EVOTING WEB APPLICATION (1.3.1.1)

SECURITY AUDIT REPORT

JULY 2023

CLASSIFICATION	PUBLIC
REFERENCE	P021121
PROJECT MANAGER	SCRT
CUSTOMER	Federal Chancellery
LAST MODIFIED	2023-08-07

SCRT Information Security

RUE DU SABLON 4
1110 MORGES
SWITZERLAND

TÉL : +41 21 804 64 01
FAX : +41 21 804 64 02

WWW.SCRT.CH
INFO@SCRT.CH

Client contact information

Federal Chancellery

SCRT SA contact information

SCRT SA
Rue du Sablon 4
1110 Morges
Suisse

Versions

DATE	VERSION	AUTHOR	DESCRIPTION
2023-07-21	0.1	SCRT	Report writing
2023-07-26	0.2	SCRT	Initial release
2023-08-07	1.0	SCRT	Final 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.

SCRT reviewed the source code and performed dynamic analysis of a production environment and a test platform. Very strict security rules applied by the Web Application Firewall on the production environment drastically reduce the attack possibilities. SCRT therefore also attempted attacks against a locally setup instance which was not protected by a Web Application Firewall.

Throughout the audit, SCRT was unable to affect the integrity or confidentiality of any vote within the e-voting servers. Some minor issues were discovered and documented in the Vulnerabilities and exploitation and Additional remarks sections of this report, but none of them were actually exploitable during this assessment.

The secrecy of the vote of a targeted individual could potentially still be compromised through spear-phishing attacks as detailed in the previous report. This issue has not been reproduced in the current report.

Overall, SCRT found that the application and its infrastructure are well hardened, with additional in-depth protections implemented between the users and the server. Hence, the overall risk level of using the E-voting web application is considered as low.

HIGH LEVEL IMPRESSIONS

	STRENGTHS
⊕	WAF configuration
⊕	Parameter filtering and validation
⊕	Limited attack surface

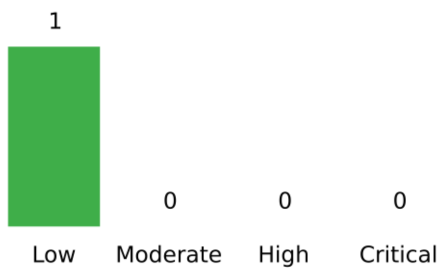
	WEAKNESSES
⊖	Outdated component

Security dashboard

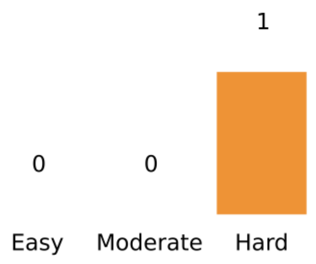
Scope

Type	White-box
Scope	Web application
Positioning	SCRT Offices
Schedule	2023-07-17 – 2023-07-21
Effort	15 days
Consultants	3

Risks by level



Risks by remediation



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
Gain access to the internal network	✓	✓	✓
Execute arbitrary commands	✓	✓	✓
Vote confidentiality and integrity	✓	✓	✓
Application infrastructure	✓	✓	✓

✓ NOT COMPROMISED ! PARTIALLY COMPROMISED ! COMPROMISED

IDENTIFIED RISKS

ID	RISK LEVEL	RISK DETAILS	RELATED FLAWS	FIX
1	LOW	The use of outdated packages in the build dependencies increases the likelihood of a vulnerability being exploitable, even though none currently are.	P021121-01	🔧

🔧 EASY	🔧 MEDIUM	🔧 HARD

PROPOSED REMEDIATION PLAN

ID	ACTION	DIFFICULTY	RELATED RISKS
1	Use the latest version of potentially vulnerable packages.	HARD	1

TECHNICAL SUMMARY

SCOPE

The scope of the audit includes the e-voting web application (release 1.3.1.1), which was reachable during the audit at the following address:

- `https://pit.evoting.ch/`

Auditors were able to generate as many voting cards as needed during the tests.

The source code is also available online:

- `https://gitlab.com/swisspost-evoting/e-voting/e-voting`

RESTRICTIONS

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

RESULTS

While the main goal was to attack the testing environment available at `https://pit.evoting.ch`, SCRT engineers first decided to locally deploy the application to gain full control over the server and to increase the probability of finding security bugs. This part of the audit allowed SCRT engineers to identify potential weaknesses in the building process which could lead under specific condition to the compromise of the solution.

Indeed, as detailed in the Additional remarks section of this report, an attacker who temporarily manages to control the DNS entries of the `post.ch` domain could potentially be able to compromise future builds of the project. The exploitation of this issue highly depends on the configuration of the building environment which is not directly part of the scope of this audit. As a result, SCRT engineers could not confirm the exploitability of the issue during the time frame of the tests. However, given the trendiness of this type of vulnerability at the time of writing the report, it was deemed worth mentioning.

In addition, the review of the building process and the project dependencies allowed engineers to detect that some of the dependencies embed vulnerable packages which could lead to denial of service or arbitrary code execution. However, those vulnerabilities require specific prerequisites, none of which were present in the test environment. Nevertheless, it is advised to determine whether vulnerable packages can be updated and update them accordingly when possible.

Then, SCRT engineers reviewed the source code of the different components both manually and using automated tools. In addition, they also performed several tests aimed at detecting potential weaknesses such as bugs in the logic of the application or potential injections which could endanger the integrity and the privacy of the vote. However, engineers were not able to identify vulnerabilities in the application during the allotted time of the audit.

Moreover, SCRT auditors noticed that the online application is further protected by a Web Application Firewall (WAF) which severely restricts the requests that can be made by external users to the E-Voting server and thus reduces the attack surface. The WAF appropriately whitelists both the accessible URLs and parameters which can be sent.

VULNERABILITY SUMMARY

ID	VULNERABILITY	IMPACT	PROBABILITY	CVSS
P021121-01	Use of outdated system or software	★☆☆☆	☆☆☆☆	3.7

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

ADDITIONAL REMARKS

REFLECTED HOST HEADER

It was noticed that if the X-Forwarded-For header was added to a valid request, a redirection was triggered because the request is detected as being suspicious. In this case, the Host header from the requests is then reflected in the response. Below an example of such behaviour:

Request:

```
POST /vs-ws-
rest/api/v1/processor/voting/authenticatevoter/electionevent/71E6A4D40CA7DADB12929FBC18D442FB/crede
ntialId/F455985AA0960BE2F4086CF53C3E6C89/authenticate?reflected2/ HTTP/1.1

Host: pit.evoting.ch.reflected1
X-Forwarded-For: whatever
[...]
```

Response:

```
HTTP/1.1 302 Found

[...]

<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html><head>
<title>302 Found</title>
</head><body>
<h1>Found</h1>
<p>The document has moved <a href="https://pit.evoting.ch.reflected1/errordocuments/suspicious-
connection.html?reflected2/">here</a>.</p>
</body></html>
```

Note that this only works if the term pit.evoting.ch is found somewhere within the header. Although there doesn't seem to be any way to exploit this in practice, it seems like some security measure is somewhat circumvented and that the regular expression checking the header can be improved.

DEPENDENCY CONFUSION

Dependency confusion is a trendy topic at the moment with multiple articles being written about it, mostly targeting the NPM repository. The idea behind the attack is to identify private repositories used by applications and then register them on a public repository. In certain cases, a build environment might end up pulling the code from the public repository rather than the local one.

The pom.xml file located in the directory evoting-dependencies refers to various internal packages which are not published on the maven central repository.

```
[...]
      <dependencyManagement>
              <dependencies>
                      <!-- Internal dependencies -->
                      <dependency>
                              <groupId>ch.post.it.evoting.domain</groupId>
                              <artifactId>domain</artifactId>
                              <version>${domain.version}</version>
                      </dependency>
                      <dependency>
                              <groupId>ch.post.it.evoting.commandmessaging</groupId>
                              <artifactId>command-messaging</artifactId>
                              <version>${command-messaging.version}</version>
                      </dependency>
                      <dependency>
                              <groupId>ch.post.it.evoting.domain</groupId>
                              <artifactId>domain</artifactId>
                              <version>${domain.version}</version>
                              <type>test-jar</type>
                      </dependency>
                      <dependency>
                              <groupId>ch.post.it.evoting</groupId>
                              <artifactId>voting-client-js</artifactId>
                              <version>${voting-client-js.version}</version>
                              <type>pom</type>
                      </dependency>
                      <dependency>
                              <groupId>ch.post.it.evoting</groupId>
                              <artifactId>voting-client-js</artifactId>
                              <version>${voting-client-js.version}</version>
                              <type>zip</type>
                      </dependency>
                      <dependency>
                              <groupId>ch.post.it.evoting.controlcomponent</groupId>
                              <artifactId>control-component</artifactId>
                              <version>${control-component.version}</version>
                      </dependency>
                      <dependency>
                              <groupId>ch.post.it.evoting.securedatamanager</groupId>
                              <artifactId>backend</artifactId>
                              <version>${secure-data-manager.version}</version>
                      </dependency>
                      <dependency>
                              <groupId>ch.post.it.evoting.votingserver</groupId>
                              <artifactId>voting-server</artifactId>
                              <version>${voting-server.version}</version>
                      </dependency>

                      <!-- Other dependencies -->
                      <!-- crypto-primitives -->
                      <dependency>
                              <groupId>ch.post.it.evoting.cryptoprimitives</groupId>
                              <artifactId>crypto-primitives</artifactId>
                              <version>${crypto-primitives.version}</version>
                      </dependency>
                      <dependency>
                              <groupId>ch.post.it.evoting.cryptoprimitives</groupId>
                              <artifactId>crypto-primitives</artifactId>
                              <version>${crypto-primitives.version}</version>
                              <classifier>tests</classifier>
                              <type>test-jar</type>
                              <scope>test</scope>
                      </dependency>

                      <!-- crypto-primitives-domain -->
                      <dependency>
                              <groupId>ch.post.it.evoting.cryptoprimitives.domain</groupId>
                              <artifactId>crypto-primitives-domain</artifactId>
```

```
                <version>${crypto-primitives-domain.version}</version>
        </dependency>
        <dependency>
                <groupId>ch.post.it.evoting.cryptoprimitives.domain</groupId>
                <artifactId>crypto-primitives-domain</artifactId>
                <version>${crypto-primitives-domain.version}</version>
                <classifier>tests</classifier>
                <type>test-jar</type>
                <scope>test</scope>
        </dependency>

        <!-- crypto-primitives-ts -->
        <dependency>
                <groupId>ch.post.it.evoting.cryptoprimitives.ts</groupId>
                <artifactId>crypto-primitives-ts</artifactId>
                <version>${crypto-primitives-ts.version}</version>
                <type>zip</type>
        </dependency>

        <!-- e-voting-libraries -->
        <dependency>
                <groupId>ch.post.it.evoting.evotinglibraries</groupId>
                <artifactId>direct-trust</artifactId>
                <version>${e-voting-libraries.version}</version>
        </dependency>
        <dependency>
                <groupId>ch.post.it.evoting.evotinglibraries</groupId>
                <artifactId>domain</artifactId>
                <version>${e-voting-libraries.version}</version>
        </dependency>
        <dependency>
                <groupId>ch.post.it.evoting.evotinglibraries</groupId>
                <artifactId>protocol-algorithms</artifactId>
                <version>${e-voting-libraries.version}</version>
        </dependency>
        <dependency>
                <groupId>ch.post.it.evoting.evotinglibraries</groupId>
                <artifactId>xml</artifactId>
                <version>${e-voting-libraries.version}</version>
        </dependency>
        <dependency>
                <groupId>ch.post.it.evoting.evotinglibraries</groupId>
                <artifactId>direct-trust</artifactId>
                <version>${e-voting-libraries.version}</version>
                <type>test-jar</type>
        </dependency>
        <dependency>
                <groupId>ch.post.it.evoting.evotinglibraries</groupId>
                <artifactId>domain</artifactId>
                <version>${e-voting-libraries.version}</version>
                <type>test-jar</type>
        </dependency>
[...]
```

In order to public to maven's central repository, it is required to be able to control a DNS record for the `post.ch` domain to setup the repository.

This means that a malicious actor able to configure DNS entries of the `post.ch` domain can publish a malicious package with the name `ch.post.it.evoting.evotinglibraries` to the Maven Central Repository.

Then, depending on the build server configuration, the malicious online package could be used instead of the local legitimate one during the build process. In this case, the adversarial actor would be able to run backdoored code in the voting infrastructure and potentially compromise the whole system.

This is purely theoretical as during this particular pentest, SCRT did not have the ability to register the required DNS record or any knowledge of the build environment for the production systems. This remark is therefore to be considered as a warning towards these types of attacks.

CONTENT-SECURITY-POLICY WEAKNESS

The application defines a Content-Security-Policy, but it allows inline scripts to be executed through the use of the `unsafe-inline` policy. This means that the application is not as well protected against Cross-Site Scripting issues as it could be. The exact policy returned by the server is the following:

```
Content-Security-Policy: default-src 'none'; script-src 'self' 'unsafe-inline' 'unsafe-eval';
style-src 'self' 'unsafe-inline'; img-src 'self' data:; connect-src 'self'; worker-src 'self';
frame-src 'none'; frame-ancestors 'none'; font-src 'self'; base-uri 'self'; form-action 'none'
```

Given the fact that no XSS issues were discovered during the audit, this is not reported as a vulnerability, but as a remark and possible improvement.

DETAILED RESULTS

VULNERABILITIES AND EXPLOITATION

P021121-01 USE OF OUTDATED SYSTEM OR SOFTWARE

SCRT		CVSS	
Impact	★☆☆☆	Base	3.7
Probability	☆☆☆☆	AV:N/AC:H/PR:N/UI:N/S:U/C:N/I:N/A:L	

PREREQUISITES	COMPROMISED ASSETS
» Specific conditions with vulnerable code	» Denial of Service » Remote Code Execution

AFFECTED SYSTEMS

ch.post.it.evoting.securedatamanager:backend	ch.post.it.evoting.votingserver:voting-server
ch.post.it.evoting.securedatamanager:packaging	ch.post.it.evoting.controlcomponent:control-component
ch.post.it.evoting.architecture:architecture-rules	ch.post.it.evoting.commandmessaging:command-messaging

DESCRIPTION

An outdated system, or a system using outdated software is more likely to be prone to attacks than a system with all updates and patches installed. A regular update is mandatory in order to correct issues that could enable an attacker to compromise the normal behaviour of the application.

EXPLOITATION

During the audit, automated checks revealed that some maven dependencies used to build components of the project use outdated packages vulnerable to known CVEs. Although none of these dependencies could be exploited during the tests, it is recommended to, wherever possible, ensure that the latest version of each package is used in order to benefit from the latest security patches:

» The `securedatamanager` and the `architecture-rules` use the package `com.squareup.retrofit2:converter-gson:2.9.0` which embeds the package `com.google.code.gson` in version 2.8.5 which may be subject to DoS attacks due to an insecure deserialization (CVE-2022-25647). It also relies on the `com.orientechnologies:orientdb-graph:3.2.19` which uses `common-collections:3.2.1` and `commons-beanutils:1.7.0` both subject to CVEs which help in exploiting deserialization issues.

- The E-voting server uses `spring-boot-starter` which relies on `snakeyaml:1.30`. This version of the package suffers multiple issues which could lead to DoS attacks or remote code execution (CVE-2022-25857 (DOS), CVE-2022-1471 (RCE)).

Possible solutions

It is recommended to regularly check whether the dependencies used in the project are vulnerable or obsolete and, if so, to update them.

COMPLEMENTS

LEGEND

SCRT SCORE

For each vulnerability discovered and detailed in this report, SCRT provides a threat assessment based on two indicators, an **Impact** and a **Probability** of exploitation.

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

However, it is important to keep in mind that this assessment is solely based on the information available to the engineers at the time of the audit. The engineers are not necessarily aware of all the details regarding the vulnerable applications or systems. Consequently, these ratings should always be reconsidered based on the context of the information system as a whole.

CVSS SCORE

In addition to its own scoring system, SCRT also provides an evaluation based on the **Common Vulnerability Scoring System** (CVSS), for each vulnerability.

As a reminder, 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

RISK CALCULATION

Each risk presented in this report is calculated as the product of an **impact** and a **probability** of exploitation, as defined in the matrix below.

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			

SCRT provides an estimation of the effort required to fix each vulnerability and thus mitigate their associated risk. It should be noted that this assessment is based on SCRT's experience, and as such might not fully reflect the context of the company or organization.

CONTEXT

The context of each vulnerability is defined by its prerequisites and a list of compromised assets. The prerequisites represent the conditions that are required for the exploitation of a given vulnerability (*e.g.*: social engineering). Compromised assets represent the theoretical or tangible result of its exploitation (*e.g.*: a user account).

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, and the 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 often published to inform end users and push developers to correct them. Many flaws are discovered and published daily, which are 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 websites 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 website, such as 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 regularly updated repository of the most commonly discovered and exploited vulnerabilities in web applications: the Open Web Application Security Project's (OWASP) TOP 10.

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 client believe it is the default gateway and make the gateway believe it is the end client, which then leads to the attacker proxying all requests between the two.

Cleartext 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 they may leverage it to attack other systems.

Additional attacks

The following attacks are usually not performed during penetration tests as they would go beyond the scope of the targeted application or system. However, SCRT deems it important to mention them here because they could be a key element in the exploitation of certain vulnerabilities.

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 a client and a server, without any of them being able to notice the manipulation. An adversary can undertake such an attack only if they have 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 targets. Sophisticated attacks (*e.g.*: *phishing*, *phoning*) are often developed in order to manipulate victims. When stated as a prerequisite for a vulnerability, social engineering means that an attacker must have some kind of interaction with their victim in order to trick them into performing an action desired by the attacker, such as clicking on a link or opening an e-mail attachment.