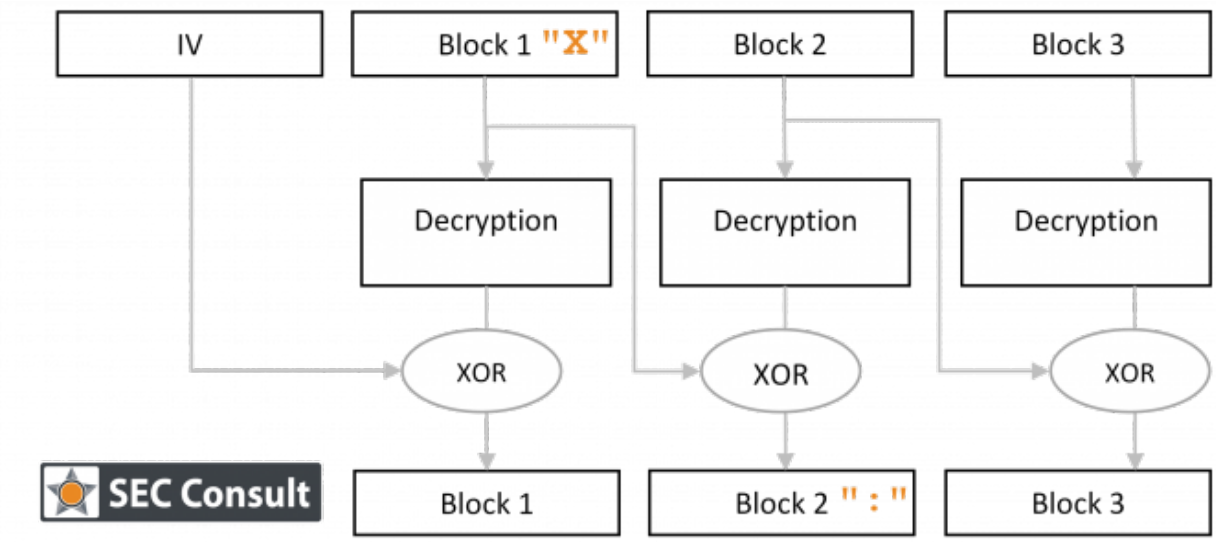
SEC Consult

BLOG



CRYPTOGRAPHIC VULNERABILITIES IN GERMAN E-GOVERNMENT LIBRARY < /EN/BLOG/2019/02/CRYPTOGRAPHIC-VULNERABILITIES-IN-GERMAN-E-GOVERNMENT-LIBRARY/>

On 5. Feb

The OSCI-Transport Library is a software component used by many German government agencies to securely exchange data via the OSCI-Transport protocol. This Java library was susceptible to two attacks that could potentially allow an attacker to bypass some of the OSCI-Transport protocol's core security promises.

Since its introduction in 2001, the OSCI-Transport protocol has been adopted by many German government agencies to allow them to securely exchange data over untrusted networks < https://www.xoev.de/detail.php?gsid=bremen83.c.3355.de> (e.g. the Internet). The OSCI-Transport protocol provides integrity, authenticity, confidentiality and non-repudiation for all data exchanged. The OSCI-Transport Library is a free implementation of this protocol and is distributed by KoSIT < https://www.xoev.de/downloads-2316#Standards> .

Back in 2017 < /vulnerability-lab/advisories/#a230> SEC Consult identified several vulnerabilities in this library. When we had a short look at the code in September 2018 again, two other vulnerabilities cou be identified, which are described in this blog post. These vulnerabilities allow an attacker to bypass the security layers that are designed to protect a message's metadata (*request and response signature* and *request and response encryption*).

Therefore, when all security measures provided by OSCI-Transport are activated, the impact of these vulnerabilities is limited. However, the impact is critical if e.g. *content signature* and *content encryption* a not active. As we have very little insight into how the OSCI-Transport protocol is used in practice, it is not possible to reasonably estimate the real-life impact of these vulnerabilities. Setting the practical applicability aside, the technical details of these vulnerabilities are rather interesting and are being shared with you now.

SEC Consult did not conduct a full security audit and can therefore not make any statement regarding the overall security of the library or its security mechanisms.

OSCI-TRANSPORT OVERVIEW

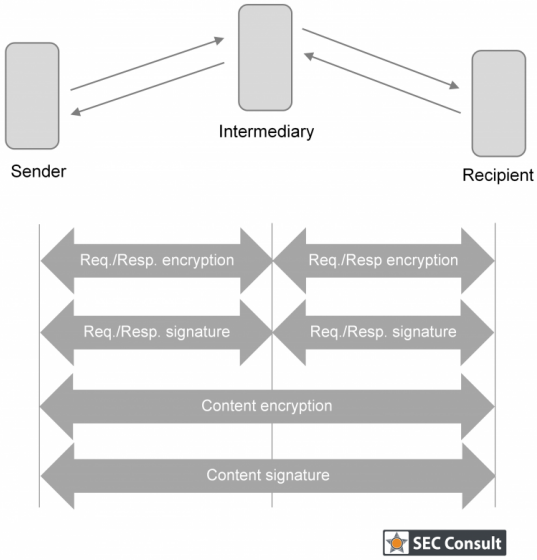
OSCI-Transport is an XML-based data exchange protocol that requires an *Intermediary* to route communication between a *Sender* and a *Recipient*. The main security features of OSCI-Transport are:

- The *content signature* provides authenticity for the payload data that is exchanged.
- The *content encryption* provides confidentiality for the payload. As the Intermediary cannot decrypt the payload, OSCI-Transport provides end-to-end encryption.
- The *request and response signature* provides authenticity for the data that is exchanged between a *Sender* and the *Intermediary* or between the *Recipient* and the *Intermediary*.
- The *request and response encryption* provides confidentiality for the communication to and from the *Intermediary*.

All of these security mechanisms are *optional* – the communication partners have to agree on which mechanisms to use.

When sending data, the *Sender* has to initiate a connection to the *Intermediary* (e.g. over HTTP) and send the OSCI-Transport XML message. For receiving data, two scenarios are defined: in the *active recipient* scenario, the recipient initiates a connection to check for a message while in the *passive recipient* scenario, the *Intermediary* initiates the connection as soon as a new message is available.

The following diagram shows to which communication paths the four security layers of OSCI-Transport apply to.



XML SIGNATURE WRAPPING

In 2017 < https://sec-consult.com/en/vulnerability-lab/advisories/#a230> SEC Consult discovered that the *request and response signature* verification of the OSCI-Transport library was vulnerable to an XM signature wrapping attack. Such an attack tricks the parser into verifying the signature against one part of the message while the data that is processed comes from another part of the message. For exam an OSCI-Transport message may contain an XML element *soap:Body* that contains the transmitted payload. Consider an attacker inserts multiple body elements and manages to trick the parser into using body for signature verification and another body to obtain the payload. The attacker could in this case provide a correctly signed element for signature verification while providing another body containing manipulated data that is then processed by the application.

The previously identified vulnerability has since been fixed. However, another approach for an XML signature wrapping attack was identified:

```
<soap:Envelope>
 <soap:Header>
  ...
  <osci:ClientSignature>
    <ds:Signature>
     <ds:SignedInfo>
      <ds:Reference URI="#body">
       ...
       <ds:DigestValue>(digest value of _original_ body)</ds:DigestValue>
      </ds:Reference>
      ...
     </ds:SignedInfo>
     <ds:SignatureValue>...</ds:SignatureValue>

     <ds:Reference URI="#body">
      ...
      <ds:DigestValue>(digest value of _new_ body)</ds:DigestValue>
     </ds:Reference>

    </ds:Signature>
  </osci:ClientSignature>
  ...
 </osci:Header>
 <soap:Body Id="body">(modified content here)</soap:Body>
</soap:envelope>
```

Unlike the previous attack, this attack does not inject an additional body element. Instead, another `Reference` element is introduced.

`Reference` elements are used to tie an XML element to their expected hash values (e.g. a `Reference` element that refers to the body contains the hash value of the body element and its children). These `Reference` elements are normally stored in a `SignedInfo` element. The actual signature is verified against the `SignedInfo` element and all its children. If a referenced element's contents were modified, the parser would detect it because the hash value in the `Reference` element no longer matches the calculated hash value of the element.

Also, the OSCI-Transport library seems to accept `Reference` elements outside of a `SignedInfo` element for hash verification. If multiple `Reference` elements refer to the same element, only the last reference is processed. An attacker could therefore manipulate the body and insert an additional `Reference` element after the `SignedInfo` element containing the modified hash value of the body. As the `SignedInfo` element has not been modified, the signature still verifies as correct.

CBC MODE WEAKNESSES

Another identified vulnerability results from the usage of the CBC encryption mode for the request and response encryption. The W3C strongly recommends < https://www.w3.org/TR/xmlenc-core1/#sec-Table-of-Algorithms> using more secure modes instead of the CBC mode. Therefore, the maintainer of the OSCI-Transport standard (KoSIT) is in the process of replacing < https://www.xoev.de/sixcms/media.php/13/KoSIT_Umstellung_GCM_20180625_finaleFassung.pdf> the legacy CBC mode with the more secure GCM mode. This transition is expected to finalize in Novem 2019. Right now, the OSCI-Transport library still supports the CBC mode, though SEC Consult is unaware of how many organisations still use it.

In 2017 < https://sec-consult.com/fxdata/seccons/prod/temedia/advisories_txt/20170630-0_KOSIT_XOEV_OSCI-Transport_library_critical_vulnerabilities_german_egovernment_v10.txt> researchers at SEC Consult were able to demonstrate a padding oracle attack against the CBC mode as it was used by the OSCI-Transport library for request and response encryption. This vulnerability has since been fixed, bu due to the known weaknesses of the CBC mode, a modification of the original attack is still viable.

In order to exploit the weaknesses of CBC, an attacker needs to find a way to get the server to behave differently depending on the contents of the decrypted message. For example, a padding oracle attack exploits the fact that a difference in behavior can be observed depending on whether the decrypted padding bytes are valid.

The new attack abuses the fact that the encrypted message is a MIME object. The following shows an example of such a MIME object:

```
MIME-Version: 1.0
Content-Type: Multipart/Related; boundary=MIME_boundary_9MnMkzyjeaR9bguP9KZvpdeoY1GEChQI; type=text/xml

--MIME_boundary_9MnMkzyjeaR9bguP9KZvpdeoY1GEChQI
Content-Type: text/xml; charset=UTF-8
Content-Transfer-Encoding: 8bit
Content-ID: <osci@message>
Content-Length: 123

...
--MIME_boundary_9MnMkzyjeaR9bguP9KZvpdeoY1GEChQI--
```

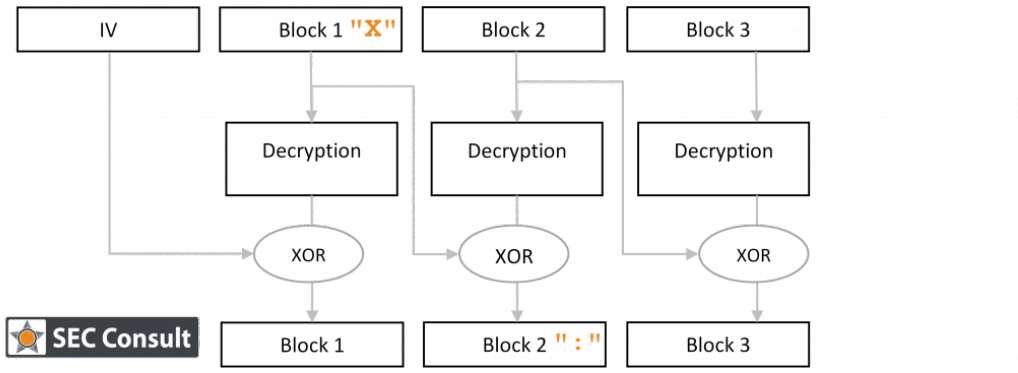
In order for this attack to work it was necessary to find a part of the MIME message that elicits a difference in attacker-observable behavior based on its content. The part we chose was the MIME header, specifically the colon character in the header line. Consider the following example:

`SomeHeader123!"%:test`
`SomeHeader123!"%_test`

The first line is parsed as a valid MIME header while the second line, due to the absence of a colon character, results in the message being rejected with an error message – that is perfect for an attack.

CBC Mode Background

The following diagram shows an example of a CBC decryption operation:



In this example we know that the last byte of the decrypted block 2 is a colon character. It is also known that the last character of the encrypted block 1 is "X". Due to the structure of the CBC mode, the last byte for the block cipher decryption function for block two can be deducted. Now it is known that

Encrypted Block 1 **XOR** *Decryption Function Output 2* = *Decrypted Block 2*

and therefore

Encrypted Block 1 (last byte) **XOR** *Decrypted Block 2 (last byte)* = *Decryption Function Output 2 (last byte)*

In this case we can figure out that "`X`" XOR "`:`" = "`b`" – which is the last byte of the output of the block cipher decryption function for block 2.

When an attacker is able to figure out that a specific character in a decrypted text is a colon character, the corresponding byte of the decryption function output can be deducted. If the attacker manages to find all the bytes the decryption function produces for a message, the message can be decrypted easily.

Finding the Colon Character

In order to exploit this vulnerability, we replace the encrypted block of a header that contains the colon character with manipulated blocks. As the headers of an OSCI message are rather predictable, we can make a pretty good guess at which block number that is. When we send this message, three outcomes are possible ("X" represents random data other than `":"`, `"\r"` and `"\n"`):

```
Content-TypeXXXXXXXXXXXXXXXXtext/xml
Content-TypeXXXXXXX:XXXXXXXXtext/xml
Content-TypeXXXXXXX\r\nXXXXXXXtext/xml
```

In the first case, a parsing error would occur, as no colon character can be found in the manipulated header. In the second case, the header would be parsed and ignored – no error would occur. In the third case, a parsing error would occur. Note that the third case is rare, as two consecutive characters by chance would have to have specific values. We can therefore, for now, ignore this case.

Effectively, this means we have a way to figure out whether a decryption of a specific message fragment contains a colon character, even though we don't know which character that is.

The Exploit

We have shown how leaked information about the plain text can allow an attacker to get the output of the decryption function. Futhermore, we have found a way to determine if the decryption of chosen cip blocks contains a colon. We can now piece those two facts together to write an exploit:

The exploit inserts 3 concatenated blocks into a MIME header (as described in the previous section). These three blocks are:

1. The *test block*: This block is initialized with random data. In each iteration one byte is specifically chosen to be modified.
2. The *target block*: The part of the original message that should be decrypted is inserted here.
3. The *suffix block*: This block contains random data.

The exploit conducts the following steps to decrypt a single byte:

1. Set the byte that should be decrypted to a known value in the test block (e.g. set the first byte to 0)
2. Send the blocks for decryption.
3. Because of the XOR operation of the CBC mode the byte in the test block influences the corresponding plain text byte in the decryption of the target block. From the server response v learn:
    1. If the server indicates that no colon character is in the header, it is certain that the tested byte did not produce a colon character in the decryption of the target block. We can remove one candidate for the plain text value.
    2. If the server indicates a colon character was found, we learn nothing – the colon character could occur in any decrypted block (e.g. also in the decryption of the test block or th suffix block).
4. The byte in the test block that was chosen in step 1 is changed to another value and execution continues at step 2.

After all possible byte values were tested, the set of possible plaintext characters has been reduced. Then, the test block and the suffix block are randomized and the process is started again. This operatio repeated until there is only one possibility left.

Lastly, several messages are sent that, based on the information gathered, should contain a colon character. If an error is recorded (i.e. the message contains no colon character) the whole process is restarted. This is to check for false positives (e.g. when \r\n was encountered at some point).

This process is repeated to gather all plaintext bytes of the message. This allows an attacker to bypass the request and response encryption of the OSCI-Transport protocol.

The following shows the exploit in action. Note the set of possible characters is reduced with every iteration:

THE PATCH

The SEC Consult Vulnerability Lab reported the issues in September 2018 through CERT-Bund, who have been very helpful with coordinating vulnerabilities in the past. KoSIT and the software vendor Governikus then analyzed the vulnerabilities and privately provided a patch for the Java and .Net version to the affected organizations. In order to give affected parties enough time to apply the patch, SEC Consult has postponed the security advisory release to 5th February 2019.

It is strongly recommended to upgrade the OSCI-Transport Library to version 1.8.3 (Java or .Net). SEC Consult also highly recommends utilizing the GCM mode as soon as possible. For organizations that depend on the CBC mode it is highly recommended using TLS as an additional layer of security.

This research was conducted by Wolfgang Ettlinger (@ettisan < https://twitter.com/ettisan>) on behalf of SEC Consult Vulnerability Lab and has also been published as a security advisory < /en/blog/advisories/multiple-vulnerabilities-in-osci-transport-library-1-2-for-german-egovernment/> .

SEC Consult is one of the leading consultants in the field of cyber and application security. The company specializes in the introduction of Information Security Management, Security Audits, Penetration Testing and DDoS Benchmarktesting, ISO 27001 Certification Support, Cyber Defense and secure Software certification.