Department of Computer Science

Security Failures in Modern Software

Thanh Bui

Aalto University

DOCTORAL
DISSERTATIONS

Aalto University publication series
DOCTORAL DISSERTATIONS 46/2021

Security Failures in Modern Software

Thanh Bui

A doctoral dissertation completed for the degree of Doctor of Science (Technology) to be defended, with the permission of the Aalto University School of Science, on 27th April 2021 at 12.

Aalto University
School of Science
Department of Computer Science

Supervising professor
Professor Tuomas Aura, Aalto University, Finland

Thesis advisor
Dr. Markku Antikainen, Aalto University, Finland

Preliminary examiners
Dr.-Ing. Ben Stock, CISPA Helmholtz Center for Information Security, Germany
Professor Davide Balzarotti, Eurecom, France

Opponent
Associate Professor Robert Lagerström, KTH Royal Institute of Technology, Sweden

Aalto University publication series
DOCTORAL DISSERTATIONS 46/2021

ISBN 978-952-64-0326-7 (printed)
ISBN 978-952-64-0327-4 (pdf)
ISSN 1799-4934 (printed)
ISSN 1799-4942 (pdf)
http://urn.fi/URN:ISBN:978-952-64-0327-4

Unigrafia Oy
Helsinki 2021

Finland

Printed matter
4041-0619

Aalto University

Abstract

Aalto University, P.O. Box 11000, FI-00076 Aalto www.aalto.fi

Author
Thanh Bui

Name of the doctoral dissertation
Security Failures in Modern Software

Publisher School of Science

Unit Department of Computer Science

Series Aalto University publication series DOCTORAL DISSERTATIONS 46/2021

Field of research Computer Science

Date of the defence 27 April 2021

Language English

☐ **Monograph** ☒ **Article dissertation** ☐ **Essay dissertation**

Abstract

Security vulnerabilities are a major concern for software developers. Some vulnerabilities are simple software bugs, while others result from fundamental changes in the software architecture and the underlying technologies. This dissertation studies the security concerns that arise from these ongoing architectural developments in modern software.

A major change in software systems over time has been the shift towards more distributed architectures. This development takes place on several levels. One of the most prominent changes has been the transformation of cloud applications towards a microservice architecture, in which loosely-coupled software modules communicate over the network through well-defined APIs. This architecture enables each module to be developed and operated independently. Moreover, the APIs can be opened for third parties to build add-on features. A similar architectural transformation can also be seen in desktop applications. Instead of running as a single computer program, many follow the client-server architecture and have separate frontend and backend components. The components run on the same computer and connect to each other through inter-process communication (IPC). There have also been changes to the underlying networking technologies. In enterprise and data-center networks, the traditional network paradigm is gradually replaced with software-defined networking (SDN) for more flexibility and control. Regular users, on the other hand, have adopted virtual private networks (VPN), which were initially developed for corporate networking, as a solution for enhanced security and privacy in the distributed software world.

The contributions of this dissertation include discovery of several new types of security failures in modern software, and empirical analysis of these vulnerabilities in deployed software products. We study the security of third-party add-ons in cloud applications and explain how they can bring cross-site scripting vulnerabilities to the applications. We show that such vulnerabilities appear widely in the wild. We also study the security of IPC between software components inside the computer and show that desktop application developers have overlooked critical security issues. We find IPC in many applications, including password managers, security tokens, and cryptocurrency wallets, to be vulnerable to impersonation and man-in-the-middle attacks mounted by local attackers. Furthermore, we study the security of SDN with focus on topology poisoning attacks by compromised network elements. We also examine commercial VPN services and identify several configuration flaws in the VPN clients. Finally, we analyze the potential solutions of each type of vulnerability.

Keywords Inter-process communication, Cloud-application add-ons, Virtual private network (VPN), Software-defined networking (SDN)

ISBN (printed) 978-952-64-0326-7	**ISBN (pdf)** 978-952-64-0327-4
ISSN (printed) 1799-4934	**ISSN (pdf)** 1799-4942
Location of publisher Helsinki	**Location of printing** Helsinki **Year** 2021
Pages 142	**urn** http://urn.fi/URN:ISBN:978-952-64-0327-4

Preface

This dissertation would not have been made possible without the help and support of many people. First, I would like to express my deepest appreciation to my supervisor, Professor Tuomas Aura. Pursuing a doctoral degree had never crossed my mind until you offered me the opportunity. You always believed in the best of me and patiently guided me through this journey. It was my privilege to have you as my supervisor.

I also am extremely grateful to my fellow worker, Sid Rao. Thank you for going through all the obstacles along the way with me. I would also like to extend my gratitude to my advisor, Markku Antikainen. Discussing ideas with you was a great help in pushing my research forward. Many thanks to the Secure System group for all the interesting presentations and discussions. Special thanks to Business Finland (Tekes) project 3772/31/2014 Cyber Trust and Academy of Finland project 296693 Secure Connectivity of Future Cyber-Physical Systems for supporting my research.

During my study, I did two summer internships at Google. I was fortunate to work with Thai Duong in one of the internships. You taught and inspired me a lot. In 2019, I started working as a security consultant at Nixu. I am thankful to all my colleagues at Nixu for the great working environment and for providing the support I needed to complete this dissertation.

Many thanks to the pre-examiners, Dr.-Ing. Ben Stock and Professor Davide Balzarotti, for the thorough reviews and insightful feedback. Additionally, I would like to thank Associate Professor Robert Lagerström for the great honor of having him as the opponent for my defense.

Finally, I am deeply grateful to my parents and sister for their everlasting encouragement. To my wife, Trang, thank you for having brought me all the luck I needed and making this work much more enjoyable.

Espoo, April 2, 2021,

Thanh Bui

Contents

List of Publications

This thesis consists of an overview and of the following publications which are referred to in the text by their Roman numerals.

I Thanh Bui, Siddharth Rao, Markku Antikainen, Viswanathan Bojan, Tuomas Aura. Man-in-the-Machine: Exploiting Ill-Secured Communication Inside the Computer. In *27th USENIX Security Symposium (USENIX Security 18)*, pp. 1511–1525, August 2018.

II Thanh Bui, Siddharth Rao, Markku Antikainen, Tuomas Aura. Pitfalls of Open Architecture: How Friends Can Exploit Your Cryptocurrency Wallet. In *Proceedings of the 12th ACM European Workshop on Systems Security*, pp. 1–6, March 2019.

III Thanh Bui, Siddharth Rao, Markku Antikainen, Tuomas Aura. Client-side Vulnerabilities in Commercial VPNs. In *24th Nordic Conference on Secure IT Systems*, LNCS vol. 11875, pp. 103–119, November 2019.

IV Thanh Bui, Siddharth Rao, Markku Antikainen, Tuomas Aura. XSS Vulnerabilities in Cloud-Application Add-ons. In *Proceedings of the 15th ACM Asia Conference on Computer and Communications Security*, pp. 610–621, October 2020.

V Thanh Bui, Markku Antikainen, Tuomas Aura. Analysis of Topology Poisoning Attacks in Software-Defined Networking. In *24th Nordic Conference on Secure IT Systems*, LNCS vol. 11875, pp. 87–102, November 2019.

Author's Contribution

Publication I: "Man-in-the-Machine: Exploiting Ill-Secured Communication Inside the Computer"

Viswanathan Bojan analyzed security threats to the communication between the desktop applications and the browser extensions of several password managers. The adversary that he considered was processes with the same access rights as the victim's, such as malware. I found that processes with much fewer privileges, such as those created by the guest user, could cause similar threats. They were, however, usually overlooked. The other authors and I generalized the problem and found more examples of it in the wild. My contribution to the writing of the publication was about 35 percent.

Publication II: "Pitfalls of Open Architecture: How Friends Can Exploit Your Cryptocurrency Wallet"

I first found a cryptocurrency wallet that was vulnerable to the Man-in-the-Machine threat model introduced in Publication I. The other authors and I analyzed more cryptocurrency wallets to understand the problem and find mitigation solutions. My contribution to the writing of the publication was about 40 percent.

Publication III: "Client-side Vulnerabilities in Commercial VPNs"

I observed that many commercial VPN providers used insecure configurations for L2TP/IPSec. Thus, I decided to analyze other VPN protocols' configurations as well to see whether they were configured securely. The other authors and I did the analysis together. My contribution to the

writing of the publication was about 35 percent.

Publication IV: "XSS Vulnerabilities in Cloud-Application Add-ons"

I discovered that cloud-application add-ons were vulnerable to XSS. The other authors and I analyzed the add-ons of popular cloud applications together to see how widespread the problem was in the wild and what could be done to mitigate it. My contribution to the writing of the publication was about 35 percent.

Publication V: "Analysis of Topology Poisoning Attacks in Software-Defined Networking"

Markku Antikainen and Prof. Tuomas Aura came up with the idea of analyzing what a compromised switch could do in a software-defined network (SDN). I pursued the idea with the focus on topology poisoning attacks. I came up with two out of the four attack variants presented in the publication and implemented all the emulation and simulation in the research. I did the first version of the research for my Master's thesis. Together with the other authors, we extended the work for the publication. My contribution to the writing of the publication was about 30 percent.

Other publications

During my studies, I also worked on the following publications, which are not included in this dissertation.

VI Thanh Bui, Tuomas Aura. Application of Public Ledgers to Revocation in Distributed Access Control. In *20th International Conference on Information and Communications Security*, LNCS vol. 11149, pp. 781–792, October 2018.

VII Thanh Bui, Tuomas Aura. Key Exchange with the Help of a Public Ledger. In *25th International Workshop on Security Protocols*, LNCS vol. 10476, pp. 123–136, March 2017.

VIII Thanh Bui, Tuomas Aura. GPASS: A Password Manager with Group-Based Access Control. In *22th Nordic Conference on Secure IT Systems*, LNCS vol. 10674, pp. 229–244, November 2017.

IX Marcin Nagy, Thanh Bui, Emiliano De Cristofaro, N Asokan, Jörg Ott, Ahmad-Reza Sadeghi. How Far Removed Are You? Scalable Privacy-Preserving Estimation of Social Path Length with Social PaL. In *Proceedings of the 8th ACM Conference on Security & Privacy in Wireless and Mobile Networks*, pp. 1–12, June 2015.

1. Introduction

Computer software has changed significantly from what it used to be a decade ago. Traditional applications typically follow the monolithic architecture [66], where the user interface and the application logic are combined into a single program and run locally in the computer. Modern applications, on the other hand, tend to have more *distributed* structure. Instead of running as a single computer program, they typically separate the user interface from the application logic. For instance, web applications follow the client-server architecture, where the server stores and processes user data in the cloud, while the user interface runs on a web browser on the user's device. The separation enables the application providers to implement all of their functionality on the server side while keeping the clients simple ("thin clients"). Furthermore, it enables the providers to maintain and update their applications easily and allows the users to access them any time from any device.

Nowadays, many desktop applications also follow the web software architecture and have separated frontend and backend components. They run on the same computer, and they connect to each other via an inter-process communication (IPC) channel [92]. Password managers are examples of such applications. They usually consist of two discrete components: a stand-alone application for managing the password vault and a browser extension for assisting the user on the web browser. The browser extension acts as an additional user interface of the password manager, and it communicates with the application via an IPC channel, such as network sockets or named pipes.

Apart from splitting into server and client sides, many cloud applications have switched to the microservice architecture [74], in which they are decomposed into loosely-coupled modules. Each module (i.e., a microservice) is operated as a small yet independent system, which communicates with other modules over the network through well-defined APIs. This architecture enables each module to be developed, deployed, and scaled independently. Furthermore, by implementing the application functionality as independent services that are loosely coupled through APIs, the APIs

can also be opened for third parties to build customized features for the application. The features implemented with these APIs are usually called "add-ons" [37]. Successful cloud applications have created marketplaces for add-ons and aim to grow an ecosystem of innovative add-on services around their core platform.

1.1 Research goals

This dissertation looks at *security aspects of modern applications*. While distributed application models solve many problems of the traditional, bundled applications, they also introduce new security concerns.

In particular, the communication between the software components increases the application's attack surface because, in many cases, they must communicate over hostile channels. For example, web servers usually communicate with their clients over the Internet, where network attackers may sniff the traffic or impersonate the endpoints. Even applications that run entirely locally inside one computer are not an exception. The reason is that the inter-process communication between their components might still be intercepted by local attackers, such as malware or another user on the same computer.

Furthermore, opening APIs to external developers to build customized features for the application may also create new security issues. Many add-ons are quick hacks by inexperienced developers. As a result, they might have lower code quality than the application itself and, thus, contain vulnerabilities that can be exploited to compromise the user data.

Besides, the architectural shifts in software also encourage new developments in computer networking. The growth in the volume of data exchanged, stored, and processed in data-centers and the trend of deploying applications and services on the cloud have driven the demand for new networking paradigms that are more flexible and responsive than the traditional ones. These are some of the key factors that triggered infrastructure providers to gradually switch to using software-defined networking (SDN) [55] where network functionality is implemented in logically centralized software controllers rather than in the network hardware. Such change radically alters the threat environment. Therefore, it must be carefully studied for possible design or implementation flaws.

Some traditional network technologies have also found new uses. For example, virtual private network (VPN), which was initially developed for corporate networking, is now commonly used by regular users who seek improved security and privacy on the Internet. Because of the increasing demand, a large number of commercial VPN services have appeared in the market. It is essential to carefully scrutinize them for flaws that could undermine the user's security and privacy.

In this dissertation, we address these security concerns. The goal is to find common security issues in the wild that are unknown or have not been given as much consideration as they deserve.

1.2 Research methodology

This work uses empirical analysis as the primary research methodology. We first look for popular security-critical applications that are used by a large number of users. We then study how they are implemented and analyze what could go wrong with such implementation. If we find a potential security issue, we verify the issue by implementing the attack to exploit it either in the application itself (Publications I-IV) or in an emulated environment (Publication V). When the issue is confirmed, we generalize the principle behind the vulnerability and analyze its root causes. We then investigate similar applications to find out how common it is and report the issue to the vulnerable applications' providers. We also suggest technical solutions to address the root causes or to mitigate the issue.

1.3 Contributions

This dissertation includes five original publications. The main contributions are the discovery of various severe vulnerabilities in a wide range of security-critical applications and the solutions to mitigate them. They can be summarized as follows:

- Publication I presents the attacks by nonprivileged users or processes against inter-process communication (IPC) between software components on the same computer. The attacks are similar to impersonation and man-in-the-middle attacks on the open networks but take place inside a computer, where application developers often do not expect adversaries. We call them Man-in-the-Machine (MitMa) attacks. In the publication, we show that many password managers and security tokens are vulnerable to the attacks, and exploiting them is not difficult. We also discuss potential mitigation techniques.

- Publication II extends the work in Publication I with the focus on cryptocurrency wallets. Specifically, desktop cryptocurrency wallet applications often provide a remote procedure call (RPC) interface, through which other applications can access the wallet's functionality either locally or remotely. In the publication, we study the security of the RPC interface in the MitMa attacker's presence. We show that the current

authentication mechanisms used by popular cryptocurrency wallet applications are not sufficient, and we discuss mitigation solutions.

- Publication III covers client-side vulnerabilities in commercial VPN services. These vulnerabilities are in the VPN client configurations, which allow attackers to strip off traffic encryption in the VPN tunnel or bypass server authentication. By exploiting these configuration flaws, the attackers can intercept network traffic to and from the victim. Apart from showing how common these vulnerabilities are, we provide guidelines for fixing them.

- Publication IV shows that add-ons that add customized commands and features to cloud applications may introduce security risks to the users of the applications. In particular, we demonstrate that many add-ons process untrusted user input in an unsafe way, allowing an attacker to perform cross-site scripting (XSS).

- Publication V investigates the vulnerability of software-defined networking to one particular class of attacks: topology poisoning. We analyze the attacks against SDN both qualitatively to understand the principles and quantitatively with simulations to assess and explain their impact. Our focus is specifically on attacks mounted from compromised SDN switches.

1.4 Structure of the dissertation

The rest of the dissertation is organized as follows: Chapter 2 presents the MitMa threat model and the related attacks (Publications I and II). Chapter 3 describes commercial VPNs and their configuration flaws that we found (Publication III). Chapter 4 gives an overview of cloud-application add-ons and shows how the attacker can exploit vulnerable add-ons to perform XSS attacks (Publication IV). Chapter 5 describes how SDN works and presents different variants of topology poisoning attacks that can be carried out by compromised switches (Publication V). Finally, Chapter 6 concludes the dissertation.

2. Man-in-the-Machine (MitMa)

Publications I and II focus on the security of inter-process communication (IPC), i.e., communication channels that are internal to the computer. In the traditional network threat model, the network is usually assumed to be hostile. The attacker can mount various types of attacks, such as man-in-the-middle and impersonation [26, 76], to steal sensitive user data from the network traffic. However, not all communication goes over the network. Computer software often comprises multiple components, such as a frontend application and a backend database, which obviously need to exchange information over some IPC channel inside the computer. Also, many modern desktop applications follow the architecture of web software and have a separate UI component, which connects to the business logic via RESTful APIs. Protecting user information that is exchanged over these local channels is as important as protecting data on the physical networks because there can be malicious components running inside the user device.

However, in Publications I and II, we show that such local communication usually does not get the same protection as communication over physical networks. The adversary model that we consider assumes the attacker to have login access as non-administrator or, at minimum, the ability to keep a non-privileged process running in the background. Thus, we use the name *Man-in-the-Machine (MitMa)* to describe such attacker. Publication I demonstrates the importance of the adversary model by showing that a large number of password managers and security tokens were vulnerable to it. Publication II extends the work with the focus on cryptocurrency wallets.

In this chapter, we will first give an overview the MitMa attacker. We will then describe the attack vectors that the attacker may exploit against against three common IPC methods: network sockets, Windows named pipes, and Universal Serial Bus (USB). We will also demonstrate each attack vector with a vulnerable application that we found. More vulnerable cases can be found in the publications.

2.1 The adversary

In the MitMa adversary model, we consider threats on *multi-user computers* that may have processes of two or more users running at the same time. The attacker is a logged-in user who tries to steal sensitive information from or interfere with another user of the same computer. The attacker does this by intercepting communication between the victim user's processes, as illustrated in Figure 2.1. Unlike malware that runs with the victim's privileges or as administrator, the attacker here is *nonprivileged*, and we do not assume that he can perform any kind of privilege escalation. The attack is similar to impersonation or man in the middle in computer networks but takes place inside one computer.

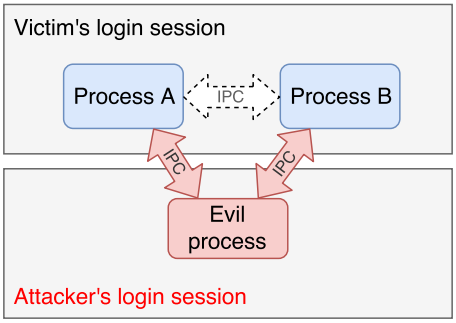


Figure 2.1. MitMa attack

To exploit IPC channels, in addition to having access to a nonprivileged user account, the MitMa attacker needs to run a process in the background when the victim is using the computer. On Linux and macOS, the attacker only needs to log in, run the process, and leave it running when he logs out, e.g., with the `nohup` command. On Windows, however, user processes are killed at the end of the login session. Thus, the attacker needs to do *fast user switching* [64] to leave his session in the background. The attacker can also remotely run his malicious process if SSH [108] or remote desktop server is enabled on the target computer.

While a PC is often considered personal, it is relatively common that several people can access it. For example, in enterprise environments that use centralized access control, users are typically able to sign in to each other's workstations. Shared family computers with multiple user accounts and machines with a guest account enabled are similarly vulnerable.

2.2 Attack vectors

Modern operating systems provide several IPC methods, but not all of them are vulnerable to the MitMa adversary model. An IPC method is vulnerable only if the following two conditions are satisfied: (1) one of its

communication endpoints binds to a specific identifier or name and waits for connections from other processes and (2) these connections can occur between different login sessions. We focus on three of such IPC methods: network sockets, Windows named pipes, and Universal Serial Bus (USB) communication. The attack vectors to these vulnerable IPC methods are described below.

2.2.1 Network sockets

While network sockets were originally intended for communication across a network, they are also used for IPC within one host. For this purpose, the server listens on the loopback interface (i.e., on one of the special `localhost` addresses `127.0.0.0/8` and `::1/128`) on a specific port number and waits for local client processes to connect to it. Any process, regardless of its owner, can listen on a port 1024 or higher as long as the port has not been taken by another process. Also, any local process can connect as a client to any port where a server is listening on the loopback interface.

Network socket communication is inherently vulnerable to local attackers like MitMa because it does not have any built-on access control to restrict the communication endpoints. It is the responsibility of the client and server processes to authenticate each other on the application layer.

Attack vectors. By running malicious processes, the MitMa attacker can easily perform a client impersonation attack by connecting as a client to any `localhost` port where a server is listening. The attacker can also perform server impersonation by binding to the server port before the legitimate process does. This is possible because any process, regardless of its owner, can listen on a port 1024 or higher as long as the number has not been taken by another process. Combining server and client impersonation to a full man-in-the-middle attack, however, is not trivial because the legitimate server and the attacker cannot both bind to the same port number. One way for the attacker to overcome the limitation is to alternate between the client and server roles and replay messages.

2.2.2 Windows named pipes

Named pipes are available on both Windows and Unix systems. We focus on Windows named pipes, in which the actual vulnerabilities were found. Windows named pipes follow the client-server architecture like network sockets. However, instead of binding to a port number and an IP address as network sockets do, a named pipe is bound to a name. When no pipe with the given name exists, any process can create it. The named pipe can have multiple instances, each of which connects exactly one pipe server and one pipe client. The creator of the first instance decides the maximum number of instances and specifies the *security descriptor*, which includes

an access control list (DACL) that applies to all instances of the named pipe.

Attack vectors. Despite the built-in access control mechanism, it is easy to overlook the necessary security controls for named pipes, thus creating vulnerabilities. Specifically, if the named pipe is created with the default security descriptor (i.e., everyone has read access), or with open read-write access, the attacker can connect to it and impersonate the legitimate pipe clients. The pipe server would have to configure the DACL on the named pipe object carefully to allow access for only legitimate clients.

In addition, the attacker can hijack the pipe name by creating the first pipe instance and thus become the owner of the pipe object and impersonate the pipe server. Furthermore, if the legitimate server is careless, it will not check that it is creating the first instance of the pipe. By choreographing the creation of the instances and client connections, the attacker can then become a man in the middle between the legitimate client and server and pass messages between the two pipes.

2.2.3 Universal Serial Bus

USB allows peripheral devices to communicate with a computer. Even though USB is not actually an IPC mechanism, we considered it in this work because USB human interface devices (HID), such as keyboards and security tokens, bind to an identifier and wait for incoming connections in a similar way as network sockets and Windows named pipes do. Furthermore, in Windows, a user can access USB HIDs plugged in by other users, making them vulnerable to our adversary model. Such cross-user access, however, is not possible in other operating systems.

Attack vectors. In Windows, the MitMa attacker can access USB HIDs plugged in by other users. Thus, their security depends on application-level security mechanisms implemented in the hardware or software.

2.2.4 Safe IPC methods

It is worth noting that some IPC mechanisms, such as anonymous pipes and socket pairs, are not vulnerable to our attacks. In these methods, both communication endpoints are created at the same time by the same process, which prevents an untrusted process from getting to the middle. Unfortunately, these IPC methods can only be used between related processes (typically parent and child), which severely limits the software architecture.

On macOS, apart from the same IPC methods that are available on Windows and Linux, there are also Mach IPC methods that are based on the Mach kernel. These IPC channels are associated with a login

session [3], and a process from one login session cannot interact with another. Thus, they are immune to MitMa attacks between users.

2.3 Related work

There has been not much work in the literature on IPC security. Windows named pipes have been an attractive target for security analysts, even though the OS offers security controls to the named pipes using DACL. The reason is that there could be write access for everyone because of the developer's negligence. In such scenarios, even a remote attacker may be able to impersonate the pipe client to perform code execution or denial of service [19, 24]. The server-impersonation and name-hijacking attacks explained in the previous section are not feasible for such remote attackers.

Additionally, Windows named pipes are also known to be vulnerable to an impersonation attack [101] (unrelated to the client or server impersonation in this work). The pipe server impersonates its client's security context, which allows it to perform actions on behalf of the client. This attack requires the server and the client processes to run as the same user or for the server to run as a superuser, which is a stronger assumption than our threat model.

Vulnerabilities have also been found for other IPC mechanisms. Xing et al. [106] demonstrated that a malicious application on macOS and iOS can access another application's resources despite their isolation. The attacks intercept IPC in a way similar to ours, but the malicious binary is executed with the victim's privileges. Related problems have also been found in Android applications' isolation [35, 90].

2.4 Our study

We analyzed various widely-used applications against the MitMa adversary model. In Publication I, we chose password managers and security tokens as our primary case study, and in Publication II, we focused on cryptocurrency wallets. The reason we chose these application classes is that the information they communicate over IPC is critical and, thus, it is easy to identify security violations. We usually selected the popular applications in each class. However, in some cases, we also chose interesting or architecturally different applications because we wanted to have variety.

The main observation is that application developers usually have an ambiguous attitude towards local attackers and the security of IPC channels. They often make some attempt to authenticate or encrypt the communication, but rarely with the same prudence as seen in communication over physical networks. In this section, we will present three vulnerable appli-

cations to demonstrate the attack vectors that we presented in the previous section. More vulnerable applications can be found in Publications I and II.

2.4.1 Case 1: Bitcoin Core wallet

Cryptocurrencies, such as Bitcoin [73] and Ethereum [105], have become remarkably popular in the last decade. Users access a cryptocurrency with a *wallet* application, which manages the private/public key pairs that are used for transactions and allows the user to store, send and receive the cryptocurrency. We will use Bitcoin Core[1], one of the most popular Bitcoin wallets on desktop, to demonstrate the attack vectors against network sockets. We considered Bitcoin Core v0.16.3, the latest version of the application at the moment of our study (October, 2018).

Like most desktop wallet applications, Bitcoin Core provides a remote procedure call (RPC) interface, through which other applications can access the wallet's functionality (e.g., querying account balance, or making transactions) either locally or remotely. When Bitcoin Core is started, it runs the RPC server in the background on `localhost` port 8332 by default so that only local clients can access it. The RPC server does not support TLS, and it authenticates the client with HTTP basic access authentication [86], in which the client sends its username and password to the server in an HTTP header. Bitcoin Core allows the user to encrypt each private key with a password, and the client must send the password as HTTP POST data to the RPC server to unlock the key before the key can be used for transactions.

Attacks. Since only the RPC server authenticates the client, it is possible for a MitMa attacker to impersonate the server to benign clients. This can be done by the attacker running a malicious server on port 8332 before the benign server starts and accepting any clients regardless of the credentials they present. Luckily for the attacker, when Bitcoin Core fails to start the RPC server, it does not notify the user about the failure. As a result, the attacker can capture both the plaintext authentication credentials and the decryption passwords for the victim's private keys sent by the benign clients without alerting the victim. Once the attacker has obtained these, the attacker can terminate the malicious RPC server, wait for the benign RPC server to restart, use the captured secrets to connect to the benign RPC server and steal the coins from the wallet.

2.4.2 Case 2: Password Boss

Password Boss[2] is a popular desktop password manager. We will use it to demonstrate the attack vectors against Windows named pipes. We

[1] https://bitcoin.org/

[2] https://www.passwordboss.com/

considered Password Boss v3.1.3434, the latest version of the application at the moment of our study (October, 2017).

As many other password managers, it is integrated to web browsers with *browser extensions*. Passwords are managed by the Password Boss desktop application, and they are communicated to the browser extensions over a local communication channel so that the browser extensions can assist the user in creating and storing passwords and also in entering them into login pages.

At the moment of testing, Password Boss supported Chrome and Firefox. On Windows, the communication between the desktop application and the browser extensions is implemented with a Windows named pipe as follows. When the desktop application is started, it creates a named pipe with a fixed name and a maximum of 50 instances. The browser extensions connect to the named pipe as pipe clients, and they exchange messages with the desktop application in plaintext. The access-control list on the named pipe is set to allow all authenticated users on the computer to read and write to the pipe's instances. (Note that browser extensions implemented in JavaScript cannot directly access Windows named pipes. Instead, they communicate via a native-code component using a mechanims called *native messaging*[3].)

Attacks. Because of the improper configuration of the named pipe, a MitMa attacker can perform a man-in-the-middle attack as follows. First, the attacker connects as a client to the desktop application's named pipe instance. The attacker then creates another instance of the named pipe, which is possible thanks to the unnecessarily high maximum number of instances. When a Password Boss's browser extension tries to communicate with the desktop application, it will connect to the attacker's instance because it is the only one available. The attacker can thus sit between the two pipe instances, forward messages, and read their content, including passwords.

2.4.3 Case 3: FIDO U2F security key

FIDO U2F [36] is a USB HID security key device that is used to enable strong two-factor authentication on online services. It is supported by various online service providers and browsers. We will use it to demonstrate attack vectors against USB communication.

To use a FIDO U2F device for two-factor authentication, the user must first register the device to the online service. During the registration, the device generates a service-specific key pair and stores it together with a key handle (i.e., identifier) and the origin URL of the service.

After the device has been registered, it can be used as a second factor

[3]https://developer.chrome.com/apps/nativeMessaging

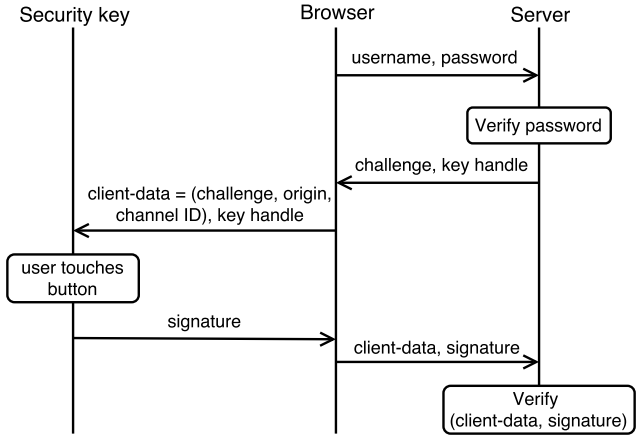


Figure 2.2. Basic authentication flow with U2F security key [4]

as illustrated in Figure 2.2. First, the browser receives a challenge together with a key handle from the web server. It then forms the so-called `client-data` object and sends the object to the device for signing. The user needs to activate the device by touching a button on the device. This is meant to prevent unauthorized use of the device. Once the device has been activated, it signs the first received client-data object and sends it back to the browser. The browser then delivers the signed object to the web server for verification.

Attacks. As mentioned in Section 2.2, the MitMa attacker can access USB HIDs plugged in by other users in Windows. Thus, if the MitMa attacker has compromised the victim's password, the attacker can also subvert the hardware-device authentication as follows. The attacker first creates a malicious (browser) process that runs on the victim's computer and tries to log into one of the user's online services. The attacker's process then sends client-data objects to the U2F device at a high rate. When the user decides to log in to any service using U2F authentication and touches the button on the device, there is a high probability that the attacker's request will be signed. The user may notice that the first button press had no effect, but such minor glitches are normal in computers and typically ignored.

2.5 Discussion

In Publication I, we found 6 password managers, 2 hardware tokens, and 5 other security-critical applications that were vulnerable to the MitMa attacks. In Publication II, 7 wallet applications of 5 popular cryptocurren-

cies were discovered vulnerable. There are likely to be other vulnerable applications in the wild that can be exploited by the MitMa attackers. The explanation why the problems are so widespread is probably twofold. First, developers are inclined to consider the user's computer a trusted environment. Second, the best practices for secure IPC are not documented, and therefore developers may simply be unaware of the threats and solutions. We therefore believe that the best way to address both of these potential explanations is to publicly discuss the attacks and defenses, as we did in the publications. Over time, better tools such as safe APIs and security test benches could help eradicate the entire class of problems.

The idea of protecting the users of a multi-user computer system from each other takes us back to the early days of computer security. With personal computers, this has not been perceived as so important. It has also become common wisdom among information-security experts that, if the attacker can run a process on the computer, they always can find a way to perform privilege escalation [6, 50, 106] and gain full administrative access. There is, however, the opposite trend towards greater isolation of applications from each other and containing malicious applications. This started in mobile devices, but desktop operating systems are beginning to provide similar protections. The MitMa attacks are one way for a non-privileged process to circumvent isolation boundaries within the computer, and we believe that the observations of the papers will prove useful in the design of application-isolation mechanisms.

3. Configuration Vulnerabilities in Commercial VPNs

Virtual private networks (VPN) [104] were originally developed for connecting geographically distributed corporate networks to each other with encrypted tunnels so that they would form a single secure logical network. However, today they are commonly provided as subscription-based services by *commercial VPN* providers to regular Internet users for personal purposes, such as protecting Internet traffic when using a shared WLAN, hiding sensitive online activities, and accessing geo-blocked media content.

In Publication III, we study the client-side configuration of commercial VPNs. The work was primarily motivated by the observation that many commercial VPN providers configured L2TP/IPsec, a popular VPN protocol, in an insecure way[1]. Specifically, the protocol relies on IPsec [11] to provide the secure transport, but many VPN providers used a single publicly-known pre-shared key (e.g., "12345678") for all users to authenticate the IPsec tunnels. If an attacker knows the pre-shared key, the attacker can perform a man-in-the-middle attack on the VPN connection and, as the result, obtain all the network traffic to and from the victim's computer. This problem was discussed on public forums in 2016. However, when we re-analyzed the 14 insecure commercial VPN services mentioned in the discussion, we found that 10 of them were still using the insecure configuration. Since such a vulnerability remains opaque to most end-users, we felt that it was important to further scrutinize commercial VPN services for flaws that could undermine the user's security and privacy.

In this chapter, we first give an overview of how commercial VPNs work and related work in the literature. We then describe our study in detail.

3.1 Background

This section gives an overview of commercial VPNs and related work.

[1] https://gist.github.com/kennwhite/1f3bc4d889b02b35d8aa

3.1.1 Commercial VPNs

The principle of commercial VPNs is quite simple: they tunnel the user's Internet traffic through a trusted remote server before it is forwarded to its final destination. This achieves two goals: first, the traffic is protected by an encrypted VPN tunnel against dangers in the access network and, second, the destination server does not learn the real IP address of the client.

Most commercial VPN providers provide *native desktop applications*, which set up the VPN connection, to their users. These applications are usually available only for the Windows and macOS operating systems. For Linux users or those who prefer not to install the provided applications, the VPN service providers usually give instructions for configuring the built-in VPN client in the user's operating system (OS) to work with their servers.

Commercial VPN desktop applications. The desktop applications provided by commercial VPN providers typically work as follows. The user must first enter his VPN user-login credentials into the application. The application then pulls configuration data, such as VPN server addresses, roots of trust for the authentication, and VPN-client credentials (which are not necessarily the same as the user-login credentials), from the VPN provider's server. When the user wants to use the VPN, the application typically allows the user to choose the location of the VPN server and the VPN protocol (e.g., PPTP, SSTP, OpenVPN) for the tunnel implementation and then sets up the VPN connection accordingly.

The process of setting up a VPN connection is similar regardless of the protocol used: First, the VPN client establishes a VPN tunnel to the VPN server with the selected protocol. The client and the server authenticate each other in the protocol handshake with predefined credentials and roots of trust. If the authentication succeeds, the client and server negotiate various parameters for the VPN connection, such as the encryption scheme and the DNS servers, and the VPN tunnel is established. After that, the client creates a virtual network interface (e.g., `tun0` or `ppp0`) and modifies the host's routing table to redirect all Internet traffic towards it. Traffic that goes through the virtual interface is tunneled to the VPN server through the established VPN connection.

3.1.2 Related work

Prior work in the literature has shown various security and privacy threats against the users of commercial VPN services.

Deanonymization is one of the biggest threats against commercial VPN users. First, it is obvious that VPN users are not anonymous from their service providers. They must blindly trust the providers to not be malicious

and to not disclose their activity log to third parties. The data that the providers retain about their users exacerbates this problem further. For example, many VPN services ask for the user's personal information at registration time. A number of them even retain the traffic log of each VPN connection [75]. Another way that a user can be deanonymized is through end-to-end traffic correlation [60, 70, 89]. Furthermore, users typically tend to give priority to connection quality over location diversity. This means that they often rely on the same small subset of VPN servers, making end-to-end attacks easier. User anonymity can also be compromised because of misconfiguration [10]. In many cases, the VPN reveals user information due to IPv6 traffic and DNS leakage [82]. Such leaks can also occur due to advanced VPN functionality such as port forwarding [81], WebRTC [7], and web cookie synchronization [79].

Security of the VPN tunnel is another concern. Cryptographic weaknesses can undermine the security of the tunneled traffic. Nafeez disclosed the compression oracle attack on OpenVPN compression algorithms, where an attacker can send cross-domain requests when an HTTP website is tunneled through OpenVPN connections [72]. Felsch et al. demonstrated that reusing the same key pair across IKEv1 and IKEv2 allows an attacker to bypass authentication as well as perform impersonation attacks [34]. Weak pseudo-random number generators (PRNG) in the VPN implementation can also subvert the security of the cryptographic protection. The weaknesses could be due to faulty implementation, such as a hard-coded seed key along with a legacy PRNG [25], an intentional backdoor [17], or an intentionally insecure random-number generator [22]. In any case, an attacker may be able to recover the secret keys for the VPN tunnel and intercept the traffic passing through the VPN.

3.2 Our study

As we described in the previous section, commercial VPNs have undergone severe scrutiny and various problems have been found. In Publication III, we extended this theme in the literature with focus on the security of the VPN tunnel configuration. Specifically, we studied how popular commercial VPN providers set up, or how they instructed users to set up, desktop VPN clients for common VPN protocols. The goal was to find configuration flaws that could undermine the user's security and privacy.

In this section, we first describe the adversary model and our methodology for systematically finding vulnerabilities in VPN client configuration. We then present some highlighted results from the study.

3.2.1 Adversary model

In the study, we considered two types of attackers whose ultimate goal was to bypass the protection mechanisms of the VPN connection and steal sensitive data transmitted through it: *network attacker* and *MitMa attacker*.

The network attacker is the standard attacker model for network security. We considered an active network attacker who could intercept and modify network traffic originating from and destined to the user's machine. The attacker could, for example, be a rogue hotspot operator or a compromised core-network operator.

The MitMa attacker could be any non-privileged user on the same computer, as described in Section 2.1. We included this type of attackers to the study because the VPN client software on the user's computer often comprised multiple separate components that communicate with each other via IPC (e.g., network sockets, files on disk) and the attacker might be able to exploit them to steal sensitive information or to modify the VPN connection settings.

3.2.2 Methodology

We selected 30 popular commercial VPN services based on popularity and advertised features. We focused on standard VPN protocols, including PPTP [40], L2TP/IPsec [97, 11], IKEv2 [51], Cisco IPsec [2], SSTP [65], OpenVPN [33], and SoftEther VPN[2].

For each of the selected commercial VPN services, we first looked at the way its provided VPN client application created and configured VPN connections. If the VPN service provider also recommended a built-in client in the OS or a third-party client, we scrutinized the provided configuration instructions and unchanged default settings. In both cases, we looked for potential misconfigurations and architectural mistakes that might compromise the security of the resulting VPN connection. We did not try to find flaws in the cryptographic protocols themselves or code-level implementation errors.

When we found a potential client-configuration issue, depending on the type of the attack, we verified it with experiments. In particular, for network attacks, we created a fake VPN server, configured the VPN client with the potentially vulnerable settings, routed the VPN client's traffic to the fake server, and checked whether the client could detect the attack. For local attacks, we created two users accounts on a test machine: one acted as the honest user and the other as the MitMa attacker. We then checked whether the attacker was able to exploit the vulnerability in the VPN client application that was running in the honest user's login session.

[2]https://www.softether.org/

3.2.3 Study results

We found security issues in the configurations of all of the selected protocols. They existed not only in the way the commercial VPN providers configured the protocols but also in their instructions for built-in or third-party VPN clients. In this section, we will describe the issues in three protocols: Cisco IPsec, IKEv2, and OpenVPN. The details of the remaining issues can be found in Publication III.

Cisco IPsec is not only widely used in enterprise VPNs, but it is also supported by many commercial VPN services. Like L2TP/IPsec, the protocol uses IPsec to tunnel traffic. The main distinguishing feature of Cisco IPsec is that, after the communicating nodes have completed the conventional IKEv1 authentication, an additional phase of Extended Authentication (XAUTH) [80] is performed to authenticate the user. XAUTH allows various types of user authentication, such as challenge-response and one-time password.

In the study, we found that all of the VPN services that supported Cisco IPsec had the same issue as with L2TP/IPsec: they used a single pre-shared key to authenticate the IPsec channel for all of their users. This allowed the network attacker to perform MitM attacks on these IPsec connections and to obtain all the network traffic.

IKEv2 is a more modern VPN protocol based on IPsec. As the name suggests, the protocol uses IKEv2 [51] for authentication as well as establishing and maintaining security associations. The server in a typical IKEv2 VPN is authenticated with a server certificate, while the client is authenticated with the EAP-MSCHAPv2 protocol [44] and user password.

We observed that some of the commercial VPN providers instructed their Linux users to install StrongSwan[3], an open-source IPsec client, to use IKEv2 with their services. To establish an IKEv2 connection with StrongSwan, the user has to create a profile for the connection. The profile that the providers instructed their users to create was as follows (only important parts are shown):

```
leftauth=eap-mschapv2
...
right=<server-address>
rightauth=pubkey
rightid=%any
```

Left and right indicate the client and the server, respectively. With such a profile, the server uses a public key for authentication while the client uses EAP-MSCHAPv2, as we described above. The problem is with the `rightid` setting, which tells how the server should be identified in the authentication. Since it is set to `%any`, the client will accept any certified

[3]https://www.strongswan.org/

server regardless of its identity. Consequently, the network attacker can pick any domain that it owns, purchase a certificate from a widely trusted CA, and then impersonate the server in the server authentication step because the client does not check the name in the certificate.

Fortunately, EAP-MSCHAPv2 actually provides mutual authentication with the user password. The binding of this authentication to the SA prevents the attacker from completing the protocol without knowing the password. Thus, the misconfiguration effectively reduces the security of IKEv2 to that of EAP-MSCHAPv2, which is significantly weaker [61, 88].

OpenVPN appears to be the most widely supported protocol by commercial VPN services. It uses TLS as the underlying authentication and key exchange protocol. Commercial VPN services deploy OpenVPN in the client-server mode, in which the server authenticates itself to the client with an X.509 certificate signed by a CA that the client trusts while the client proves its identity with a username and password.

Despite the wide range of configuration options that OpenVPN supports, we did not find any broken configuration that would allow the network attacker to compromise the OpenVPN connection. We found, however, that the MitMa attacker could steal the username and password that were used for authenticating the client in some VPN services as follows. To support OpenVPN, commercial VPN providers included the open-source `openvpn` client binary in their client software and run it as a daemon. To create an OpenVPN connection, the `openvpn` daemon required configuration information including the server address, server name, trusted CA certificate, and the user's VPN username and password. The problem was that some commercial VPN client applications passed the information to the OpenVPN daemon via configuration files, but the files were readable to all users on the user's computer. Therefore, the MitMa attacker could capture this sensitive information. Some of these services removed the credentials from the file after the connection was established, but this still left a window of a few seconds to capture the information.

3.3 Discussion

It appears that commercial VPN services compete by providing the maximum number of features, such as different VPN protocols, and maximum ease of use, and that security is a secondary concern for them. For example, the issue of using publicly known pre-shared keys for L2TP/IPsec has been known for years, but it has not been addressed by most commercial VPN services. While this issue could be solved by provisioning certificates to the clients, that would be an administrative hurdle for the VPN service providers and might scare away non-expert customers. This could be the reason why they have opted to continue the insecure practices.

Furthermore, the vulnerabilities that we found were surprisingly common among the commercial VPN services that we analyzed. For example, 14 of the selected services supported Cisco IPsec, and all of them had the publicly known pre-shared key problem described in Section 3.2.3. This seems to indicate lack of technical knowledge or security awareness across the commercial VPN industry. We hope that our study will, at least to some extent, increase awareness about the importance of correct VPN configuration among the commercial VPN developers and operators.

4. XSS vulnerabilities in cloud-application add-ons

Modern web applications store and process user data mainly in the cloud, and their user interface is implemented with HTML and JavaScript and runs on the web browser. With this architecture, the users do not need to install or update the applications, and sharing and synchronizing between users and services becomes easier. Furthermore, many cloud applications follow the microservice architecture where much of the functionality is implemented as independent services that are loosely coupled to the core service through APIs. The APIs can also be opened to external developers so that they can develop customized features for the applications. These features are usually called "add-ons". For example, Google Docs is a well-known online document editor, and its Translate add-on allows the user to translate text to another language — a feature that is not part of the core service.

While cloud-application providers have come a long way in securing their applications, third-party add-ons may be developed by inexperienced developers and, thus, bring in security vulnerabilities. In Publication IV, we present one of the first, if not the first, studies on the effect of add-ons on cloud-application security. The study focuses on how add-ons process untrusted user input. This is a critical issue because the emphasis on collaboration and data sharing in the cloud applications makes it easy to exploit vulnerabilities in the handling of untrusted data. The study reveals that many cloud-application add-ons are vulnerable to cross-site scripting (XSS). In this chapter, we first give an overview of how cloud-application add-ons work and how they can be vulnerable to XSS attacks. We then discuss the impact of the vulnerabilities and show how common vulnerable add-ons are in the wild.

4.1 Background

In this section, we give an overview of cross-site scripting attacks and cloud-application add-ons as well as related work.

4.1.1 Cross-site scripting

In a cross-site scripting (XSS) attack [38], the attacker injects malicious client-side code, typically JavaScript, to a website that does not sufficiently filter user inputs. When a victim accesses the site, the injected code is executed in the user's web browser in the same context as the legitimate scripts on the same page. Thus, the code gains unauthorized access to the site's resources and bypasses the same-origin policy (SOP) [109].

In general, there are four types of XSS attacks: *stored XSS*, *reflected XSS*, *DOM-based XSS*, and *persistent client-side XSS*. The first three can be considered the "traditional" XSS attacks [38] and are well-known in the web security community, while the fourth is relatively new in the literature [93].

XSS variants can also be classified into *server-side XSS* and *client-side XSS*[1]. The former occurs when untrusted user data is included in an HTML response generated by the server, while the latter occurs when a JavaScript call uses untrusted user data to update the DOM of the vulnerable page in an unsafe way.

4.1.2 Cloud-application add-ons

Add-ons (also known as add-ins, plugins, extensions, or apps) add customized commands and features to a cloud application, called the *host application*. Cloud-application vendors often distribute add-ons for their applications through an online *marketplace*. For example, the G Suite marketplace[2] lists add-ons of Google applications, such as Gmail and Google Docs.

Each add-on is basically a separate web service with its own server and web front-end, but it has access to the user data and some core functionality of the host application through APIs defined by the application. When the user starts an add-on, the host application typically loads the add-on UI into an `iframe` and displays it as a part of the application user interface. There are two fundamentally different ways for the add-on UI to interact with the host application. It can communicate either locally with the host-application UI component or via the add-on server. In the latter case, the add-on UI usually connects to the add-on server in the cloud, which interacts with the host application server and accesses the user data through backend APIs that are not visible to the user.

Access control. Cloud-application vendors typically implement *permission-based access control* for add-ons to limit their access to user data in the host application. Each add-on has a list of the permissions which it requires to

[1] https://owasp.org/www-community/Types_of_Cross-Site_Scripting
[2] https://gsuite.google.com/marketplace

operate. The host application usually asks the user to explicitly approve the permissions when the user runs the add-on for the first time or during its installation. In cases where the add-on is updated and needs new permissions, the host application will ask the user to review and approve them again. This access control tends to be rather coarse grained, i.e., the user has to grant all the requested permissions for either all user data or for a specific document. Furthermore, the add-on retains the permissions until the user uninstalls it.

4.1.3 Related work

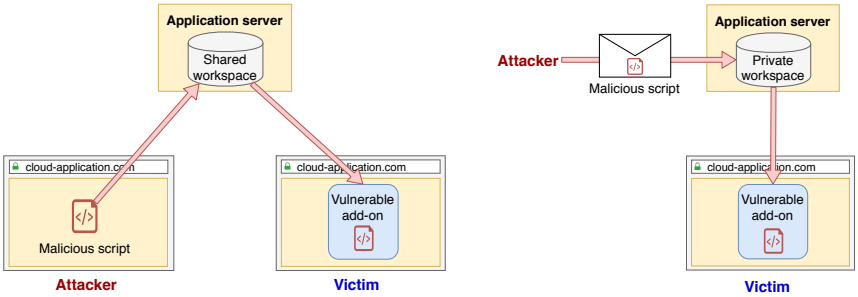
This section summarizes the related literature about XSS attacks and security analysis of add-on ecosystems outside the domain of web applications.

XSS vulnerabilities. XSS has always been a prevalent problem in web security [59, 63, 71, 93, 94, 95]. The literature on XSS defenses includes both client and server-side solutions. The client-side solutions involve sanitizing user input before sending it to the server. However, distinguishing between trusted and untrusted content is a challenging task. This is why the validation of web pages is sometimes outsourced to the browsers [96] or to the web firewalls that run on the client device [46, 54].

Taint checking is a server-side protection mechanism where the input originating from untrusted sources is flagged as potentially malicious and subjected to further scrutiny. Similar techniques can be employed on the client side if combined with the static analysis of the input processing [98, 100]. There are other server-side solutions that involve, for example, passive monitor of web traffic [49] or dynamic comparison of HTTP responses with pre-defined ones [18].

Add-on security. Even though there are add-ons for almost any type of software, it is mostly the browser add-ons which have undergone critical security scrutiny. For example, Google Chrome has a browser extension ecosystem, where the extensions themselves [47], their architecture [21] and protection mechanisms [42] have been targets of security research. Weissbacher et al. inspected Chrome extensions for privacy violations (e.g., leaking browsing history) by observing the network traffic patterns generated by the add-ons [102]. Firefox add-ons have also undergone similar vetting [13, 14, 15]. Text editors also have add-on ecosystems (e.g., Sublime plugins) that have been recently criticized for security vulnerabilities. Azouri Dor analyzed several text editors and found that it is possible for a malicious add-on to achieve privilege escalation on the victim's computer [12].

Cloud-application add-ons are still a relatively new phenomenon, and their effect on the application security has not been widely studied. We believe that it deserves the same attention from the security research



(a) With shared workspace **(b)** With outside input

Figure 4.1. XSS attacks with vulnerable add-ons

community as any other type of add-ons.

4.2 XSS against cloud-application add-ons

In Publication IV, we studied the security risks that arose from potential security vulnerabilities in cloud-application add-ons. The focus of the study was on cross-site scripting.

4.2.1 Attack variants

We identified two types of XSS attacks against vulnerable add-ons, as illustrated in Figure 4.1:

1. *Attack with shared workspace*: The attacker and the victim are colleagues, friends or remote collaborators who use the same cloud application and share a *workspace*. The workspace basically is any environment through which changes made by one user are propagated to the others (e.g., a shared Google Docs document). The attacker injects malicious JavaScript code into the workspace. When the victim enables the vulnerable add-on for the shared workspace and the add-on renders the attacker's input in an unsafe way, the injected script may be executed in the add-on's `iframe`. This way, the attacker has performed an XSS attack on the victim.

2. *Attack with outside input*: Some host applications accept external input, which may contain XSS code for an add-on that later processes the data. For example, if the host application is an email service (e.g., Gmail or Outlook), the attacker can hide the malicious script in an email and send it to the victim. If the victim has enabled a vulnerable add-on to process emails, the injected script may again find its way to the add-on's `iframe` and be executed there like any JavaScript in that frame.

How the attacker injects the script into the shared workspace is naturally specific to the cloud application, to the add-on, and to the vulnerability that is being exploited. In any case, the root cause of the above attacks is that the vulnerable add-on routes the untrusted user input to JavaScript's execution sinks in the add-on UI without sanitizing it. For example, the malicious data from the attacker can be executed if the add-on renders it as HTML rather than as text.

4.2.2 Attack consequences

We now discuss what the attacker can gain by performing XSS on a vulnerable add-on. As in all XSS attacks, with the ability to run arbitrary scripts in the context of the vulnerable add-on, the attacker can access HTML5 APIs and request access to local resources, such as geolocation, or authorization to access external resources owned by the victim user. The attacker can also access data on the add-on server through its APIs. In a microservice architecture, the add-on server is likely to have its own data storage. Furthermore, the attacker can spoof another user interface and trick the user into entering confidential data or credentials.

The attacker, however, cannot directly access data in the host application. This is because the malicious script runs in the add-on's `iframe`, whose origin is different from the host application's. Thus, it cannot access the DOM model of the host-application within the web browser or the cookies related to the host application.

To understand the real consequences of XSS attacks against cloud-application add-ons, we analyzed the add-on architectures of three popular cloud application suites: Microsoft (MS) Office Online[3], G Suite[4], and Shopify[5]. What the attacker could do with malicious code execution in each case is as follows.

- *MS Office Online* is a cloud-based office suite, which includes popular office applications like Word, Excel, PowerPoint, and Outlook. Add-ons of MS Office Online applications interact with the host application on the client side via JavaScript APIs. Specifically, the `window.postMessage()` function [69] is used for cross-origin messaging between the add-on `iframe` and the parent application window.

  With this architecture, once the attacker has managed to inject malicious scripts into the add-on UI, the attacker can access any resources in the host application which the add-on is permitted to access. This is because the scripts can call the JavaScript APIs provided by the host application. Since the attacker's scripts run within the add-on's `iframe`, it

[3]https://office.com/
[4]https://gsuite.google.com/
[5]https://shopify.com/

is not possible for the host application to differentiate between malicious requests from the injected code and legitimate ones from the add-on UI code.

- *G Suite* is another office suite, which is developed by Google. Some well-known applications in the suite are Google Docs, Google Sheets, and Gmail. The main difference to MS Office Online's add-ons is that G Suite add-ons are always hosted in the Google cloud and they interact with the host application's data on the server side. The add-on server is basically a Google APIs client[6], and it obtains the *access token* for the user's private data when the user starts the add-on for the first time. The add-on UI sends requests to interfaces defined by the add-on server, and the server implements the desired actions on user data as well as returns responses.

  At first glance, since the host application window does not accept local messages from the add-on UI, it appears that the XSS attacker cannot compromise the victim's data. However, there is a very common Google API used by add-on UIs, which allows the attacker to bypass this limitation: the Picker API[7]. It is used for the user to select a file or folder that is stored in Google servers. Like any other Google APIs, the Picker API requires an access token to operate. Add-on servers commonly create an interface by which the add-on code running in the browser can obtain a copy of the server's token; this is even a recommended practice [1]. Thus, the injected XSS code can request the access token from the same interface. As the result, the attacker gains the same permissions to the user's data as the add-on server.

- *Shopify* is an e-commerce platform with which small merchants can create online shops. Each shop is managed through a web admin interface, on which the owner can access the built-in services of the platform. Shopify add-ons integrate third-party services into this admin interface.

  Unlike the other two application suites, we could not find any way for the XSS attacker to gain access to Shopify data through the published APIs. The add-on server runs in the cloud and accesses the shop data with the Shopify REST APIs over HTTPS. It is authorized with an OAuth 2.0 access token, which it obtains when the user starts the add-on for the first time. Since the add-on accesses the user's data from the server side, the attacker cannot directly access the victim's data with the injected client-side script. There was also no dangerous API like the Picker API used by G Suite add-ons.

We can see different design choices in the three add-on architectures

[6] https://developers.google.com/apis-explore
[7] https://developers.google.com/picker/

above and how they significantly affect what the attacker gains with XSS attacks on the add-ons. However, even if the attacker cannot gain access to the data in the host application through the published APIs, spoofing always poses a great threat to the user. In particular, we found that in all three analyzed application suites, the attacker could create a malicious add-on or app with the same name as the vulnerable add-on and display an OAuth authorization prompt to trick the victim into granting it permissions to access private resources. Since the victim is already familiar with the authorization prompt that the add-on displays when it runs for the first time or when it is updated (see Section 4.1.2), there is a high chance that the trick will work.

4.3 Empirical analysis

To gain deeper understanding of the attacks described above and to find out how common they are in the wild, we looked for vulnerable add-ons in the marketplaces of the three selected cloud application suites (MS Office Online, G Suite, and Shopify). For each of them, we selected 100 free add-ons, in which 50 were popular ones and another 50 were selected randomly, and we manually analyzed the add-ons to see whether they were vulnerable to our XSS attacks. Among the 300 analyzed add-ons, we found 28 vulnerable ones (around 9%). Some of the vulnerable add-ons had nearly 10 million users. We have disclosed all the vulnerabilities to the corresponding developers (their contact information was available on the marketplaces).

We observed that the vulnerability rate in the popular add-ons was lower than in the randomly-selected ones in all the three marketplaces. The reason could be that the popular add-ons are more likely to be written by experienced developers. Moreover, add-ons that were vulnerable to outside input were rare. Specifically, only one add-on in our study was vulnerable. This could be because the add-on developers are more familiar with threats from external input (i.e., emails in this case) than those from a shared workspace.

4.4 Discussion

Since cross-site scripting is a well-known vulnerability, prudent engineering practices have been developed to prevent such mistakes [77]. Nevertheless, the problem has not been completely solved. From our empirical analysis, we can see that vulnerable add-ons are not rare in the wild. Attackers always find new ways of bypassing the defenses, and the speed of software development makes it difficult for threat analysis and defenses to

stay up to date.

We have confirmed by experiments that the vulnerabilities described in this paper are real and exploitable. There are, however, some additional practical considerations which a real-world attacker would face. The attacker needs to know which vulnerable add-on the victim is using, and the victim has to enable the add-on for the shared document. Thus, a successful attack probably requires a vulnerable add-on that users regularly invoke on large classes of documents which they are reading or editing. Translator and writing-assistant add-ons could meet these criteria. Add-ons that fix problems in data, such as duplicate removers, could even be installed by the victim when they receive a document with the corresponding problem.

In addition to the XSS vulnerabilities in the add-ons, another serious issue discovered in this work is the way OAuth 2.0 tokens are used in the Picker API, and the powerful exploits that it enables for the XSS attacker. The Picker API documentation has educated developers to use a design pattern where the add-on server shares its OAuth 2.0 token with the client-side code. Even if the Picker API itself is replaced with a safer solution, this unsafe software pattern might persist among developers.

Overall, we hope that our work will attract more attention to the area of cloud-application add-ons because further work is clearly needed. There might be other attack vectors that allow the attacker to exploit non-malicious add-ons. Analyzing threats from malicious add-ons could also be an interesting area for future work.

5. Topology poisoning in software-defined networking

Software-defined networking (SDN) [55] is a relatively new networking paradigm. Unlike traditional networks where routers and other dedicated network devices control the network's behavior with distributed routing protocols (e.g., OSPF [68], BGP [85]), SDN separates the control plane of the network from the network devices and moves it to a centralized software-based controller. With this architecture, it provides a network-wide view and flexible programmability to network administrators, allowing them to design and control the network with their own applications and to respond quickly to changing business needs.

As for any new technology, it is essential to understand the security threats in SDN. In Publication V, we consider the network-wide visibility of SDN and investigate one particular class of attacks: *topology poisoning*. We analyze the topology poisoning attacks against SDN both qualitatively to understand the principles, and quantitatively with simulations and abstract network structures to assess and explain their impact. Our focus is specifically on attacks mounted from compromised SDN switches.

In this chapter, we first give an overview of SDN and how topology discovery is performed. We then explain topology poisoning attacks and present the variants that we found. Finally, we describe the simulations and present the main results and insights.

5.1 Background

This section gives an overview of SDN and its topology discovery process as well as related work.

5.1.1 Software-defined networking

The principal idea of SDN is to decouple the control plane and the data plane of network devices. This is achieved by removing the control-plane logic from the routers and moving it to a logically centralized controller.

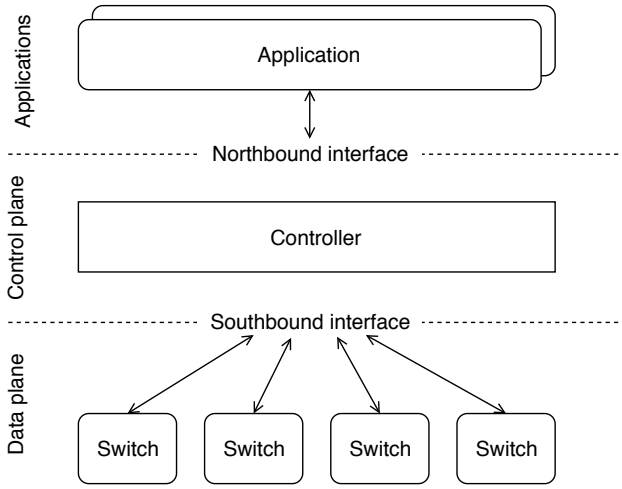


Figure 5.1. SDN architecture

Figure 5.1 shows a high-level overview of the SDN architecture. The SDN applications, such as network monitors and load balancers, make decisions based on the abstracted view of the network provided by the controller and specify their network requirements towards the controller via the northbound interface. The SDN controller, besides providing relevant information to the SDN applications, translates the network requirements to configuration commands and sends them to the physical network. (While the SDN controller is defined as a single logically centralized entity, it can be physically distributed to improve its performance and reliability.) Instead of routers and switches, all SDN network devices are called *switches*, although they forward packets based on both layer-2 and layer-3 headers. The controller exchanges control messages with the switches via the southbound interface using standardized protocols, such as Forwarding and Control Element Separation (ForCES) [30, 107], SoftRouter [58], and OpenFlow [62].

Topology discovery with OpenFlow. To make routing decisions, the SDN controller needs to know the network topology, i.e., how the switches are connected to each other. In this work, we used *OpenFlow* as the example controller-channel protocol and architecture because of its popularity.

In an OpenFlow software-defined network, the Link Layer Discovery Protocol (LLDP) [5] is used to dynamically detect layer-2 links between adjacent OpenFlow switches. Figure 5.2 illustrates how the controller detects a link between two switches. First, the controller sends to one switch an OFPT_PACKET_OUT message, which contains an LLDP packet and the instruction to forward it on a specific port. The switch forwards the LLDP packet on the specified port. If there is another switch connected to the port, it receives the LLDP packet and sends it to the controller in an OFPT_PACKET_IN message along with the ingress-port identifier. The

controller can thus reason that there is a directional link between the two switches. The controller then proceeds to do the same in the other direction.

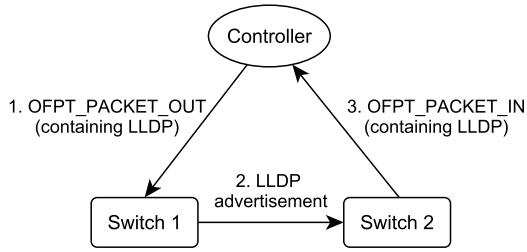


Figure 5.2. Switch-to-switch link discovery in OpenFlow networks

5.1.2 Related work

The earlier literature on SDN security has focused on the security of SDN applications [84, 91, 103], real-time verification of network policies [39, 52, 53, 83], and vulnerabilities in the controllers [56, 57, 87] as well as controller-switch communication [20, 28]. Much less effort has been put into studying attacks originating from the data plane and, more specifically, from compromised hosts and switches [9, 16, 48].

Topology poisoning in SDN has stimulated some interest among scholars. Hong et al. [43] presented several ways in which the controller's network view can be poisoned by a compromised host even when the controller channel is protected with TLS. The proposed attacks are carried out either by creating fake Link Layer Discovery Protocol (LLDP) packets or by forwarding genuine ones from one switch to another. We build on this basic technique while discussing a broader range of attack variants and analyzing them in more depth.

5.2 Topology poisoning attacks

This section presents our threat model and different variants of topology poisoning attacks in SDN.

Threat model. The attacker that we consider is in control of a small number of *compromised switches*. By compromised switches, we mean that the attacker is able to access and manipulate their configuration, authentication credentials, and flow tables. For example, the switches could be compromised by attacking them via a console serial port or by exploiting a software vulnerability. The compromised switches are not assumed to have any special hardware capabilities.

The goal of the attacker is to route more traffic to the compromised

nodes by spoofing fake links that create shorter or alternative paths in the network. Since all routing algorithms favor shorter paths at least to some extent, the SDN controller may route packets via these fake links. The attacker can then sniff the packets or mount MitM attacks on the data plane.

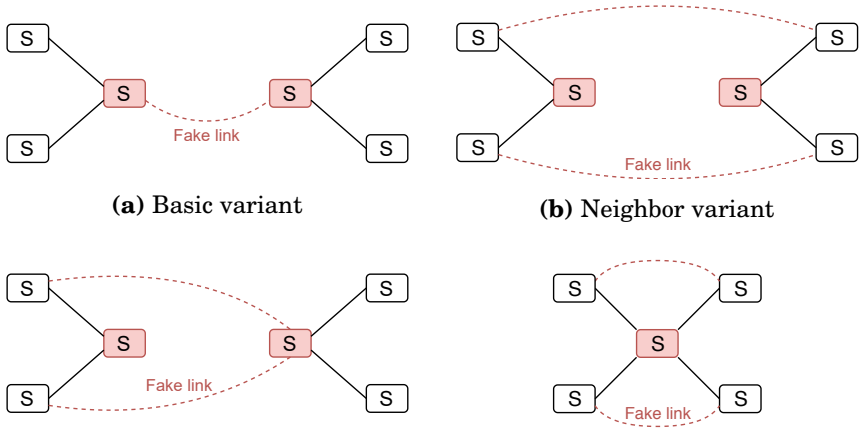
Attacks mounted by a small number of compromised nodes in an otherwise secure network are a timely issue for two reasons. First, in targeted attacks against high-security organizations, the attacker first tries to gain any kind of small foothold, such as one compromised network device, and then starts to extend this gradually. Second, the trustworthiness of network equipment is often called into question. This concern becomes even more acute with SDN because one of its promises is that networks can be built with low-cost, heterogeneous equipment.

Attack principle. The opportunities for topology poisoning in SDN are narrower than in traditional networks [45, 67, 78, 99]. In a well-managed SDN network, there is a direct secure channel between the non-compromised switches and the controller. Also, the switches do not aggregate routes or forward unauthenticated topology or routing information from each other. Thus, the compromised switches cannot spoof or make malicious modifications to the routes or to the forwarded topology data. This means that the only way for the compromised switches to influence routing in SDN is to *tweak the topology-discovery process locally between the compromised switches and their neighbors*.

The attacker first needs to manipulate the propagation of the LLDP packets to fabricate non-existing links and then establishes tunnels that make these links appear functional. For example, if there are two compromised switches, whenever one of them receives an LLDP packet that should cross the fake link, it tunnels the packet to the other switch, which encapsulates the packet and sends it to the controller. As the result, the controller thinks that there is a direct link between the two switches. The attacker, however, needs to avoid creating a forwarding loop. The issue is familiar from any type of tunneling in traditional networks: the tunnel packets must not themselves be routed into the tunnel because that would result in a loop. To avoid this pitfall, the attacker needs a *relay node* via which the tunnel endpoints communicate.

Attack variants. We found four different variants of topology poisoning in SDN, as illustrated in Figure 5.3. Among these, only the basic variant has been discussed in the literature [9, 28].

- *Basic variant*: The controller is fooled into thinking that there is a direct link between two compromised switches. This is the most obvious variant of the attack.



(a) Basic variant

(b) Neighbor variant

(c) Merging variant

(d) Single-switch variant

Figure 5.3. Four variants of topology poisoning in SDN

- *Neighbor variant*: This variant also requires two compromised switches, but the fake link is created between two good switches that are neighbors to the two compromised switches. The compromised switches typically have many neighbors, and it is possible to pair them up to create multiple fake links.

- *Merging variant*: In this variant, two or more compromised switches merge together and appear as one big virtual switch with many neighbors. They all share one identity, which can be taken from one of the participating switches.

- *Single-switch variant*: The last variant differs from the previous ones in that only one compromised switch is needed. The attacker configures the compromised switch to forward the LLDP packets received from one port directly to another port. Thus, fake links are established between neighbors of the compromised switch.

The above attack variants were tested in an emulated OpenFlow network environment using the Mininet emulator [41]. The principles of the attacks, however, are general and can be applied to most SDN technologies.

5.3 Attack simulation

Previous research has found topology poisoning attacks in SDN are particularly difficult to defend against, especially those based on LLDP relay/-tunneling [8, 28, 43]. Thus, we decided to simulate the topology poisoning attacks that are described in the previous section against a wide variety

Table 5.1. Network topologies in the simulations

Topology	Type	Edges
Grid	regular mesh	yes
2D torus	regular mesh	no
3D torus	regular mesh	no
Hypercube	regular mesh	no
Triangulated planar	irregular mesh	yes
Ad-hoc radio	irregular mesh	yes
Fat tree	tree-like	yes
Binary tree	tree-like	yes

of network topologies and routing strategies and analyze their threats. The goal was to gain insights into effective attack strategies and also into factors that mitigate their effect.

In this section, we give an overview of our simulation setup and the main observations from the simulation. The details can be found in Publication V.

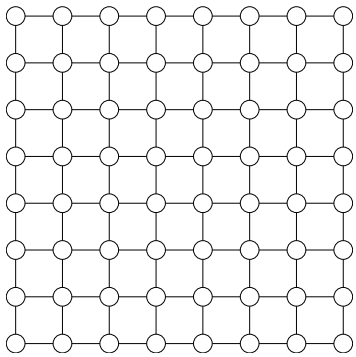
5.3.1 Simulation setup

In the simulations, we tested two routing strategies:

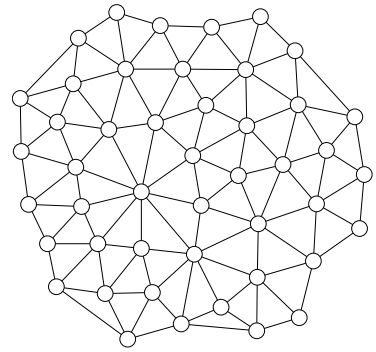
- *Fully-deterministic routing*: The Dijkstra algorithm [29] was used to find the path with the smallest number of hops.

- *Load-balancing routing*: This strategy also used the Dijkstra algorithm but with the total number of active flows on the links in the considered path as the primary distance measure. It was non-deterministic in the sense that the route for a new flow depended on the other flows in the network and, thus, could not be predicted reliably.

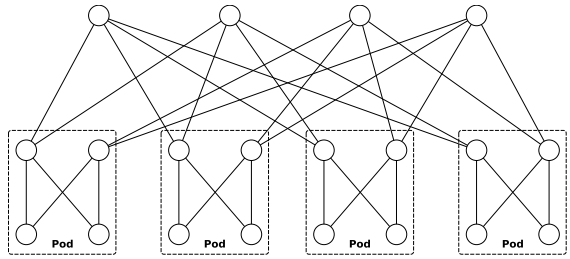
We considered a broad range of network topologies with different characteristics, which are shown in Table 5.1. They were categorized into three types: regular mesh, irregular mesh, and tree-like. The regular mesh topologies have been widely used for high-performance computing applications [23, 32]. The triangulated planar topology, which is a Delaunay triangulation [27] of a set of points in a plane, and the ad-hoc radio topology, where geographically placed nodes communicate with those within the radio range, represent wireless mesh networks. The fat-tree topology is popular in data centers because of its high fault tolerance. Simple tree topology can also be deployed in small-scale networks. After experimenting with different node degrees, we chose binary trees for the simulations because they exhibit the properties of tree topologies at their most extreme.



(a) Grid topology



(b) Triangulated planar topology



(c) Fat tree topology

Figure 5.4. Topology examples

Figure 5.4 illustrates an example of each type of the topologies.

For each simulation, we picked one, two, or many compromised switches in random and measured the number of data flows that passed through them. This metric was chosen because it reflects how many flows are vulnerable to follow-up attacks (e.g., sniffing, man-in-the-middle) by the compromised switches. We first simulated the baseline case, i.e., with no topology poisoning. We then repeated the experiment for each of the attack variants described in Section 5.2 to see how many more data flows were captured by the compromised switches.

5.3.2 Results

Our simulations indicated that topology poisoning clearly increases the attacker's ability to capture traffic flows. The neighbor and merging variants of the attack were significantly more successful than the basic variant. Even the single-switch attack could divert considerably more flows than the baseline.

We also observed that there was big variation in the results between simulations. Thus, we analyzed the factors that influenced the attacker's success. One key result was that, in most cases, load balancing and non-determinism in the routing reduced the effects of the attacks. This can be explained by the fact that, if a fake link is able to capture many traffic

flows, the controller soon thinks that the link is congested and routes the new flows around it. In general, any randomness or state-dependency in the routing means that the shortest path by hop count is not always used, and that reduces the attacker's ability to attract large numbers of flows to a small number of compromised switches. However, the load-balancing was less effective in mitigating the neighbor and merging variant attacks because the attacks create multiple fake links, which have more apparent capacity than the single fake link in the basic variant.

Irregularity in the network topology also helped reduce the impact of topology poisoning. This tendency can be explained by the fact that a regular network structure has many equal-length shortest paths between two endpoints, and one small shortcut may be able to divert them all. The phenomenon is familiar to cities with a rectangular grid plan (Figure 5.4a). If even just one of the rectangles is a park that can be crossed diagonally, that park attracts a large number of people walking through — and not just for the birds and trees but because it shortens many paths.

Another factor that affected the outcome of the attacks was the location of the compromised switches. It is obvious that central network locations are best for passive sniffing if the network has a center and an edge. For example, the grid, triangulated planar, and tree topologies have a center and edge, while the 2D torus, 3D torus, and hypercube do not. However, for topology spoofing, we observed that the central network locations may not be as critical because the fake links attract more traffic when their endpoints are more distant from each other. The longer jump a fake link makes, the bigger its effect on the apparent path lengths. This observation was clear in networks with no center and edge.

5.4 Discussion

Our work shows that topology poisoning attacks remain a serious threat in SDN. These attack techniques are attractive to an attacker who is looking for any way to expand a small foothold on a network. Also, an attacker that controls a significant fraction of the network devices can use them selectively to gain effective control of most data-plane traffic.

The attacks were tested against OpenFlow, but the principles are more general. First, the link discovery with LLDP is not specific to OpenFlow. It and similar proprietary protocols are widely used for topology discovery. Second, our simulations were done on a high level of abstraction that does not depend on the exact control protocol or method of link discovery, so the result can be generalized to most SDN technologies.

Furthermore, the observations from the simulation results can help set up the network in such a way that the impact of topology poisoning attacks is reduced. For example, since load balancing and non-determinism in the

routing strategy mitigated the attacks, random route mutation [31], which is a technique where routes between endpoints are periodically randomized, could be a very effective countermeasure. Also, a small amounts of random variation, such as shortcut links, could be artificially added to the network to increase the irregularity of the network topology and counter the effect of topology poisoning.

6. Conclusion

In this dissertation, we present various vulnerabilities in a wide range of security-critical applications and we analyze the potential solutions of each type of the vulnerabilities. Publication I presents the novel Man-in-the-Machine (MitMa) adversary model, in which the attacks are performed by nonprivileged users or processes against inter-process communication (IPC) between software components on the same computer. We demonstrate that the model has been overlooked in various vulnerable password managers and security tokens. Publication II is incremental work that builds on the research in Publication I. It shows that the current authentication mechanisms that are used by popular desktop cryptocurrency wallet applications are not effective against the MitMa adversary model. Publication III presents a large number of vulnerabilities in commercial VPN services. These vulnerabilities are in the configuration of the VPN clients and allow attackers to bypass the protection of the VPN tunnel and intercept network traffic to and from the victim. Publication IV considers cloud-application add-ons, a relatively new phenomenon in modern software, and how they bring unwanted security vulnerabilities to the applications. Publication V focuses on software-defined networking, an emerging network paradigm, and investigates a particular class of attack: topology poisoning.

The security of modern applications is a broad and ever-evolving topic and there are still many open and undiscovered problems in the area. While this dissertation is a step on the path towards more secure and resilient software, there is definitely a continuous demand for further research on security vulnerabilities in new software technologies.

References

[1] Dialogs and sidebars in G Suite documents. https://developers.google.com/apps-script/guides/dialogs. [Accessed Aug. 2019].

[2] Introduction to Cisco IPsec technology. https://www.cisco.com/c/en/us/td/docs/net_mgmt/vpn_solutions_center/2-0/ip_security/provisioning/guide/IPsecPG1.html. [Accessed Jan. 2019].

[3] Root and login sessions on OS X. https://developer.apple.com/library/content/documentation/MacOSX/Conceptual/BPMultipleUsers/Concepts/SystemContexts.html. [Accessed Oct. 2017].

[4] U2F technical overview. https://developers.yubico.com/U2F/Protocol_details/Overview.html. [Accessed Oct. 2017].

[5] IEEE standard for local and metropolitan area networks – Station and media access control connectivity discovery. *IEEE Std 802.1AB-2009*, 2009.

[6] Kernel local privilege escalation "Dirty COW" (CVE-2016-5195). https://access.redhat.com/security/cve/cve-2016-5195, 2016.

[7] Nasser Mohammed Al-Fannah. One leak will sink a ship: WebRTC IP address leaks. In *International Carnahan Conference on Security Technology (ICCST)*. IEEE, 2017.

[8] Talal Alharbi, Marius Portmann, and Farzaneh Pakzad. The (in)security of topology discovery in software defined networks. In *Conference on Local Computer Networks (LCN)*. IEEE, 2015.

[9] Markku Antikainen, Tuomas Aura, and Mikko Särelä. Spook in your network: Attacking an SDN with a compromised OpenFlow switch. In *Secure IT Systems (NordSec)*. Springer, 2014.

[10] Jacob Appelbaum, Marsh Ray, Karl Koscher, and Ian Finder. vpwns: Virtual pwned networks. In *USENIX Workshop on Free and Open Communications on the Internet*, 2012.

[11] Randall Atkinson and Stephen Kent. Security architecture for the Internet protocol. RFC 4301, 1998.

[12] Dor Azouri. Abusing text editors via third-party plugins. SafeBreach Labs research, 2018.

[13] Sruthi Bandhakavi, Samuel T King, Parthasarathy Madhusudan, and Marianne Winslett. VEX: Vetting browser extensions for security vulnerabilities. In *USENIX Security Symposium*, 2010.

[14] Adam Barth, Adrienne Porter Felt, Prateek Saxena, and Aaron Boodman. Protecting browsers from extension vulnerabilities. In *Network and Distributed System Security Symposium*, 2010.

[15] Anton Barua, Mohammad Zulkernine, and Komminist Weldemariam. Protecting web browser extensions from JavaScript injection attacks. In *International Conference on Engineering of Complex Computer Systems*. IEEE, 2013.

[16] Kevin Benton, L Jean Camp, and Chris Small. OpenFlow vulnerability assessment. In *ACM SIGSAC Workshop on Hot topics in software defined networking*. ACM, 2013.

[17] Daniel J Bernstein, Tanja Lange, and Ruben Niederhagen. Dual EC: A standardized back door. In *The New Codebreakers*. Springer, 2016.

[18] Prithvi Bisht and VN Venkatakrishnan. XSS-GUARD: precise dynamic prevention of cross-site scripting attacks. In *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*. Springer, 2008.

[19] Jesse Burns. Fuzzing Win32 inter-process communication mechanisms. In *Presentad in Black Hat 2006 Conference, Las Vegas, NV, USA*, 2006.

[20] Jiahao Cao, Qi Li, Renjie Xie, Kun Sun, Guofei Gu, Mingwei Xu, and Yuan Yang. The CrossPath attack: Disrupting the SDN control channel via shared links. In *USENIX Security Symposium*, 2019.

[21] Nicholas Carlini, Adrienne Porter Felt, and David Wagner. An evaluation of the Google Chrome extension security architecture. In *USENIX Security Symposium*, 2012.

[22] Stephen Checkoway, Ruben Niederhagen, Adam Everspaugh, Matthew Green, Tanja Lange, Thomas Ristenpart, Daniel J Bernstein, Jake Maskiewicz, Hovav Shacham, and Matthew Fredrikson. On the practical exploitability of dual EC in TLS implementations. In *USENIX Security Symposium*, 2014.

[23] Hyunseung Choo, Seong-Moo Yoo, and Hee Yong Youn. Processor scheduling and allocation for 3D torus multicomputer systems. *IEEE Transactions on Parallel and Distributed Systems*, 2000.

[24] Gil Cohen. Call the plumber – you have a leak in your (named) pipe. In *Presentad in DEF CON 25 (2017) Conference, Las Vegas, NV, USA*, 2017.

[25] Shaanan N Cohney, Matthew D Green, and Nadia Heninger. Practical state recovery attacks against legacy RNG implementations. In *ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2018.

[26] Mauro Conti, Nicola Dragoni, and Viktor Lesyk. A survey of man in the middle attacks. *IEEE Communications Surveys & Tutorials*, 2016.

[27] Boris Delaunay. Sur la sphere vide. *Izv. Akad. Nauk SSSR, Otdelenie Matematicheskii i Estestvennyka Nauk*, 1934.

[28] Mohan Dhawan, Rishabh Poddar, Kshiteej Mahajan, and Vijay Mann. Sphinx: Detecting security attacks in software-defined networks. In *Network and Distributed System Security Symposium*, 2015.

[29] Edsger W Dijkstra. A note on two problems in connexion with graphs. *Numerische mathematik*, 1959.

[30] Avri Doria, J Hadi Salim, Robert Haas, Horzmud Khosravi, Weiming Wang, Ligang Dong, Ram Gopal, and Joel Halpern. Forwarding and control element separation (ForCES) protocol specification. RFC 5810, 2010.

[31] Qi Duan, Ehab Al-Shaer, and Haadi Jafarian. Efficient random route mutation considering flow and network constraints. In *IEEE Conference on Communications and Network Security*. IEEE, 2013.

[32] Ralph Duncan. A survey of parallel computer architectures. *Computer*, 1990.

[33] Markus Feilner. *OpenVPN: Building and integrating virtual private networks*. Packt Publishing Ltd, 2006.

[34] Dennis Felsch, Martin Grothe, Jörg Schwenk, Adam Czubak, and Marcin Szymanek. The dangers of key reuse: Practical attacks on IPsec IKE. In *USENIX Security Symposium*, 2018.

[35] Adrienne Porter Felt, Helen J Wang, Alexander Moshchuk, Steve Hanna, and Erika Chin. Permission re-delegation: Attacks and defenses. In *USENIX Security Symposium*, volume 30, 2011.

[36] FIDO ALLIANCE. Universal 2nd factor (U2F) overview. `https://fidoalliance.org/specs/fido-u2f-v1.2-ps-20170411/fido-u2f-overview-v1.2-ps-20170411.html`. [Accessed Nov. 2017].

[37] Google. Extending Google workspace with add-ons. `https://developers.google.com/gsuite/add-ons/overview`. [Accessed Dec. 2020].

[38] Jeremiah Grossman, Seth Fogie, Robert Hansen, Anton Rager, and Petko D Petkov. *XSS attacks: cross site scripting exploits and defense*. Syngress, 2007.

[39] Arjun Guha, Mark Reitblatt, and Nate Foster. Machine-verified network controllers. In *ACM SIGPLAN Notices*. ACM, 2013.

[40] Kory Hamzeh, Grueep Pall, William Verthein, Jeff Taarud, W Little, and Glen Zorn. Point-to-point tunneling protocol (PPTP). RFC 2637, 1999.

[41] Nikhil Handigol, Brandon Heller, Vimalkumar Jeyakumar, Bob Lantz, and Nick McKeown. Reproducible network experiments using container-based emulation. In *International conference on Emerging networking experiments and technologies*. ACM, 2012.

[42] Stefan Heule, Devon Rifkin, Alejandro Russo, and Deian Stefan. The most dangerous code in the browser. In *Workshop on Hot Topics in Operating Systems*. USENIX, 2015.

[43] Sungmin Hong, Lei Xu, Haopei Wang, and Guofei Gu. Poisoning network visibility in software-defined networks: New attacks and countermeasures. In *Network and Distributed System Security Symposium*, 2015.

[44] Ryan Hurst and Ashwin Palekar. Microsoft EAP CHAP extensions. *IETF Draft*, 2007.

[45] Geoff Huston, Mattia Rossi, and Grenville Armitage. Securing BGP – a literature survey. *IEEE Communications Surveys Tutorials*, 2011.

[46] Omar Ismail, Masashi Etoh, Youki Kadobayashi, and Suguru Yamaguchi. A proposal and implementation of automatic detection/collection system for cross-site scripting vulnerability. In *International Conference on Advanced Information Networking and Applications*. IEEE, 2004.

[47] Nav Jagpal, Eric Dingle, Jean-Philippe Gravel, Panayiotis Mavrommatis, Niels Provos, Moheeb Abu Rajab, and Kurt Thomas. Trends and lessons from three years fighting malicious extensions. In *USENIX Security Symposium*, 2015.

[48] Samuel Jero, William Koch, Richard Skowyra, Hamed Okhravi, Cristina Nita-Rotaru, and David Bigelow. Identifier binding attacks and defenses in software-defined networks. In *USENIX Security Symposium*, 2017.

[49] Martin Johns, Björn Engelmann, and Joachim Posegga. XSSDS: Server-side detection of cross-site scripting attacks. In *2008 Annual Computer Security Applications Conference (ACSAC)*. IEEE, 2008.

[50] Leon Juranic. Back to the future: Unix wildcards gone wild. `http://www.defensecode.com/public/DefenseCode_Unix_WildCards_Gone_Wild.txt`. [Accessed Oct. 2017].

[51] Charlie Kaufman, Paul Hoffman, Yoav Nir, Parsi Eronen, and Tero Kivinen. Internet key exchange protocol version 2 (IKEv2). RFC 7296, 2014.

[52] Peyman Kazemian, Michael Chan, Hongyi Zeng, George Varghese, Nick McKeown, and Scott Whyte. Real time network policy checking using header space analysis. In *USENIX Symposium on Networked Systems Design and Implementation*. USENIX, 2013.

[53] Ahmed Khurshid, Wenxuan Zhou, Matthew Caesar, and P Godfrey. Veriflow: verifying network-wide invariants in real time. *ACM SIGCOMM Computer Communication Review*, 2012.

[54] Engin Kirda, Christopher Kruegel, Giovanni Vigna, and Nenad Jovanovic. Noxes: a client-side solution for mitigating cross-site scripting attacks. In *ACM symposium on Applied computing*. ACM, 2006.

[55] Teemu Koponen, Martin Casado, Natasha Gude, Jeremy Stribling, Leon Poutievski, Min Zhu, Rajiv Ramanathan, Yuichiro Iwata, Hiroaki Inoue, Takayuki Hama, et al. Onix: A distributed control platform for large-scale production networks. In *USENIX Symposium on Operating Systems Design and Implementation*. USENIX, 2010.

[56] Diego Kreutz, Fernando Ramos, and Paulo Verissimo. Towards secure and dependable software-defined networks. In *ACM SIGCOMM workshop on Hot topics in software defined networking*. ACM, 2013.

[57] Diego Kreutz, Fernando MV Ramos, Paulo Esteves Verissimo, Christian Esteve Rothenberg, Siamak Azodolmolky, and Steve Uhlig. Software-defined networking: A comprehensive survey. *Proceedings of the IEEE*, 2015.

[58] TV Lakshman, T Nandagopal, R Ramjee, K Sabnani, and T Woo. The SoftRouter architecture. In *ACM SIGCOMM Workshop on Hot Topics in Networking*. ACM, 2004.

[59] Sebastian Lekies, Ben Stock, and Martin Johns. 25 million flows later: Large-scale detection of DOM-based XSS. In *ACM SIGSAC conference on Computer & communications security*. ACM, 2013.

[60] Brian N Levine, Michael K Reiter, Chenxi Wang, and Matthew Wright. Timing attacks in low-latency mix systems. In *International Conference on Financial Cryptography*. Springer, 2004.

[61] Moxie Marlinspike and David Hulton. Divide and conquer: Cracking MS-CHAPv2 with a 100% success rate. `https://www.cloudcracker.com/blog/2012/07/29/cracking-ms`, 2012. [Accessed Jan. 2019].

[62] Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner. OpenFlow: enabling innovation in campus networks. *ACM SIGCOMM Computer Communication Review*, 2008.

[63] William Melicher, Anupam Das, Mahmood Sharif, Lujo Bauer, and Limin Jia. Riding out DOMsday: Towards detecting and preventing DOM cross-site scripting. In *NDSS*, 2018.

[64] Microsoft. Fast user switching. `https://msdn.microsoft.com/en-us/library/windows/desktop/bb776893`. [Accessed Feb. 2020].

[65] Microsoft. Secure Socket Tunneling Protocol (SSTP). `https://msdn.microsoft.com/en-us/library/cc247338.aspx`. [Accessed Jan. 2019].

[66] Microsoft. Three-tier application model. `https://docs.microsoft.com/en-us/previous-versions/office/developer/server-technologies/aa480455(v=msdn.10)`, 2006. [Accessed Feb. 2020].

[67] Alper Tugay Mizrak, Yu-Chung Cheng, Keith Marzullo, and Stefan Savage. Fatih: Detecting and isolating malicious routers. In *IEEE International Conference on Dependable Systems and Networks*. IEEE, 2005.

[68] John Moy. OSPF version 2. RFC 2328, 1997.

[69] Mozilla. window.postmessage() method. `https://developer.mozilla.org/en-US/docs/Web/API/Window/postMessage`. [Accessed Aug. 2019].

[70] Steven J Murdoch and Piotr Zieliński. Sampled traffic analysis by Internet-exchange-level adversaries. In *International workshop on privacy enhancing technologies*. Springer, 2007.

[71] Marius Musch, Marius Steffens, Sebastian Roth, Ben Stock, and Martin Johns. ScriptProtect: mitigating unsafe third-party javascript practices. In *ACM Asia Conference on Computer and Communications Security*. ACM, 2019.

[72] Ahamed Nafeez. Compression Oracle attacks on VPN networks. *Blackhat USA*, 2018.

[73] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system, 2008.

[74] Sam Newman. *Building microservices: Designing fine-grained systems*. O'Reilly Media, Inc., 2015.

[75] Aimee O'Driscoll. Does your VPN keep logs? 123 VPN logging policies revealed. `https://www.comparitech.com/vpn/vpn-logging-policies/`, 2019. [Accessed Dec. 2019].

[76] Alberto Ornaghi and Marco Valleri. Man in the middle attacks. In *Blackhat Conference Europe*, 2003.

[77] OWASP. XSS filter evasion cheat sheet. `https://www.owasp.org/index.php/XSS_Filter_Evasion_Cheat_Sheet`, 2019.

[78] Panagiotis Papadimitratos and Zygmunt J Haas. Secure link state routing for mobile ad hoc networks. In *Symposium on Applications and the Internet Workshops*. IEEE, 2003.

[79] Panagiotis Papadopoulos, Nicolas Kourtellis, and Evangelos P Markatos. Exclusive: How the (synced) cookie monster breached my encrypted VPN session. In *European Workshop on Systems Security*. ACM, 2018.

[80] R Pereira and S Beaulieu. Extended Authentication within ISAKMP/Oakley (XAUTH). *IETF Draft*, 1999.

[81] Perfect Privacy. IP leak affecting VPN providers with port forwarding. https://www.perfect-privacy.com/blog/2015/11/26/ip-leak-vulnerability-affecting-vpn-providers-with-port-forwarding/. [Accessed Jan. 2019].

[82] Vasile C Perta, Marco V Barbera, Gareth Tyson, Hamed Haddadi, and Alessandro Mei. A glance through the VPN looking glass: IPv6 leakage and DNS hijacking in commercial VPN clients. Sciendo, 2015.

[83] Philip Porras, Seungwon Shin, Vinod Yegneswaran, Martin Fong, Mabry Tyson, and Guofei Gu. A security enforcement kernel for OpenFlow networks. In *ACM Workshop on Hot topics in software defined networks*. ACM, 2012.

[84] Phillip Porras, Steven Cheung, Martin Fong, Keith Skinner, and Vinod Yegneswaran. Securing the software-defined network control layer. In *Network and Distributed System Security Symposium*, 2015.

[85] Yakov Rekhter and Tony Li. A border gateway protocol 4 (BGP-4). RFC 4271, 1995.

[86] Julian Reschke. The 'basic' HTTP authentication scheme. RFC 7617, 2015.

[87] Christian Röpke and Thorsten Holz. SDN rootkits: Subverting network operating systems of software-defined networks. In *International Symposium on Recent Advances in Intrusion Detection*. Springer, 2015.

[88] Bruce Schneier, Mudge, and David Wagner. Cryptanalysis of Microsoft's PPTP authentication extensions (MS-CHAPv2). In *International Exhibition and Congress on Network Security*. Springer, 1999.

[89] Andrei Serjantov and Peter Sewell. Passive attack analysis for connection-based anonymity systems. In *European Symposium on Research in Computer Security*. Springer, 2003.

[90] Yuru Shao, Jason Ott, Yunhan Jack Jia, Zhiyun Qian, and Z. Morley Mao. The misuse of Android Unix domain sockets and security implications. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, CCS '16, pages 80–91, New York, NY, USA, 2016. ACM.

[91] Seungwon Shin, Phillip A Porras, Vinod Yegneswaran, Martin W Fong, Guofei Gu, and Mabry Tyson. FRESCO: Modular composable security services for software-defined networks. In *Network and Distributed System Security Symposium*, 2013.

[92] Abraham Silberschatz, Peter Baer Galvin, and Greg Gagne. *Operating system concepts essentials*. John Wiley & Sons, Inc., 2014.

[93] Marius Steffens, Christian Rossow, Martin Johns, and Ben Stock. Don't trust the locals: Investigating the prevalence of persistent client-side cross-site scripting in the wild. In *NDSS*, 2019.

[94] Ben Stock, Martin Johns, Marius Steffens, and Michael Backes. How the web tangled itself: Uncovering the history of client-side web (in) security. In *USENIX Security Symposium*, 2017.

[95] Ben Stock, Stephan Pfistner, Bernd Kaiser, Sebastian Lekies, and Martin Johns. From facepalm to brain bender: Exploring client-side cross-site scripting. In *ACM SIGSAC conference on computer and communications security*. ACM, 2015.

[96] Mike Ter Louw and VN Venkatakrishnan. Blueprint: Robust prevention of cross-site scripting attacks for existing browsers. In *IEEE symposium on security and privacy*. IEEE, 2009.

[97] W Townsley, A Valencia, Allan Rubens, G Pall, Glen Zorn, and Bill Palter. Layer two tunneling protocol "L2TP". RFC 2661, 1999.

[98] Philipp Vogt, Florian Nentwich, Nenad Jovanovic, Engin Kirda, Christopher Kruegel, and Giovanni Vigna. Cross site scripting prevention with dynamic data tainting and static analysis. In *NDSS*, 2007.

[99] Feiyi Wang, Brian Vetter, and Shyhtsun Felix Wu. Secure routing protocols: Theory and practice. Technical report, North Carolina State University, 1997.

[100] Gary Wassermann and Zhendong Su. Static detection of cross-site scripting vulnerabilities. In *ACM/IEEE International Conference on Software Engineering*. IEEE, 2008.

[101] Blake Watts. Discovering and exploiting named pipe security flaws for fun and profit. `http://www.blakewatts.com/namedpipepaper.html`. [Dec. 2017].

[102] Michael Weissbacher, Enrico Mariconti, Guillermo Suarez-Tangil, Gianluca Stringhini, William Robertson, and Engin Kirda. Ex-ray: Detection of history-leaking browser extensions. In *Annual Computer Security Applications Conference*. ACM, 2017.

[103] Xitao Wen, Bo Yang, Yan Chen, Chengchen Hu, Yi Wang, Bin Liu, and Xiaolin Chen. SDNShield: Reconciliating configurable application permissions for SDN app markets. In *IEEE/IFIP international conference on dependable systems and networks*. IEEE, 2016.

[104] D. Wood, V. Stoss, L. Chan-Lizardo, G. S. Papacostas, and M. E. Stinson. Virtual private networks. In *International Conference on Private Switching Systems and Networks*, 1988.

[105] Gavin Wood. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum project yellow paper*, 2014.

[106] Luyi Xing, Xiaolong Bai, Tongxin Li, XiaoFeng Wang, Kai Chen, Xiaojing Liao, Shi-Min Hu, and Xinhui Han. Cracking app isolation on Apple: Unauthorized cross-app resource access on macOS. In *ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2015.

[107] Lily Yang, Ram Dantu, T Anderson, and Ram Gopal. Forwarding and control element separation (ForCES) framework. RFC 3746, 2004.

[108] Tatu Ylonen and Chris Lonvick. The secure shell (SSH) protocol architecture. RFC 4251, 2006.

[109] Michal Zalewski. *The Tangled Web: A Guide to Securing Modern Web Applications*. No Starch Press, 2012.

Aalto-DD 46/2021

ISBN 978-952-64-0326-7 (printed)
ISBN 978-952-64-0327-4 (pdf)
ISSN 1799-4934 (printed)
ISSN 1799-4942 (pdf)

Aalto University
School of Science
Department of Computer Science
www.aalto.fi

BUSINESS +
ECONOMY

ART +
DESIGN +
ARCHITECTURE

SCIENCE +
TECHNOLOGY

CROSSOVER

**DOCTORAL
DISSERTATIONS**