International Journal of Research Publication and Reviews

Journal homepage: www.ijrpr.com ISSN 2582-7421

A Review of Various Attacks on Web Service

Swapnil Mahajan [a], Dr. Harsh Lohiya [b], Mr. Rishi Kushwah [c]

[a] Research scholar, Department of CSE, School of Engineering, SSSUTMS, Sehore (M.P.)
[b] Associate Professor, Department of CSE, School of Engineering, SSSUTMS, Sehore (M.P.)
[c] Assistant Professor, Department of CSE, School of Engineering, SSSUTMS, Sehore (M.P.)

ABSTRACT

Web services are considered a new concept of Internet communication and have introduced many new standards and technologies. Despite the development of many years of telecommunications work, Web services do not have more security protection than other open networks. Attack vulnerability: Thanks to new attacks specific to Web services, Web services are easy to attack according to Internet rules. In addition to being extremely effective, most attacks can be carried out with minimal effort from the attacker. This article examines vulnerabilities in the context of Web services. We attacked several web services to demonstrate the effectiveness of the threat. General countermeasures to prevent and mitigate these attacks are also discussed.

Keywords: Web Services, Security Attacks, Denial of Service, Flooding Attacks, XML, WS-Security.

1. Introduction

Web services and service-oriented architecture (SOA) are often considered one of the most important technologies of the last decade. However, the benefits of this new system are hampered by some major problems that come with this new technology. The biggest problem on the web is security [19].

The requirements of security are integrity, confidentiality and availability. Any behavior that aims to violate one of these is called attack, and the possibility of attack is called negativity.

This article lists security issues in the Web Services world. This list is not intended to be exhaustive; it is simply a selection of the best attacks we have examined during our research. Since this study focuses on availability, most attacks fall into the Denial of Service (DoS) category [22]. Denial of Service (DDoS) attacks against Estonian government and commercial websites in May [25]. These attacks are carried out by botnets using network flooding techniques. As we will show in this article, a DoS attack on a web service is less resource intensive than an attack on a non-web service. Starting with an attack on an unprotected web service, we describe in more detail the attack on WS-Security-enabled web services, and finally we describe the attack on web services used in web services. Although it has been used before for all kinds of web services, we chose WS-BPEL (or BPEL for short) for our attack because it tends to be a standard web service. The next section introduces important concepts and terms related to web services security and BPEL. Section 3 lists vulnerabilities and attacks against web services. Section 4 discusses general countermeasures, and Section 5 provides countermeasures. Finally, in Section 6 we conclude the work presented in this article.

2. Fundamentals

2.1 WS-Security

The maximum essential specification addressing the security desires of internet offerings is WS-security [21]. It collaborates with the soap specs, supplying integrity, confidentiality and authentication for web offerings. WS protection defines a cleaning soap header block- the so-called protection header-that incorporates the WS-protection extensions. Moreover, it defines how existing XML protection standards like XML Signature [2] and XML Encryption [13] are implemented to soap messages.

XML Signature permits XML fragments to be digitally signed to make certain integrity or to proof authenticity. The end result of the signing operation - i.e. the encrypted digest is positioned in a Signature element, which again is introduced to the safety header.

XML Encryption allows XML fragments to be encrypted to ensure statistics confidentiality. The encrypted fragment is replaced by an Encrypted statistics element containing the cipher textual content of the encrypted fragment as content.

Similarly, XML Encryption defines an Encrypted Key detail for key transportation functions. The default software for an encrypted key's a hybrid encryption: an XML fragment is encrypted with a randomly generated symmetric key, which itself is encrypted the use of the general public key of the message recipient. In soap messages, the Encrypted Key detail-if gift-ought to seem within the protection header.

Further to encryption and signatures, WS-safety defines safety tokens appropriate for transportation of digital identities, e.g. Username Token or X.509 certificate. An important feature of the mechanisms utilized in WS-protection is their high flexibility. they may be relevant to arbitrary parts of the soap message, leaving all other parts unattended. as a consequence, internet carrier servers and clients need to negotiate a security coverage defining the WS-security factors to be used.

WS-safety coverage [17] gives an XML syntax for declaring such safety rules. In extension to the internet service description, a server may use a WS-safety coverage record for declaring its safety needs. WS security policy permits to specify the components of a cleaning soap message that shall be encrypted or signed, the algorithms to use and the required protection tokens.

2.2 BPEL

The enterprise technique Execution Language [1] is one of the preferred applicants to end up the essential internet provider composition preferred. each BPEL report describes the workflow of a so-known as BPEL method. this type of method includes activities, representing instructions to be executed with the aid of a BPEL runtime surroundings |the BPEL engine. those activities may be labeled into communication activities representing incoming or outgoing internet provider calls, structure activities for execution order description, and other fundamental activities for added tasks, which include method variable access, temporal constraints in workflow execution or fault managing. At runtime, every deployed BPEL manner may additionally have multiple process instances, that are concurrent execution contexts of the same procedure.

One key feature of BPEL-based internet service composition is the capacity to apply asynchronous verbal exchange. A ordinary net carrier name includes a request message, at once answered by using a respond message. The requester need to hold the relationship to the server till the reply message arrives. the usage of a special language assemble, BPEL permits asynchronous behaviour, permitting the requester to disconnect after sending its request. In this example, the reply message is delivered through a new connection initiated by using the net provider server, e.g. via invoking an internet carrier on the original requester. This conversation sample is beneficial for lengthy-jogging responsibilities that can't be completed inside timeout limits of a unmarried internet provider call.

The specification in use for specifying the callback vacation spot is WS-Addressing [11], permitting the requester to specify an abstract endpoint reference within its request message, containing all facts essential for the BPEL engine to invoke the web service on the requester.

A further task a BPEL engine has to carry out is message correlation. As a BPEL engine may run numerous times of one BPEL procedure on the equal time, it turns into vital to use precise message information fields to perceive the goal method example for an incoming net carrier message. those are called correlation units in the context of BPEL.

3. Attacks

In this section we present a list of attacks on Web Services. For each attack an abstract attack methodology and impact is given, demonstrated by a concrete attack execution where appropriate. Additionally, countermeasures against the particular attacks are discussed.

3.1 Oversize Payload

One important category of Denial-of-Service attacks is called Resource Exhaustion [24]. Such attacks target at eliminating a service's availability by exhausting the resources of the service's host system, like memory, processing resources or network bandwidth. One \classic" way to perform such a Resource Exhaustion attack is to query a service using a very large request message. This is called an Oversize Payload attack [19].

Against Web Services, an Oversize Payload attack is quite easy to perform, due to the high memory consumption of XML processing. The total memory usage caused by processing one SOAP message is much higher than just the message size. This is due to the fact that most Web Service frameworks implement a tree-based XML processing model like the Document Order Model (DOM [12]). Using this model, an XML document like a SOAP message is completely read, parsed and transformed into an in-memory object representation, which occupies much more memory space than the original XML document. For common Web Service frameworks, we observed a raise in memory consumption of factor 2 to 30.Example: An Axis Web Service was attacked using a large SOAP message document, which consisted of along list of elements considered as parameter values of the Web Service operation:

```
<Envelope>

<Body>

<getArrayLength>

<item>x</item>

<item>x</item>
```

<item>x</item>

...

</getArrayLength>

</Body>

1

</Envelope>

The SOAP message had a total size of approx. 1.8 MB. The message processing induced a full CPU load for more than one minute and an additional memory usage of more than 50 MB. Enlarging the message to approx. 1.9 MB even resulted in an out-of-memory exception. An obvious countermeasure against Oversize Payload attacks consists n restriction of the total buffer size (in bytes) for incoming SOAP messages. In this case, it is sufficient to check the actual message size and reject any message exceeding the predefined limit. This method is used by the .NET 2.0 framework, which discards all SOAP messages larger than 4 MB (in the default configuration). While this countermeasure is very simple to implement, it is not suitable for Web Service messages.

A more appropriate approach uses restrictions on the XML infoset. This can be realized by modifying the XML schema inside the Web Service description and validating incoming SOAP message to this schema [7]. Details of this approach can be found in section 4.

3.2 Coercive Parsing

One of the _rst steps in processing a Web Service request is parsing the SOAP message and transforming the content to make it accessible for the application behind the Web Service. Especially when using namespaces, XML can become verbose and complex in parsing, compared to other message encodings. Thus, the XML parsing process allows other possibilities for a special kind of Denial-of Service attacks, which is called Coercive Parsing attacks [19].

Example: The following attack was performed targeting an Axis2 Web Service. The attack used a continuous sequence of opening tags:

<x>

<x>

<x>

...

The attack resulted in a CPU usage of 100% on the target system. The service's availability was massively reduced, and the incoming message was finally received with a constant rate of 150 byte/s. Thus, the attack would perform well even if the attacker has a very low bandwidth connection. The Web Service server did not abort the connection, thus this attack could apparently be continued infinitely. In our experiment, we stopped the attack after one hour.

Typical Coercive Parsing attacks targeting at resource exhaustion use a large number of namespace declarations, oversized prefix names or namespace URIs or very deeply nested XML structures. These types of attacks require different countermeasures.

An attack that is based on complex or deeply nested XML documents (like the one in the example above) can be fended by using schema validation (compare section 4).

Attacks misusing namespace declarations are harder to prevent. As the XML specification does neither limit the number of namespace declarations per XML element nor the length of the namespace URIs, any restriction on the number or length of namespace declarations would be arbitrary and could lead to unpredictable rejection of messages.

3.3 SOAPAction Spoofing

The actual Web Service operation addressed by a SOAP request is identified by the first child element of the SOAP body element. Additionally, the optional HTTP header field “SOAPAction" can be used for operation identification. Although this value only represents a hint to the actual operation, the SOAPAction field value is often used as the only qualifier for the requested operation.

This is based on the bogus optimization that evaluating the HTTP header does not require any XML processing. This twofold operation identification enables two classes of attacks. The first one is executed by a man-in-the middle attacker and tries to invoke an operation different from the one specified inside the SOAP body. It is based

on modification of the HTTP header.

Example: The following attack was performed targeting a .NET Web Service. The deployed service provided two operations: op1(string s) and op2(int x)| with the respective SOAPAction and message element also named opn. The following message (including the HTTP header) was sent to the service:

```
POST /Service.asmx HTTP/1.1
...
SOAPAction: "op2"
<Envelope>
<Body>
<op1>
<s>Hello</s>
</op1>
</Body>
</Envelope>
```

The method call that was triggered by this message was: op2(0). This shows that the operation is selected solely by the SOAPAction value from the HTTP header. Even worse, the \wrong" operation was executed despite of incompatible parameter names and types.

The example shows how modifications of the HTTP header can invoke methods that were not intended by the SOAP message creator. As the HTTP header is not secured by WS-Security and is newly created at every SOAP intermediary, it can easily be modified.

The second class of SOAPAction spoofing attacks is executed by the Web Service client and tries to bypass an HTTP gateway.

Example: The following attack was performed targeting an Axis2 Web Service. The deployed service provided two operations**: hidden and visible** - with the respective SOAPAction and message element equally named. The following message (including the HTTP header) was sent to the service:

```
POST /axis2/testService HTTP/1.1
...
SOAPAction: "visible"
<Envelope>
<Body>
<hidden />
</Body>
</Envelope>
```

The Axis2 server actually ignored the SOAPAction value and invoked the hidden operation instead. If an HTTP border gateway which of course operates on the HTTP header only is configured to reject hidden and accept visible accesses, this attack allows calling hidden anyway.

A countermeasure to SOAPAction Spoofing attacks would be to determine the operation by the SOAP body content. Additionally, the operations determined by the HTTP header and by the SOAP body must be compared and any difference should be regarded as threat and result in rejecting the Web Service request.

3.4 XML Injection

An XML Injection attack tries to modify the XML structure of a SOAP message (or any other XML document) by inserting content e.g. operation parameters containing XML tags. Such attacks are possible if the special characters "<" and ">" are not escaped appropriately. At the Web Service server side, this content is regarded as part of the SOAP message structure and can lead to undesired effects.

Example: The following attack was executed against a .NET Web Service. The deployed service method has two parameters a and b, both of type xsd:int. This service was invoked using the following SOAP message:

```
<Envelope>
<Body>
<HelloWorld>
```

<a>

<b>1</b></a>

<b>

2</b>

</HelloWorld>

</Body>

</Envelope>

Such a message could result from an XML Injection

attack: <b>1</b> was inserted as parameter content

without escaping "<" and ">". As the SOAP message obviously violates the Web Service schema, it should be rejected. But in fact, not only that the message was accepted by .NET, the resulting parameter values inside the service application for this request were: a = 1, b = 0. Thus, the attacker was able to modify the value of b just by modifying the content of a. It is easy to imagine a scenario in which this can lead to unintended service behavior, e.g. access to restricted data.

An important step in detecting such attacks is a strict schema validation on the SOAP message, including data type validation as possible (see section 4). This would have rejected the message from the example attack.

3.5 WSDL Scanning

The WSDL advertises a service's operations including parameters, data types and network bindings. Usually, some of these operations should be accessed from the local network only (here called internal operations), while other operations are intended to be o_ered to the outer network (here called external operations). If the Web Service is created using common Web Service framework tools, the (only) generated WSDL contains all operations. In this case, an external client gains knowledge of the internal operations and can invoke them. The first step in avoiding such accesses is providing a separate WSDL to external clients that contains the external operations only. However, as the Web Service endpoint is still externally accessible (for invoking the external operations), an attacker can try to guess the omitted operations and call them. This attack is called WSDL Scanning. For example, an internet shop system needs methods for placing an order for customers (sendOrder) and for administrating the orders (adminOrders). Of course, the latter one is intended to be called from within the shop's intranet only. If both sendOrder and adminOrders operations are o_ered by one Web Service, an attacker with the knowledge of sendOrder can easily _nd the administration method also. Other examples for vulnerable internal operations are legacy and debug methods.

One countermeasure to this attack is deploying the internal and external operations to separate Web Services, preferable even on different servers. If this is not applicable, invocations of internal operations must be controlled and rejected if originated from an external client. This is a typical task for a border gateway. Unfortunately, external and internal request messages have the same destination IP address, TCP port and even HTTP URL. Thus, packet filters and HTTP firewalls can not decide whether the Web Service operation is allowed or not. Therefore, a Web-Service-aware XML firewall is required which is configured with the externally visible operations.

3.6 Metadata Spoofing

A Web Service client retrieves all information regarding a Web Service invocation (i.e. message format, network location, security requirements etc.) from the metadata documents provided by the Web Service server. Currently, this metadata usually is distributed using communication protocols like HTTP or mail. These circumstances open new attack possibilities aiming at spoofing these metadata. The most relevant attacks in this category are WSDL Spoofing and Security Policy Spoofing.

Supposably most promising for WSDL Spoofing is the modification of the network endpoints and the references to security policies. A modified endpoint enables the attacker to easily establish a man-in-the-middle attack for eavesdropping or data modification. If additionally a spoofed security policy with lower or no security requirements is used, such attacks are possible despite the use of WS-Security. To avoid Metadata Spoofing, all metadata documents must be carefully checked for authenticity. However, the mechanisms for securing metadata documents are not standardized-in contrast to methods for securing SOAP messages. Additionally, a prior establishment of trust relationships is required, which is not always possible or intended.

3.7 Attack Obfuscation

Using WS-Security on Web Services introduces new problems concerning service availability. By providing confidentiality to sensible data, XML Encryption can mask message content from being inspected. As this encrypted

content can contain an intended attack-like Oversize Payload, Coercive Parsing or XML Injection-encryption can be used to conceal attacks.

Most problematical is that this kind of attacks is hard to detect. To analyse the message structure-e.g. for schema validation-decryption is necessary. There are two possibilities on how a targeted system may be effected. If decryption is done after message validation, the malicious message content may pass the message validation. If decryption is done before message validation, the system may tie up during message decryption because of the XML and cryptographic processing. Thus, even if a system is able to counter the unencrypted attack, obfuscated attacks may affect a target system anyway.

Example: During tests with Axis2 a weak spot for obfuscated Oversize Payload attacks was revealed. In this scenario a single SOAP message was sent to the server. Sending a message containing an encrypted and signed body with a size of 1 MB caused a full CPU load for 23 seconds and resulted in an out-of-memory exception, while the Java runtime environment additionally consumed approx. 90 MB of system memory. In comparison, unencrypted messages with message sizes of 20 MB and more were processed by the Axis2 server within a processing time of beneath one second.

To counter obfuscated attacks, a good strategy is performing message validation on decrypted content. The best effort uses a stepwise decryption and validation.

This can help reducing memory consumption and enables an early detection of malicious message content.

3.8 Oversized Cryptography

Another problem introduced by WS-Security is the flexible usability of security elements: encryption may be used almost anywhere within a

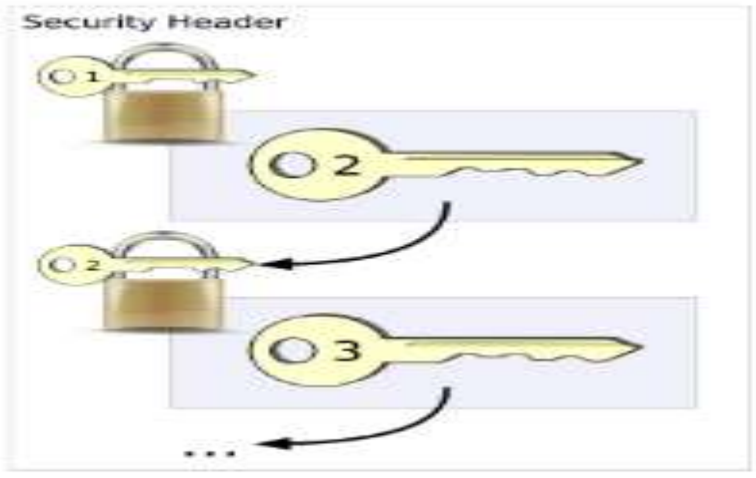

Fig. 1 Example for an encrypted key chain (schematically)

SOAP message, and the flexible layout of the security header allows no strict schema validation. The various possibilities for using security elements limits a schema validation to check each single element, but neither order nor occurrence checks for multiple elements are possible. This flexibility can be misused for attacks.

A self-evident attack relies on an oversized security header. If the target system processes or buffers the whole security header, the target system may be effected the same way as from an Oversize Payload attack (see section 3.1).

A more complex attack with an oversized security header uses chained encrypted keys. In this chain, each encrypted key is used to encrypt the next key (see figure 1). Thus, the target system is forced to decrypt every encrypted key, as each key is needed for decryption of the next one. This may effect the target system in two ways. First, the target system must buffer every key, as it is unknown before the end of message processing, whether an encrypted key is used for other encrypted content. This leads to high memory consumption. Second, the decryption operations needs processing resources. Especially asymmetric algorithms, which are typically used for key transport, are highly CPU consuming. A similar attack uses a large number of nested encrypted blocks within a SOAP message (like a Russian matryoshka doll). The target system must decrypt all the nested blocks in order to process the inner content.

This induces a high CPU load due to the large number of cryptographic operations. Additionally, if the decrypted content is buffered before further processing, the memory consumption is a multiple of the original message size.

As a countermeasure, the usage of WS-Security elements must be restricted, and messages exceeding these limits must be rejected. In contrast to the Oversized Payload and the Attack Obfuscation attack, schema based restrictions are only partly effective: the security header schema allows any kind and amount of security tokens, and encrypted blocks are allowed nearly everywhere within the SOAP message.

A better approach is accepting only the security elements explicitly required by the security policy. This is called Strict WS-SecurityPolicy Enforcement and is further explained in section 4.

3.9 BPEL State Deviation

As BPEL processes need to be called by external communication partners, a BPEL engine provides Web Service end points accepting every possible incoming message. Due to the fact that one BPEL process may have many process instances running concurrently, these communication endpoints are open for incoming connections at any time. Thus, a malicious Web Service client might attack these open Web Service endpoints using messages that are correct regarding their message structure, but that are not properly correlated to any existing process instance. These correlation-invalid messages will be discarded within the BPEL engine, but they cause a huge amount of redundant work. Each message must be read and processed completely, searching all existing process instances for a match, before the message may safely be discarded. Thus, the computational resources of the BPEL engine get exhausted by processing such invalid messages.

Example: The following attack was executed against a BPEL engine running one BPEL process. The process contained amongst other activities a sequence of two receive activities first and second, with only first initiating a new process instance. Additionally, the process defines a number of correlation properties for process instance identification. The attack used SOAP messages invoking operation second and containing correlation properties that did not match to any running process instance. The BPEL engine was attacked by a sequence of 1000 messages, summing up to a total payload of 0.5 MB. The attack messages were correctly discarded by the BPEL engine but resulted in an additional memory consumption of 350 MB and a full CPU load for more than 2 hours.

A second subtype of this attack uses correct correlation properties, but targets a receive activity that is not enabled in the actual state of the instance's process execution. These messages are not correlation-invalid but state-invalid. Their impact instead is the same: resource exhaustion on the BPEL engine's processing resources, leading to a reduced quality of service or even a loss of availability. To find state deviation attacks, it is necessary to identify and reject correlation-invalid and state-invalid messages, using as few computational resources as possible.

Note that the identification of state deviation attack messages differs widely for these two message types. A firewall approach fending both attack types was described in [10] and [15].

3.10 Instantiation Flooding

Every BPEL-based workflow definition contains at least one communication activity that creates a new process instance each time a message arrives. Such a process instance immediately starts its execution according to the instructions given in the process description. The execution will be paused each time a receive activity is reached, continuing after reception of the expected message. By reaching a termination point, execution is stopped and the process instance is destroyed. Note that all execution activities except receive activities are completely driven by the BPEL engine alone. Just in case of receive, an external message triggers the next executions.

Keeping this in mind, imagine an attack that continuously calls the instantiating activity's endpoint. For each incoming SOAP message, the BPEL engine will create a new process instance and start its execution, running each of these until they reach either a receive activity or a termination point. As a result, the BPEL engine will get into heavy load for message parsing, process instantiation and activity execution, which will decrease or even nullify the availability of the BPEL engine.

When examining instantiation flooding in the context of BPEL, there are some behavioural distinctions to make on the attack's impact. First, you have to realize that the BPEL engine itself is not the only target reached by this attack. Each newly created process instance is executed just like a valid one, including all its outgoing Web Service requests to external communication partners. Thus, these communication partners will undergo a raise in requests initiated by the BPEL engine as well (see next section).

Further, note that the processing of a single attack message and thus the resource exhaustion impact is determined by one of the following circumstances.

-If the process under attack does not contain any receive activity (beside the initial one), all executions will stop when reaching a termination point, destroying the initially created process instance. Thus, the BPEL engine will undergo the “usual“ traffic for each attack message.

-If there exists a receive (or pick) activity on the execution path chosen for the incoming attack message, all created process instances will run up to that receive activity. Here, according to the BPEL specification [1], all but the first request will cause a BPEL execution fault, as there already exists a receiving process instance with the same correlation data. Thus, all but the _rst process instance will execute the fault handler (if existing in the process description), causing a rollback on all activities that have already been executed previously. This will double the resource load for both BPEL engine and previously requested communication peers that are included in rollback procedures.

-If there exists a receive (or pick) activity on the execution path, and the attack uses unique correlation data for each attack message (such as including a counter's value in one of the correlation data fields), none of the messages will cause a fault as stated above. Instead, all attack messages will cause creation of new process instances that all will run up to the first receive activity and wait there forever (or-in case of pick- as long as the timeout case specifies). As a result, the BPEL engine will need a huge amount of persistent storage for keeping those process instances that never will complete. Further, the task of identifying the correct process instance for each message arriving at such an overfull communication endpoint will become really hard, as the message's correlation data must be compared to that of each process instance waiting. Due to the huge number of such waiting process instances, this task will exhaust resources a lot more than before the attack.

The overall impact of both attack types depends on the structure of the BPEL process, but it potentially is a multiple of the load necessary to perform the attack. Further, it may reach not just the BPEL engine, but some of its communication partners as well (see next section).

Fending such flooding attacks can only be achieved by identification and rejection of semantically invalid requests (attack messages). This is a non-trivial task, as the decision on whether an incoming message is semantically invalid can only be made after it has been processed and identified. But even at this stage it is hardly determinable whether a request was valid or malicious, as the semantics of a process usually are not included in the process description.

3.11 Indirect Flooding

Using the same methodology as presented in the previous section, the attack target of the indirect flooding attack differs. The idea of this attack is to use the BPEL engine as intermediate for an attack on a target system behind" the BPEL engine. Imagine an architecture as shown in figure 2, and think of a BPEL process that repeatedly calls a Web Service provided by the attack target system, for example creating customer accounts with several details.

By flooding the process within the BPEL engine with instantiating attack messages (as shown in the previous section), the BPEL engine will undergo a heavy load itself, but it merely will cause an equally heavy load on the target system. Thus, if the target system is not as powerful as the BPEL engine, it will loose availability, finally resulting in a Denial-of-Service.

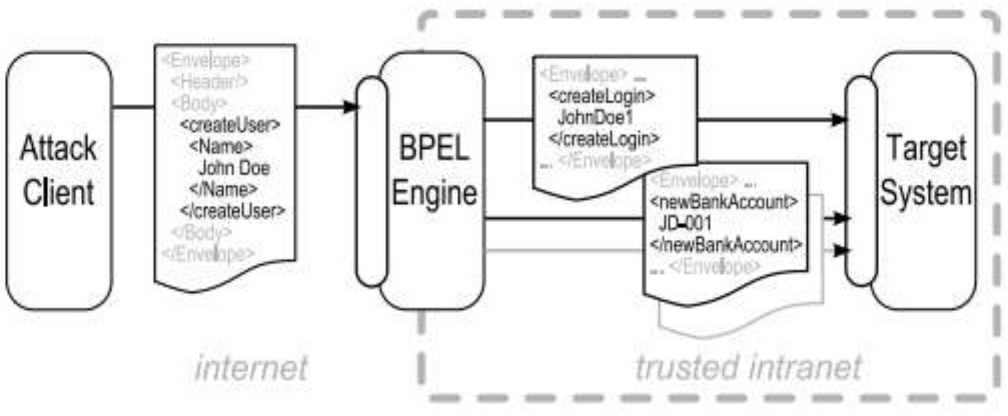

Fig. 2 Architecture for Indirect Flooding Attack

Using this attack method, the attacker bypasses any firewall on his direct link to the target system. Even if the target system is not connected to the outside world at all and only communicates with the BPEL engine, it stays attackable. Note that this attack method cannot be fended using WS-Security or similar approaches, as the connection between BPEL engine and target system is used in a completely valid and trustful way.

Again, fending such attacks needs identification and rejection of attack messages. The complication raised here is that the responsibility for attack prevention is at the BPEL engine, but the impact is on the target system. Thinking of a scenario where BPEL engine and target system communicate over inter-corporate boundaries, this task may become a political rather than a technical problem. Further, as the workflow may spread over multiple systems hosted by multiple companies, an attack may propagate throughout the system, making it difficult to identify its real entry point.

3.12 WS-Addressing Spoofing

The use of WS-Addressing for asynchronous Web Service calls raises a lot of attack possibilities, which all have in common that they use modified callback endpoint references. The most simple approach is to use an arbitrary invalid endpoint URL as callback endpoint reference. As a result, the BPEL engine will perform the execution of the process involved, then try to callback the initiator. This will result either in a direct error (refused connection, HTTP server error or SOAP fault of any kind) or in a timeout, depending on the endpoint URL the reference denotes. Thus, the BPEL engine will raise an execution fault and call matching fault handlers and compensation handlers. All in all, the BPEL engine will execute the full process and then perform a complete rollback. Used as a flooding attack, this will cause heavy load on the BPEL engine. Compared to usual flooding attacks presented above, the workload produced by each attack message is maximized, as-in most processes-the fault will be thrown within the last communication activity of the process.

The core countermeasure against any kind of WS-Addressing spoofing is verification of the caller's endpoint URL, ideally at the beginning of a process execution. This would enable early message rejection, preventing the BPEL engine from unnecessary workload.

3.13 Middleware Hijacking

This attack uses WS-Addressing spoofing again, but it points the attacker's endpoint URL to an existing target system, running a real service at the URL specified (see figure 3). As a result, the Web Service server (e.g. a BPEL engine) will repeatedly try to \answer" the attacker's requests using this specified URL. Thus, the service under attack receives a huge amount of requests containing SOAPFaults or invalid SOAP messages (or even worse: valid ones).

As Web Service servers are usually driven by powerful server machines, it is possible that the target system will suffer a Denial-of-Service before the hijacked server does (see section 3.12). Thus, the attacker uses the power of the server host system to tear down the target system. As an example, the Axis2 Web Service framework today is shipped with WS-Addressing module enabled by default. Thus, any Web Service driven by Axis2 potentially is vulnerable to become hijacked using WSAddressing Spoofing. Since SOAPFault messages are always delivered to the address specified in the <FaultTo> SOAP header, it does not even need a valid service execution at the server to play the trick; a faulty message with appropriate <FaultTo> address is sufficient. Note that-just like with the indirect flooding attack (see section 3.11)- it is possible to use this technique to attack any system behind an internet firewall. Unlike the general indirect flooding attack, the use of WSAddressing even enables the attacker to select the target system of the attack himself.

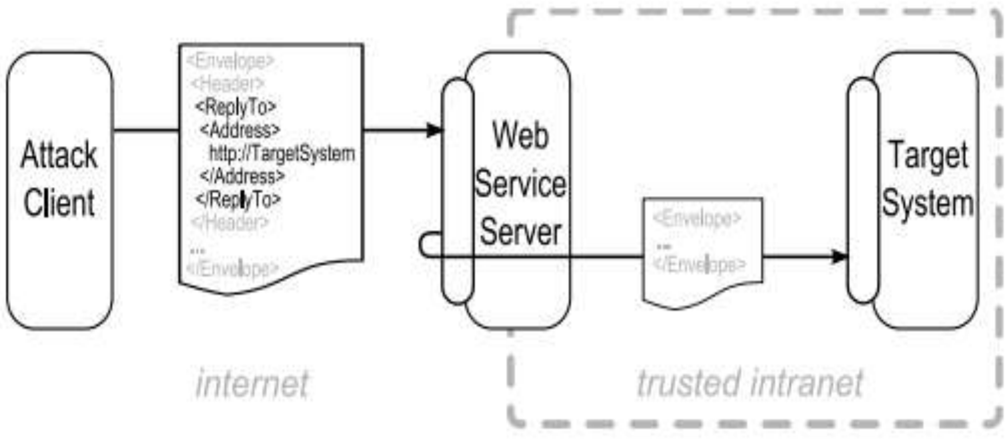

Fig. 3 Architecture for Middleware Hijacking Attack

4. General Countermeasure Approaches\

Attacks on web services, like any other system, rely on various vulnerabilities. Therefore, countermeasures against the attack are very extensive. However, there are some common defense mechanisms.

4.1 Schema Validation

Scheme authentication can be used against attacks that use messages that do not match the web service description. Such attacks are called protocol message syntax evasion [18]. By validating incoming messages from WSDL to the generated XML schema, attacks can be detected as described in Sections 3.2 and 3.4.

However, schema validation is not used or enabled in the current Web Services framework. This is mainly for performance reasons, as schema validation is associated with CPU load and memory consumption.

Schema validation is also effective against some other attacks against Web Service applications, such as SQL Injection or Parameter Tampering [19] that use invalid messages.

In addition, the approval of the scheme can be used as a basis for other countermeasures. An important example is limiting the XML infoset to limit the memory required to process the message, as described in Section 3.1. We call this hardening scheme.

4.2 Schema Hardening

The general idea is to analyze the scheme- e.g. From a Web service description—for constructs that allow unbounded large or complex XML trees. These constructions are modified to meet the limited boundaries.

For example, if a web service description defines an unbounded list of elements, the list is converted to a list with a limited number of elements. Within the XML schema, the entry <element maxOccurs="unbounded"> is replaced by <element maxOccurs="n">, where n is the bounded number. Defining such limits is easy for most services. The advantage of this restriction—compared to the network buffer size limit—is that the limit can be included in the service's “official” web service description and thus visible to clients in advance.

Another application of schema hardening is to remove non-public operations from the schema within a web service description (see Section 3.5).

There are many more possibilities to harden the description of the web service—and thus generate the XML schema. Details can be found in [7]. The same article also discusses problems caused by processing schemas with large “maxOccurs” values.

4.3 Strict WS-Security Policy Enforcement

A WS-Security policy defines a minimum set of security tokens that must be included in a SOAP message to fulfill the policy. The specification does not provide the possibility to declare their maximum use. As discussed earlier—an attacker can add an unlimited number of additional tokens, involving the target system in expensive cryptographic calculations and forcing high memory usage.

To avoid this, a good strategy is to consider the requirements of the WS-Security Policy document not only as minimum requirements, but also as maximum requirements. This means, the SOAP message must contain exactly the security tokens specified by the security policy—no less, no more.

As shown in [6], this limitation does not restrict functionality, but enables the detection of attacks using large-sized ciphers and can help mitigate their effects.

4.4 Event-based SOAP message processing

The effectiveness of the countermeasures offered above enormously depends on their implementation. Checking a soap message for conformance to the message schema and the safety policy requires XML and WS-safety processing. these operations have to be carried out very useful resource-economically, otherwise the safety device could be liable to comparable attacks because the net carrier itself.

Assaults the use of big cleaning soap messages make tree-based implementations like DOM fallacious for a protection device. Such implementations require that the message need to be completely read from the network and built intoa report tree earlier than the soap message may be in addition processed. hence, before the inspection has started out, a big quantity of reminiscence has already been fed on. some tree-primarily based implementations assemble best parts of then record tree, which barely reduces memory consumption,

However does not cast off the essential trouble of tree-primarily based approaches; each XML report ought to be completely study and stored [23].

A protection system ought to use an occasion-primarily based XML processing model like SAX [26]. the primary gain of event-based totally XML processing is the possibility to stumble on invalid message content and abort futher processing. This way, reminiscence intake and CPU usage can be minimized.

The results of the example assault described in section 3.7 reveal the fact that Axis2 uses a stream based message processing model (called AXIOM [14]), however Rampart- the Axis2 WS-security aspect- does no longer [4].

At the same time as schema validation is executed in an eventbased way by means of a number of modern-day implementations (e.g. Xerces, .internet), WS-security is normally nonetheless processed using XML bushes (e.g. Apache Rampart). WSSecurity-enabled messages consist of a number of references between the WS-security tokens, and consequently eventbased assessment is tough to realise. however, assuming some minor restrictions, it's miles possible to carry out eventbased WS-security validation [8].

Further, in [5], Gruschka indicates techniques for processing and validating net service messages in a entire occasion-based totally way. He additionally proves that the combination of event-based processing and strict protocol validation fends Denial-of-service assaults.

4.5 WS-Security

A common misunderstanding about WS-Security is that its usage automatically ensures full security for Web Services. As shown before, WS-Security defines mechanisms for enabling integrity and confidentiality for Web Service messages. However, if the corresponding WS-Security Policy is not defined correctly, attacks on integrity and confidentiality are possible using so-called XML rewriting attacks [3; 20]. More important in the context of this article, WS-Security does not define any direct countermeasures against attacks like Denial-of-Service.

A well-known protective mechanism for service availability is access control. Access control restricts access to the service to trusted users, which are supposed to be less - dangerous" regarding attacks. Additionally, access control enables accountability, allowing to exclude and prosecute the attacker. WS-Security defines security tokens for authentication, which can be used for access control systems.

Of course, access control cannot fully eliminate the threat of attacks. First of all, even trusted communication partners can-intentionally or unintentionally-execute attacks.

Further, due to the fact that authentication needs a key infrastructure, it is not applicable in B2C relationships as there is no wide-spread key infrastructure among private users.

Finally, the usage of WS-Security itself enables new kinds of DoS attacks, as seen in sections 3.7{3.8. Thus, authentication for Web Services must be performed in consideration of such attacks. In [9] e.g. a Denial-of Service-robust authentication scheme for Web Service is presented.

To resume, WS-Security is one of the important building blocks for fending attacks but has to be applied carefully and is- unlike often considered-not a magic bullet against network threats.

5. Conclusion

Like any upcoming technology, Web Services also face several security issues. The attacks presented in this article illustrate how easily a poorly secured web service server can be affected by one or a few messages. While some vulnerabilities are due to implementation weaknesses, most exploit fundamental protocol flaws and abuse the flexibility provided within WS-related standards.

Therefore, in order to cope with these threats, developers and users of web services must be aware of the vulnerabilities and their potential impact. In addition, researchers need to examine existing web service standards and identify additional vulnerabilities in order to develop more accurate countermeasures. Only the refinement of mitigation techniques, along with integration into every WebService-based system, will address these challenges and help ensure that Web Services are as secure as possible.

REFERENCES

1. Andrews T, Curbera F, Dholakia H, Goland Y, Klein J, Leymann F, Liu K, Roller D, Smith D, Thatte S, Tricko- vic I, Weerawarana S (2003) Business Process Execution Language for Web Services Version 1.1. Oasis Standard.

2. Bartel M, Boyer J, Fox B, LaMacchia B, Simon E (2002) XML-Signature Syntax and Processing. W3C Recommendation

3. Bhargavan K, Fournet C, Gordon AD, O'Shea G (2005) An advisor for Web Services security policies. In: SWS '05: Proceedings of the 2005 workshop on Secure web services, ACM Press, New York, NY, USA, pp 1{9

4. Fernando R (2006) Secure web services with apache rampart. Tech. rep., WSO2 Oxygen Tank

5. Gruschka N (2008) Schutz von Web Services durch erweiterte und effiziente Nachrichtenvalidierung. PhD thesis, Christian-Albrechts-University of Kiel, Germany

6. Gruschka N, Herkenhoner R (2006) WS-SecurityPolicy Decision and Enforcement for Web Service Firewalls. In: Proceedings of the IEEE/IST Workshop on Monitoring, Attack Detection and Mitigation

7. Gruschka N, Luttenberger N (2006) Protecting Web Services from DoS Attacks by SOAP Message Validation. In: Proceedings of the IFIP TC-11 21. International Information Security Conference (SEC 2006)

8. Gruschka N, Luttenberger N, Herkenhoner R (2006) Event-based SOAP message validation for WS Security Policy-Enriched web services. In: Proceedings of the 2006 International Conference on Semantic Web & Web Services

9. Gruschka N, Herkenhoner R, Luttenberger N (2007) Access Control Enforcement for Web Services by Event Based Security Token Processing. In: Braun T, Carle G, Stiller B (eds) 15. ITG/Gi Fachtagung Kommunikation in Verteilten Systemen (KiVS 2007), pp 371{382

10. Gruschka N, Jensen M, Luttenberger N (2007) A Stateful Web Service Firewall for BPEL. Proceedings of the IEEE International Conference on Web Services (ICWS 2007)

11. Gudgin M, Hadley M, Rogers T (2006) Web Services Addressing 1.0 - SOAP Binding. W3C Recommendation

12. Hors AL, Hegaret PL, Wood L, Nicol G, Robie J, Champion M, Byrne S (2004) Document Object Model (DOM) Level 3 Core Specifcation. W3C Recommendation

13. Imamura T, Dillaway B, Simon E (2002) XML Encryption Syntax and Processing. W3C Recommendation

14. Jayasinghe D (2006) SOA development with Axis2: Understanding Axis2 basis. IBM developer Works

15. Jensen M (2008) BPEL Firewall - Abwehr von Angriffen auf zustandsbehaftete Web Services (german). VDM Verlag Dr. Muller, ISBN 9783836485517

16. Jensen M, Gruschka N, Luttenberger N (2008) The Impact of Flooding Attacks on Network-based Services. In: Proceedings of the IEEE International Conference on Availability, Reliability and Security

17. Kaler C, Nadalin (editors) A (2005) Web Services Security Policy Language (WS-Security Policy) 1.1

18. Leiwo J, Nikander P, Aura T (2000) Towards network denial of service resistant protocols. In: Proc. of the 15th International Information Security Conference (IFIP/SEC)

19. Lindstrom P (2004) Attacking and Defending Web Service. A Spire Research Report

20. McIntosh M, Austel P (2005) XML signature element wrapping attacks and countermeasures. In: SWS '05: Proceedings of the 2005 workshop on Secure web services, ACM Press, New York, NY, USA, pp 20-27

21. Nadalin A, Kaler C, Monzillo R, Hallam-Baker P (2006) Web Services Security: SOAP Message Security 1.1 (WS Security 2004)

22. Needham RM (1994) Denial of service: an example. Commun ACM 37(11):42-46

23. Noga ML, Schott S, Lowe W (2002) Lazy XML processing. In: DocEng '02: Proceedings of the 2002 ACM symposium on document engineering, ACM Press, New York,