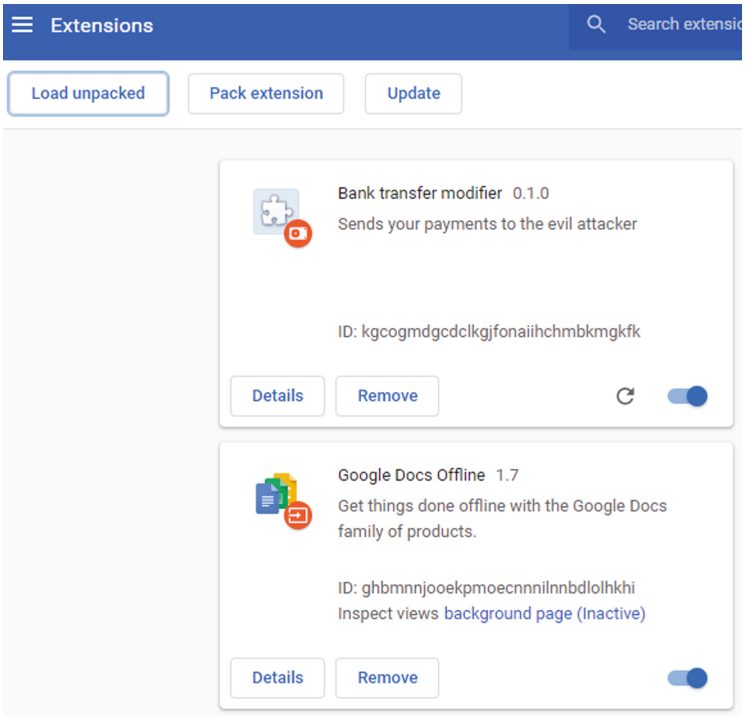
4 Experiment

We used Chrome's developer mode and installed our extension. The extension was tested on a machine with Windows 10 and Chrome version 76.0.3809.132 installed. Figure 3 shows the extension on Chrome's extension page.

We proceeded to test our extension on a real online bank service. We will leave out the name of the bank from this study, suffice it to say it was a relatively large European bank. The experiment was a success, as our extension was able to divert the payment to a different bank account than given in the bank application.

Regarding the experiment, the following observations are especially noteworthy:

1. *Two-factor authentication is useless.* The bank uses two-factor authentication in the login process. This is useless against man-in-the-browser attacks that bypass the authentication phase and modify the transaction "on the fly" as the user makes the payment. Therefore, our extension did not experience any challenges in the login phase.



Bank transfer modifier 0.1.0
Sends your payments to the evil attacker

ID: kgcogmdgcdclkgjfonaiihchmbkmgkfk

Google Docs Offline 1.7
Get things done offline with the Google Docs family of products.

ID: ghbmnnjooekpmoecnnnilnnbdlolhkhi
Inspect views background page (Inactive)

Fig. 3. Chrome's extension page.

2. *Out-of-band verification is useless to some extent.* The bank uses out-of-band (OOB) verification to confirm the transaction. OOB verification verifies the transaction using a second channel other than just the web browser [5,24]. This can be done with a mobile application (where the user gives a PIN code) or using a separate little device supplied by the bank. If the user uses the separate device, he or she gives the device a code on the bank's web page, and then gives the bank web page a code calculated by the device. However, the device never displays the receiver's account number to the user, and in this sense, the verification is not really complete and does not really protect against our MitB attack.

   The other way of verification, the mobile application, is better, because the receiver account number is displayed to the user for verification. Of course, this can potentially stop a MitB attack and our extension, if the user notices the difference in account number is different from what he or she originally inputted in the browser. One problem in mobile verification is the fact that nowadays, many users can use their mobile phones for online banking. In this case, OOB verification may be rendered useless if the phone is infected, because there is no real second channel anymore in the verification process [3,10]. In addition, our most prominent concern with OOB verification is related to user errors and simple psychology. After a while, the verification process most likely becomes an automatic routine for the user. Is the user really going to carefully check the receiver account number every time? We believe a high percentage of the users will probably not do this. Instead, the users simply automatically give the mobile app the PIN code to verify the transaction as a part of a routine.

3. *TLS encryption is useless.* TLS encryption is a good measure for protecting against man-in-the-middle attacks in general. However, when the modification attack happens inside the browser, the data can easily be modified before it is encrypted. Therefore our extension and MitB attacks in general bypass TLS encryption. Too many banks and other critical online services today still state in their security instructions that the user will be safe when he or she sees the lock indicating a secure connection in the address bar. This can lull users into a false sense of security.
4. *Many anti-virus programs are currently useless.* Sadly, anti-virus vendors have not really been interested in what happens inside web browsers, which leads to low detection rates for malicious browser extensions [2]. Anti-virus programs consider browsers safe, and therefore they often also consider browser extensions harmless without any stricter scrutiny. As web applications become more popular, replacing many desktop applications, and the browser becomes a new platform for running many application and extensions, malicious activity inside the browser should be more closely monitored. The computer we performed our experiment on also had an anti-virus program installed. Unsurprisingly, the program did not react to our man-in-the-browser attack in any way.
5. *The bank did not question the transaction.* Finally, the transaction we tested was a success and the online bank did not notice anything suspicious was going on. It is not completely fair to criticize the bank about this, because we transferred a relatively small amount inside the same country (from a Finnish account to a Finnish account). Still, we want to make this observation here to remind that banks should check all transfers on the server side and require extra verification (for example by calling the customer and asking for verification) for payments that differ from the normal pattern of transactions. Also, banks could include some client side security measures in their web applications to mitigate MitB attacks, as we will see in the next section.

5 Discussion and Countermeasures

The proof-of-concept implementation for a malicious MitB extension presented in this study shows that in 2020, about 15 years after their appearance, man-in-the-browser attacks are still a significant threat and can effectively work against the modern online banking web systems which are supposed to be at the top of their game in terms of securing transactions. As already noted by Blom a couple of years earlier [1], it still seems that many banks do not consider man-in-the-browser attacks a serious threat.

As noted before, a malicious extension is regrettably easy to implement. Writing a few lines of rudimentary JavaScript and using the DOM interface is not difficult. However, to create an extension that changes data entered in a form, not even this is actually required. This is because the code of extension could be shared to less technically oriented attackers, who would then only need to fill in two details in the code: (1) the ID of the text field which we want to fabricate (or IDs for several text fields, if required), and (2) the ID of the corresponding element on the verification page so that it can be edited as well. Actually, the latter ID is not strictly necessary, because the extension could just scan the verification page and replace the value regardless where it is. Even more dangerously, why not make the extension the look for IBAN account numbers (or any other well-formatted data) in the text fields and replace all such fields? Then the extension would be completely automatic and probably work against several banks even without prior knowledge about the exact user interfaces of the banking web apps. At any rate, it should be clear that even when not automatized to this extent, our extension is really easy to parametrize. Anyone can search for IDs of HTML elements (e.g. using Chrome's inspect functionality) and then make the necessary replacements in the JavaScript code.

Google has continuously striven to make the process of reviewing extensions more rigorous [16], and in 2018, installation from web sites other than Chrome Web Store was disabled. However, many malicious authors have still succeeded in slipping their extensions into Chrome Web Store. With over 60 % market share, Chrome is still a very attractive choice for malware developers. Many malware authors also first publish a completely harmless extension and then integrate malicious functionality to the extension later. The adversary can also use malware that circumvent Chrome's installation restrictions and programmatically install the extension to Chrome without the user's knowledge and permission. For example, the notorious "Catch-All" extension for Chrome that stole all data user typed in the browser used a malicious installer program that started Chrome from command line with parameters that allowed the installation of the extension and circumventing many security features related to extensions [12]. Finally, the adversary could employ social engineering to get the user to install the harmful extension in developer mode.

Although Chrome's extension policy has become stricter in recent years, many other browsers, other browsers such as Firefox and Opera have looser policies when it comes to extension installation and permissions. It is also important to note that the JavaScript code we wrote does not use any browser specific features, and it could be directly used for Firefox and Opera extensions as well.

It is quite apparent additional countermeasures are needed against malware with man-in-the-browser functionality modifying the user's transactions. Scientific literature has proposed numerous different countermeasures over the years, but we will discuss just a few solutions in the context of our practical experiment here:

– *Stricter permission control for browser extensions.* Chrome has a system in place that makes the users confirm the permissions an extension can have.

However, many users are probably going to accept these permissions without really reviewing them or understanding what they mean. Firefox and Opera, on the other hand, do not have this fine-grained extension permission management. Therefore, new ideas and frameworks for permission and access control management and monitoring [8,13,23] are needed. For example, Liu et al. propose assigning different sensitivity levels for HTML elements [11]. It could be a good idea to restrict the ability of extensions to modify text fields, for example. At the very least, certain patterns such as an extension modifying an invisible text field (like in our example implementation) are highly suspicious. Also, it would not be that difficult to compile a list of the most critical web sites (such as online banks) where extensions would be completely turned off.

– *Out-of-band verification.* We already saw that out-of-the-band verification has its downsides. The process can become a boring routine for the user or both web banking and verification can be done on the same infected mobile phone. However, out-of-band verification is still a good security mechanism when used correctly. An uninfected second channel has to be used for verification and the transaction details have to be shown to the user. The user has to understand why verification is important and check the transaction carefully. Aside from a mobile device which may not be completely secure, for instance a separate USB gadget with a display can be used for verification [14,18].
– *Monitoring web page integrity.* One way to protect against DOM-based man-in-the-browser attacks is to verify the integrity of the web pages [17]. The challenge here, of course, is that there are many legitimate extensions such as advertisement blockers that need to modify pages. On some web sites with critical functionality and sensitive information, however, this countermeasure could provide great benefits. Cryptography can be used to protect the integrity of web content [21]. As a mechanism to mitigate man-in-the-browser attacks, critical applications could add functionality guarding the integrity of the web page. An even more secure solution would be to integrate this check in the browser. This way, performing tricks such as adding extra text fields would become more difficult.
– *Hardening the browser.* Hardening refers to securing software by limiting the attack surface and implementing other mechanisms preventing cyber attacks. For example, a clean web browser can be loaded from an external tamper-proof device [19]. The hardened browser would use TLS to encrypt communication with the server and browser extensions would not be allowed. Therefore, setting up a man-in-the-browser attack would become difficult for the adversary. However, the usability of this solution is not as good as that of a normal browser, as the user has to attach the device and use a separate browser for critical transactions.

To summarize, thwarting man-in-the-browser attacks is a co-operative effort involving many parties. First, web browser vendors need to make sure permissions of extensions are controlled and users are informed about possible implications of granting these permissions. Installing malicious extensions should not be too easy. Intuitive mechanisms for turning off extensions on certain web sites should be provided. Second, providers of critical services such as banks should always provide appropriate out-of-band verification and emphasize the importance of carefully checking the transactions. Client-side mechanism such as DOM integrity checking can be used on client side. Third, anti-virus vendors should do even better job in analyzing what happens inside the browser (e.g. by analyzing activities of extensions and monitoring what kind of resources they access). Fourth, organizations need to pay attention to their policies on browser extensions. It would be a good idea to regularly review the installed extensions. Last but not least, it is important for the users to understand how powerful browser extensions are and select the extensions they use carefully. Many attacks could be proactively prevented by educating users.

Finally, although we have been mainly discussing online banks in our examples, it is worth noting that man-in-the-browser attacks are a threat to a wide variety of different web services. One can easily imagine replacing the content sent by the user in social media or webmail services with messages decided by the adversary. Tampering with online voting, input data for medical appliances, or industrial processes could potentially have even more serious consequences.

6 Conclusion

We have presented a case study on man-in-the-browser attacks and demonstrated how a practical attack can be carried out by building a malicious browser extension. It is concerning how simple the malicious code is and how effortlessly the attack can be deployed against users even 15 years after man-in-the-browser attacks were first discovered. While no security solution completely prevents man-in-the-browser attacks (and still preserves good usability), combining several countermeasures and enforcing these security approaches more effectively in modern web browsers and web applications should significantly alleviate the problem in the future. This goal can be reached with co-operative efforts of web developers, users, antivirus program vendors and browser manufacturers.

References

1. Blom, A., de Koning Gans, G., Poll, E., de Ruiter, J., Verdult, R.: Designed to fail: a USB-connected reader for online banking. In: Jøsang, A., Carlsson, B. (eds.) NordSec 2012. LNCS, vol. 7617, pp. 1–16. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-34210-3_1
2. DeKoven, L.F., Savage, S., Voelker, G.M., Leontiadis, N.: Malicious browser extensions at scale: bridging the observability gap between web site and browser. In: 10th USENIX Workshop on Cyber Security Experimentation and Test (CSET 2017). USENIX Association, Vancouver, BC (2017), https://www.usenix.org/conference/cset17/workshop-program/presentation/dekoven
3. Dmitrienko, A., Liebchen, C., Rossow, C., Sadeghi, A.R.: On the (in)security of mobile two-factor authentication. In: Christin, N., Safavi-Naini, R. (eds.) Financial Cryptography and Data Security, pp. 365–383. Springer, Berlin Heidelberg (2014). https://doi.org/10.1007/978-3-662-45472-5_24

4. Dougan, T., Curran, K.: Man in the browser attacks. Int. J. Ambient Comput. Intell. (IJACI) **4**(1), 29–39 (2012)
5. Entrust: Defeating Man-in-the-Browser Malware - How to prevent the latest malware attacks against consumer and corporate banking. White paper (2014)
6. Gezer, A., Warner, G., Wilson, C., Shrestha, P.: A flow-based approach for trickbot banking trojan detection. Comput. Secur. **84**, 179–192 (2019)
7. Google: Content scripts (2019). https://developer.chrome.com/extensions/content_scripts
8. Guha, A., Fredrikson, M., Livshits, B., Swamy, N.: Verified security for browser extensions. In: 2011 IEEE Symposium on Security and Privacy, pp. 115–130. IEEE (2011)
9. Gühring, P.: Concepts against man-in-the-browser attacks. Technical report (2006)
10. Konoth, R.K., van der Veen, V., Bos, H.: How anywhere computing just killed your phone-based two-factor authentication. In: Grossklags, J., Preneel, B. (eds.) FC 2016. LNCS, vol. 9603, pp. 405–421. Springer, Heidelberg (2017). https://doi.org/10.1007/978-3-662-54970-4_24
11. Liu, L., Zhang, X., Yan, G., Chen, S., et al.: Chrome extensions: threat analysis and countermeasures. In: NDSS (2012)
12. Marinho, R.: "Catch-All" Google Chrome Malicious Extension Steals All Posted Data (2017). https://morphuslabs.com/catch-all-google-chrome-malicious-extension-steals-all-posted-data-f2472e272101
13. Marouf, S., Shehab, M.: Towards improving browser extension permission management and user awareness. In: 8th International Conference on Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom), pp. 695–702. IEEE (2012)
14. Migdal, D., Johansen, C., Jøsang, A.: DEMO: OffPAD - offline personal authenticating device with applications in hospitals and e-banking. In: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security CCS 2016, pp. 1847–1849. ACM, New York, NY, USA (2016)
15. OWASP: Man-in-the-browser attack (2019). https://www.owasp.org/index.php/Man-in-the-browser_attack
16. Protalinski, E.: Google updates Chrome Web Store review process and sets new extension code requirements (2018). https://venturebeat.com/2018/06/12/google-disables-inline-installation-for-chrome-extensions/
17. Rauti, S., Leppänen, V.: Man-in-the-browser attacks in modern web browsers. In: Emerging Trends in ICT Security, pp. 469–480. Elsevier (2014)
18. Rautila, M., Suomalainen, J.: Secure inspection of web transactions. Int. J. Internet Technol. Secur. Trans. **4**(4), 253–271 (2012)
19. Ronchi, C., Zakhidov, S.: Hardened client platforms for secure internet banking. In: Pohlmann, N., Reimer, H., Schneider, W. (eds.) ISSE 2008 Securing Electronic Business Processes. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-8348-9283-6_39
20. Ståhlberg, M.: The trojan money spinner. In: Virus Bulletin Conference, vol. 4 (2007)
21. Toreini, E., Shahandashti, S.F., Mehrnezhad, M., Hao, F.: Domtegrity: ensuring web page integrity against malicious browser extensions. Int. J. Inf. Secur. 1–14 (2019)
22. Utakrit, N.: Review of browser extensions, a man-in-the-browser phishing techniques targeting bank customers (2009)

23. Wang, L., Xiang, J., Jing, J., Zhang, L.: Towards fine-grained access control on browser extensions. In: Ryan, M.D., Smyth, B., Wang, G. (eds.) ISPEC 2012. LNCS, vol. 7232, pp. 158–169. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-29101-2_11
24. Zhang, P., He, Y., Chow, K.: Fraud track on secure electronic check system. Int. J. Digit. Crime Forensics **10**(2), 137–144 (2018)