

# E-VOTING WEB APPLICATION

## SECURITY AUDIT REPORT

APRIL 2022

CLASSIFICATION	APPROVED FOR PUBLIC RELEASE
REFERENCE	P020456
CUSTOMER	Federal Chancellery
LAST MODIFIED	19/04/2022

## TABLE OF CONTENTS

<b>Executive summary</b> .....	<b>3</b>
Results summary .....	3
High level impressions .....	3
Security dashboard .....	4
Scope .....	4
Risks by level .....	4
Risks by remediation .....	4
Global risk level .....	4
Status by attacker profile .....	4
Identified risks .....	5
Proposed remediation plan .....	5
<b>Technical summary</b> .....	<b>6</b>
Scope .....	6
Restrictions .....	6
Schedule .....	6
Results .....	6
Vulnerability summary .....	8
<b>Detailed results</b> .....	<b>9</b>
Vulnerabilities and exploitation .....	9
P020456-1 Use of third-party libraries (supply chain attack) .....	9
<b>Complements</b> .....	<b>13</b>
Legend .....	13
SCRT Score .....	13
CVSS Score .....	13
Context .....	13
Additional attacks .....	14
Risk calculation .....	14
Attempted attacks .....	15
Attack scope .....	15
Search for known vulnerabilities (vulnerability scanning) .....	15
Network protocol analysis .....	16
Weak and default passwords discovery .....	16
Web applications .....	16
Network sniffing .....	17
Exploiting vulnerabilities .....	17

# EXECUTIVE SUMMARY

## RESULTS SUMMARY

The **Federal Chancellery (FCh)** contracted **SCRT SA** to perform a security audit of the E-voting system developed by **Swiss Post**.

During the audit, SCRT engineers mainly focused on the web voting platform and tried to identify vulnerabilities that would allow an attacker to compromise the integrity or the secrecy of a voter's choice. To perform the audit, the Swiss Post created a test election event on its infrastructure and provided SCRT with a set of voting cards. Besides, SCRT auditors had access to the source code of the voting system which is available through a public Gitlab repository.

The conducted test and the review of the code allowed SCRT engineers to identify a low-risk issue affecting the system. Indeed, both the client and server-side parts of the application rely on external dependencies. This introduces the risk of a "*supply chain attack*". Indeed, if one of these external projects were to be compromised, this could lead to the injection of malicious code in the E-voting project and thus to the confidentiality of the votes being compromised. It is not possible to prevent such attacks, but it is possible to detect them by performing in-depth code reviews of external dependencies before each build of the project that is meant to be deployed in production. SCRT did not verify this particular point during this intrusion test.

In conclusion, **the overall risk level is very low**. Nevertheless, the reported "*supply chain attack*" risk underlines the importance of in-depth source code review of every component used in the project.

## HIGH LEVEL IMPRESSIONS

### STRENGTHS

-  User input handling
-  Exposed attack surface
-  TLS configuration
-  HTTP security headers
-  Authentication
-  Protection against bruteforce attacks

### WEAKNESSES

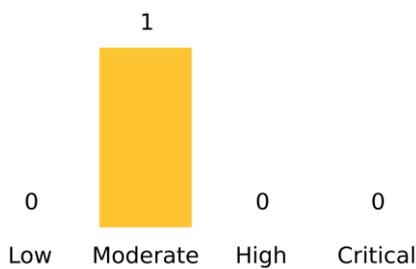
-  Use of third-party libraries

# SECURITY DASHBOARD

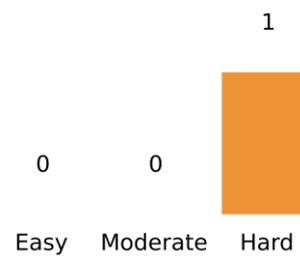
## SCOPE

Type	White-box pentest
Scope	E-voting web application
Positioning	SCRT Offices
Schedule	2021-11-26 – 2021-12-03
Effort	16 days
Consultants	3

## RISKS BY LEVEL



## RISKS BY REMEDIATION



## GLOBAL RISK LEVEL

ATTACKER PROFILES	RISK LEVEL
Unauthenticated attacker	<span style="color: green;">■</span> <span style="color: gray;">■</span> <span style="color: gray;">■</span> <span style="color: gray;">■</span>
Voter	<span style="color: green;">■</span> <span style="color: gray;">■</span> <span style="color: gray;">■</span> <span style="color: gray;">■</span>

## STATUS BY ATTACKER PROFILE

OBJECTIVES	UNAUTHENTICATED ATTACKER	VOTER
Access the administrator panel	✓	✓
Vote in place of legitimate users	✓	✓
Consult other users' votes	✓	✓
Execute arbitrary commands on a server	✓	✓
Tamper with a user's vote	✓	✓

✓ NOT COMPROMISED    
 ! PARTIALLY COMPROMISED    
 ! COMPROMISED

## IDENTIFIED RISKS

ID	RISK LEVEL	RISK DETAILS	RELATED FLAWS	FIX
1	<b>MODERATE</b>	An attacker could perform a "supply chain attack" and insert malicious code in a client-side or server-side library.	<b>P020456-1</b>	

## PROPOSED REMEDIATION PLAN

ID	ACTION	DIFFICULTY	RELATED RISKS
1	Review the source code of third-party libraries before building the project.	<b>HARD</b>	1

# TECHNICAL SUMMARY

---

## SCOPE

The scope of the audit was constituted by the following election event:

- » <https://pit.evoting.ch/vote/#/legal-terms/009e088e0b56405ab8c5644db2cfdb8e>

As the penetration test was performed as a white-box audit on an open-source project, the source code of the application was also available on GitLab: <https://gitlab.com/swisspost-evoting>.

Several voting cards were also provided for the audit to take place.

## RESTRICTIONS

No social engineering or denial of service attacks were performed during this audit.

## SCHEDULE

START DATE	END DATE
2021-11-26	2021-12-03
EFFORT	
14 days	

## RESULTS

The overall security level is **very high**. No vulnerabilities with a practical exploitation scenario could be found within the allotted time.

The engineers were provided with a set of voting cards and had access to the source code of the application (public GitLab repository). The main objectives were to verify whether an attacker could access a vote's information or tamper with a voter's choices without them noticing. However, Denial of Service (DoS) attacks were out of the scope.

The first and most obvious attack scenario considered by the engineers was *Man-in-the-Middle*. In other words, is there any way for an attacker to intercept the communication between a client and the server? It turns out the application is very well protected against this kind of attacks.

For instance, the **TLS configuration** follows all the best practices and is compatible with perfect forward secrecy. **HSTS** is also enforced with preloading, which means that the application is also protected against "*SSL stripping*" attacks. Finally, **DNSSEC** is configured, preventing a malicious DNS server on an untrusted network from redirecting the user to an arbitrary website.

That being said, the scenario in which a third-party CA certificate would have been added to the client's certificate store was also considered. However, the application also uses end-to-end encryption at the application layer, which means that even if an attacker could intercept the clear text data, he would not be able to get any exploitable information from it.

Apart from the *Man-in-the-Middle* attack scenario, the usual injection attacks were also attempted within the requests that are sent by the client to the remote application. Here again, the application is well protected as even a single character modification results in generic error messages (mostly 401, 404, 403 and 412 error codes) being returned by the server. Therefore, it was not possible to get any exploitable information from the remote application either.

After considering all the usual attacks against web applications, the engineers explored other less common avenues. As such, they analyzed the build process of the E-voting project and especially the way it handles third-party dependencies. Indeed, the *Secure Data Manager* relies on a few third-party open-source projects both for its client-side and server-side code bases. Although it is not a vulnerability per se, it does represent a security risk as this code should be considered potentially untrustworthy. A developer's account could be compromised, and malicious code could be pushed to one of the third-party project repositories in a so-called "*supply chain attack*". Unfortunately, there is no technical solution that can prevent this type of attack. Therefore, the only way to prevent it would be to perform a manual code review of each external library that is imported in the project.

In conclusion, no particular action needs to be taken in the short-term. However, a particular attention should be paid to the third-party libraries being used as they might represent a threat on the long-term.

## VULNERABILITY SUMMARY

ID	VULNERABILITY	IMPACT	PROBABILITY	CVSS
P020456-1	Use of third-party libraries (supply chain attack)	★★★★☆	★☆☆☆☆	7.7

Explanations regarding impact, exploitation and CVSS scores can be found in chapter *Complements*

# DETAILED RESULTS

## VULNERABILITIES AND EXPLOITATION

P020456-1 USE OF THIRD-PARTY LIBRARIES (SUPPLY CHAIN ATTACK)			
SCRT		CVSS	
Impact	★★★★☆	Base	7.7
Probability	★☆☆☆☆	AV:N/AC:H/PR:H/UI:N/S:C/C:H/I:H/A:N	
PREREQUISITES		COMPROMISED ASSETS	
» Malicious content inserted in a third-party library		» Vote confidentiality	

### AFFECTED SYSTEMS

E-voting project

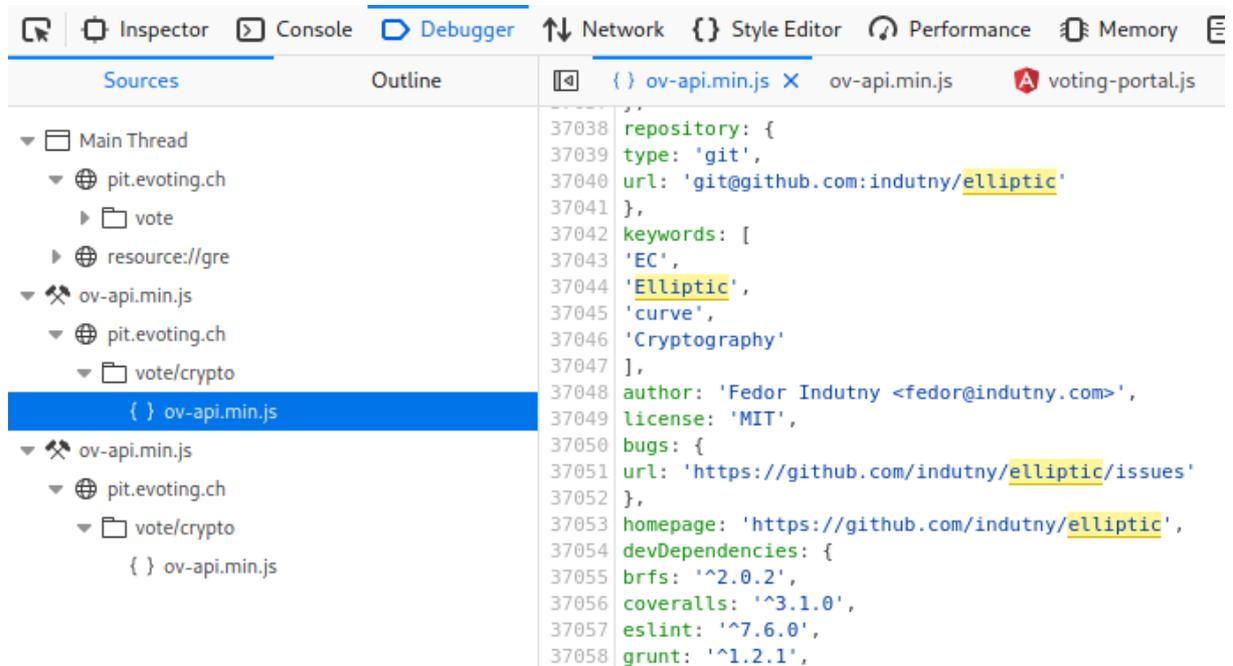
### DESCRIPTION

The application relies on third-party libraries, either for its server-side or its client-side code base. This is a very common situation as it avoids reinventing the wheel whenever a reliable package can just be imported into the project instead.

However, such a practice introduces new risks, especially for highly sensitive applications, as one of the third-party packages could be compromised. This scenario is commonly referred to as a "supply chain attack". As we have seen in the past, this may happen if a developer's account on a code repository is compromised, and malicious code is inserted. In addition, code integrity checks would fail to detect such a malicious modification as the code would be modified directly at the source, prior to the deployment of the package.

## EXPLOITATION

By inspecting the client-side JavaScript code of the application, we observe that it includes a third-party project called "elliptic".



```

37038 repository: {
37039   type: 'git',
37040   url: 'git@github.com:indutny/elliptic'
37041 },
37042 keywords: [
37043   'EC',
37044   'Elliptic',
37045   'curve',
37046   'Cryptography'
37047 ],
37048 author: 'Fedor Indutny <fedor@indutny.com>',
37049 license: 'MIT',
37050 bugs: {
37051   url: 'https://github.com/indutny/elliptic/issues'
37052 },
37053 homepage: 'https://github.com/indutny/elliptic',
37054 devDependencies: {
37055   brfs: '^2.0.2',
37056   coveralls: '^3.1.0',
37057   eslint: '^7.6.0',
37058   grunt: '^1.2.1',

```

Third-party Elliptic JS library

As the description of the project states, it is an implementation of "fast elliptic-curve cryptography in a plain JavaScript". This project is hosted on GitHub, and it is available at the following URL: <https://github.com/indutny/elliptic>.

This is a very popular project as it is used by almost 7 million users and maintained by 25 contributors, which means that the probability that malicious code could be pushed on the main branch without anybody noticing is low.



"Used by" and "contributors"

However, this project also has its own third-party dependencies, which increase the attack surface: <https://github.com/indutny/elliptic/blob/master/package.json>.

```

46     },
47     "dependencies": {
48       "bn.js": "^4.11.9",
49       "brorand": "^1.1.0",
50       "hash.js": "^1.0.0",
51       "hmac-drbg": "^1.0.1",
52       "inherits": "^2.0.4",
53       "minimalistic-assert": "^1.0.1",
54       "minimalistic-crypto-utils": "^1.0.1"
55     }
56   }

```

*Elliptic dependencies*

If an attacker were to compromise a developer's account, he could be able to insert malicious JavaScript code that would be executed on client-side. This JavaScript code could get any information about a voter's choices or even tamper with a vote directly.

Regarding server-side dependencies, the same concepts apply. In particular, every external dependency retrieved with *Maven* and listed in the multiple `pom.xml` files might be subject to similar *supply chain attacks* which could help an attacker to compromise the server and thus, the privacy and integrity of the vote.

```

[..snip..]
<!-- Bouncy Castle -->
<dependency>
  <groupId>org.bouncycastle</groupId>
  <artifactId>bcprov-jdk15on</artifactId>
</dependency>
<dependency>
  <groupId>org.bouncycastle</groupId>
  <artifactId>bcpkix-jdk15on</artifactId>
</dependency>

<!-- jackson for json processing (this is used for annotations in DTOs) -->
<dependency>
  <groupId>com.fasterxml.jackson.core</groupId>
  <artifactId>jackson-annotations</artifactId>
</dependency>
<dependency>
  <groupId>com.fasterxml.jackson.core</groupId>
  <artifactId>jackson-databind</artifactId>
</dependency>

<dependency>
  <groupId>org.apache.xmlgraphics</groupId>
  <artifactId>batik-css</artifactId>
  <version>${org.apache.xmlgraphics.version}</version>
</dependency>
[..snip..]

```

However, there is one slight difference though as the server-side code also relies on a third-party library which is not available through a well-known public repository. Indeed, while building the project, we noticed that the *Secure Data Manager* requires the "*PKCS11wrapper*" library for interacting with smart cards as explained in the [documentation](#).

### Install Manual Third-Party Dependencies

The Secure Data Manager requires a PKCS11wrapper, developed by IAIK of Graz University of Technology, for interacting with SmartCards. The PKCS11 wrapper is not available in public repositories but can be obtained for free from IAIK's website under an Apache-style license (you can find the license in the Secure Data Manager directory).

Therefore, you have to obtain yourself a copy of the `iaikPkcs11Wrapper` <https://jce.iaik.tugraz.at/products/core-crypto-toolkits/pkcs11-wrapper/> and install it as a maven artifact using the following command:

```
mvn install:install-file -Dfile=iaikPkcs11Wrapper_1.6.2.jar -DgroupId=iaik -DartifactId=iaikPkcs11Wrapper -Dvers
```

*PKCS11wrapper*

Although the security of public repository platforms such as GitHub is highly monitored, a third-party website such as <https://jce.iaik.tugraz.at> might not be as secure.

### POSSIBLE SOLUTIONS

#### Review the source code of third-party scripts and libraries

The use of Subresource Integrity is efficient only in the case where JavaScript code is included from a third-party domain. It does not protect against malicious modifications of the code in the code repository itself. In this particular case, the only solution is to perform a code review to ensure that no malicious code was inserted. Another review should be performed every time the code is updated to a new version. It should also be noted that SRI does not protect against *Man-in-the-Middle* attacks as the `integrity` value can be updated after modifying the code. Indeed, this value is just a hash of the code, not a cryptographic signature.

### REFERENCES

- » [https://developer.mozilla.org/en-US/docs/Web/Security/Subresource\\_Integrity](https://developer.mozilla.org/en-US/docs/Web/Security/Subresource_Integrity)

# COMPLEMENTS

## LEGEND

### SCRT SCORE

For each vulnerability discovered and detailed in this report, SCRT provides a threat estimation. This estimation is done according to two indicators: Impact and Probability.

IMPACT	IMPACT OF THE VULNERABILITY IN CASE OF SUCCESSFUL EXPLOITATION ("HOW BAD?")				
☆☆☆☆	★☆☆☆	★★☆☆	★★★☆☆	★★★★☆	★★★★★
N/A	Weak	Medium	High	Critical	
PROBABILITY	PROBABILITY THAT THE VULNERABILITY WILL BE DISCOVERED AND EXPLOITED BY AN ATTACKER?				
☆☆☆☆	★☆☆☆	★★☆☆	★★★☆☆	★★★★☆	★★★★★
N/A	Low	Medium	High	Very high	

It is however important to keep in mind that this evaluation is only based on information available to SCRT engineers at the time of the audit. The auditors do not necessarily know all the details about vulnerable machines or systems. Consequently these ratings have to be reconsidered by depending on the importance and exact characteristics of affected systems.

### CVSS SCORE

On top of the SCRT score, an other metric is calculated for each vulnerability using the CVSS system.

CVSS is a vulnerability scoring system designed to provide an open and standardized method for rating IT vulnerabilities. CVSS helps organizations prioritize and coordinate a joint response to security vulnerabilities by communicating the base, temporal and environmental properties of a vulnerability. More information about the CVSS scoring system can be found here : <https://www.first.org/cvss/user-guide>

## CONTEXT

The context of each vulnerability is presented by describing its prerequisites and compromised assets. The prerequisites detail what is required by an attacker to be able to exploit the flaw, such as the exploitation of a previous vulnerability or the use of social engineering. The compromised assets list the assets that are directly impacted by the exploitation of the vulnerability.

## ADDITIONAL ATTACKS

The following attacks are not usually performed during penetration tests, as their success is greatly dependant on a variety of external factors, which cannot be controlled during the tests. However, certain discovered vulnerabilities may depend on the successful exploitation of such an attack, which is why they are described here.

### MAN-IN-THE-MIDDLE

A Man-In-The-Middle attack refers to a situation where the attacker is able to eavesdrop and alter the data transmitted between the client and the server, without any of them being able to notice the modification. An adversary can undertake this attack only if he has access to specific locations on the network. Effective attacks can be launched from the local network (for example *ARP Spoofing* or *DNS Poisoning*). Additionally, any node of the network through which the client-server communication flows can be used to undertake a Man-In-The-Middle attack. ISPs as well as governments are therefore often considered as having the possibility (legitimately or not) to undertake these kinds of attacks.

### SOCIAL ENGINEERING

Users are frequently one of the attacker's primary target. Sophisticated attacks (*phishing, phoning, ...*) are often developed in order to manipulate victims. When stated as a prerequisite to a vulnerability, social engineering means that an attacker must have some kind of contact with his victim in order to lure him into performing an action desired by the attacker, such as clicking on a link or opening a file attached to an e-mail.

## RISK CALCULATION

Each risk presented in this report is based on the impact and probability of exploitation (estimated by SCRT) of one or several vulnerabilities. The risk level is calculated by using the following table for the most severe vulnerability related to the risk.

Overall Risk Severity					
Impact	CRITICAL	High	High	Critical	Critical
	HIGH	Moderate	Moderate	High	Critical
	MODERATE	Low	Moderate	Moderate	High
	LOW	Low	Low	Moderate	High
		LOW	MODERATE	HIGH	CRITICAL
Probability					

For more information on the impact and probability of exploitation of each risk, please refer to the technical details of the corresponding vulnerability.

SCRT also provides an estimation of the effort required to mitigate the various risks. This is an estimate based on SCRT's experience and can obviously be different within the specific context of a given company.

## **ATTEMPTED ATTACKS**

### **ATTACK SCOPE**

The attacks performed by SCRT engineers during this audit cover the spectrum of attacks that could be attempted by an actual attacker against the targeted information system. These attacks thus cover "system" aspects (focused on machines and operating systems) as well as "applicative" aspects (focused on applications running on top of the system).

As an example of this layered attack approach, consider a (poorly coded) web application vulnerable to SQL injection, deployed on a correctly configured and patched web server. The "system" components of this application (the OS, the web server, DB engine...) do not suffer from any known vulnerability. However the "applicative" layer is flawed and thus compromises the security of the whole system.

### **SEARCH FOR KNOWN VULNERABILITIES (VULNERABILITY SCANNING)**

Software development is a complex task, especially when developing very large applications such as operating systems, and often requires scores of developers in different teams working autonomously. It is therefore not surprising that these applications contain many hidden bugs and vulnerabilities (often due to development errors), even after they are put on the market.

These flaws, when they are then discovered – by security researchers for example or by the companies themselves – are then often published to inform end-users and push developers to correct them. Many flaws are discovered and published daily, which are then generally followed by the release of a new patch for the affected piece of software.

However, these publications do not only interest the developers trying to correct the flaws. They are also very interesting for hackers as they reveal vulnerable pieces of code in the software. Sometimes these flaws allow hackers to gain remote access on a machine. In parallel with the release of new patches, specialized web sites often release exploit code for these same vulnerabilities. These are small programs which exploit the vulnerability and are often very easy to use. This makes it very important to apply patches as quickly as possible. Not doing so leaves the door open to malicious hackers who may exploit the vulnerabilities to gain access to the affected machine.

System administrators must therefore take extreme care in making sure that all systems are up to date and that the accessible services are not prone to known vulnerabilities. This is a constantly ongoing job as a seemingly secure machine one day may suddenly become the target of attacks the next after the publication of a new vulnerability affecting it.

To check whether any of the systems within the scope are vulnerable to known vulnerabilities, SCRT engineers will research information based on the reported versions of software discovered previously.

This is partly done with the help of automated scanners whose main goal is precisely the discovery of known vulnerabilities. However, a vulnerability scan is only a small part of a security audit and – on its own – cannot substitute a manual audit.

## **NETWORK PROTOCOL ANALYSIS**

Multiple services use clear-text protocols to communicate. This means that data is not encrypted before being sent on the network, sometimes even while sending credentials. In this context it is often possible for an attacker to sniff network traffic in hope of discovering clear-text user names and passwords.

This is also true for many web applications that do not use HTTPS, or do not implement it in a secure way, even when they deal with sensitive information.

The level of security applied to the communications of a given service is therefore an important part of its security and must also be subjected to analysis.

## **WEAK AND DEFAULT PASSWORDS DISCOVERY**

Many services used on a network are protected by a password. These can be remote access services such as SSH, FTP or private sections of a web site, for example, an administration panel.

In most cases, access to these secure areas will allow an attacker to gain access to sensitive or confidential information and in some cases compromise the machine entirely. For this reason it is important that the passwords be secure enough to stop an attacker from gaining illicit access. Indeed, however secure an application may be, if a user or administrator decides to use a weak password that can easily be guessed by an attacker, the security level cannot be guaranteed. It is extremely important that chosen passwords are not part of any dictionary, as they are often used by attackers in an automated way to gain access to a service.

To check the security level of the passwords, SCRT engineers test default and weak passwords on any service requiring authentication.

## **WEB APPLICATIONS**

There are many different ways web applications may be attacked. New types of attacks are regularly discovered allowing attacker to circumvent older security mechanisms, therefore forcing developers to constantly improve their code to prevent these new attacks.

There is however a repository of the most commonly discovered and exploited vulnerabilities in web applications. It is the Open Web Application Security Project's (OWASP1) TOP 10 which mentions the following vulnerabilities (OWASP Top Ten 2013):

- » Injection
- » Broken Authentication and Session Management
- » Cross-Site Scripting
- » Insecure Direct Object Reference

- » Security Misconfiguration
- » Sensitive Data Exposure
- » Missing Function Level Access Control
- » Cross-Site Request Forgery
- » Using Components with Known Vulnerabilities
- » Unvalidated Redirects and Forwards

Depending on the context of the application and underlying infrastructure, the relevant vulnerabilities will be tested. A couple of these most common flaws are detailed in the next chapters.

However, vulnerabilities are not limited to what is published in the OWASP Top 10 and SCRT engineers are more than capable of identifying flaws that are not necessarily well documented thanks to their experience gained from years of penetration testing.

## **NETWORK SNIFFING**

Within a local network, such as a corporate network, several different services are provided for the users, such as file sharing, FTP servers, remote administration and so on. Many of these services use clear-text protocols to communicate, meaning that data transiting on the network is not encrypted. In some cases, even the user's credentials are sent in this way.

It is therefore possible for a user located on this network to intercept the network traffic in order to gather credentials or confidential information. This is usually done with the help of an ARP poisoning attack, which allows an attacker to make a targeted user believe he is the user's gateway and make the gateway believe he is the end-user, which then leads to him proxying all requests between the two.

Clear-text credentials can easily be found this way, but in cases where authentication details are encrypted, the use of "cracking" tools comes in handy and will allow an attacker to break any potentially weak passwords.

## **EXPLOITING VULNERABILITIES**

One of the main differences between an intrusion test and a simple vulnerability scan, which is too often referred to in the same terms, is the fact that an intrusion test will truly simulate what an attacker may do when attacking a company.

Any vulnerability discovered during the audit are exploited by SCRT engineers as long as it is actually exploitable and in line with the rules of engagement determined during the kick-off.

This is the only way to know how dangerous the vulnerability truly is. It will allow one to know what kind of information an attacker may access by exploiting the flaw and whether he may leverage it to attack other systems.