The Blind Spot

Content-based attacks on web-based SAP applications
Introduction

Thanks to the great success of SAP NetWeaver as an enterprise application platform, with the rich feature set it delivers, and the outlook of migrating legacy applications with limited effort, an increasing number of SAP applications are being accessed via web interfaces, partly through being available on the Internet, in order to support and streamline business processes involving external users, such as partners or suppliers. As of today, SAP customers worldwide run more than 170,000 production NetWeaver systems and this number continues to grow by over 20% year on year.

Due to the sensitive nature of the information and the business-critical nature of data stored and processed by these applications, they are destined to become a preferred target for cyber-attackers, both professional and non-professional.

Compromising an SAP application can have a severe negative impact on the organization that owns it. If disclosed, a successful attack can seriously damage a company’s reputation, for example through exposure of customers’ or clients’ personal or confidential data of a sensitive nature. Further, when maliciously manipulated, information stored in SAP applications can induce financial damage and lead to erroneous and potentially fatal business decisions.

Therefore, SAP customers implement security best practices in order to secure their application environment from various threats. However, while they can restrict or deny access to the SAP server at network level and operating system level, they obviously cannot restrict access to the web interface necessary to work with the application.

So, while the server may be well shielded by firewalls, anti-malware and proper access controls, securing the application calls for an extra effort to ensure the application is not being “fed poisonous data” by attackers; data that may lead to the exposing of critical business data, the application, the server, or even the entire network.
Content-based attack vectors

Unlike network-based attacks or authentication-based attacks, content-based attacks attempt to exploit vulnerabilities in the way applications process data provided by the user as part of their regular operation. In general, one needs to differentiate between the three standard approaches attackers can make:

Form-data based attacks
Attackers enter information into the web-based application that causes the application to either perform operations outside the norm, or that causes the browser of another user to perform unwanted tasks when viewing the data. Cross-Site Scripting (XSS) and various types of injections, most prevalently SQL injections, fall into this category.

Parameter-based attacks
Attackers attempt to pass parameters to the application, outside of the fields exposed in the web application’s user interface, in an effort to create unwanted application behavior. Often, directory traversal attacks or simple denial of service (DoS) attacks are carried out as parameter-based attacks.

File-based attacks
The attacker transfers/uploads a file to your application which, when displayed in another user’s browser, will perform unwanted and malicious tasks. While seldom critical on their own, file-based attacks are very powerful when combined with Cross-Site Scripting (XSS).
Attack techniques

Generally speaking, the techniques used to attack SAP-based web-based applications are no different than for any other web-based application. The most common ones are:

Cross Site Scripting

XSS aims at injecting code, most commonly JavaScript, into a web session or into dynamically generated web pages, which are the basis of virtually any web-based application. There are three types of XSS:

1. Reflected/non-persistent XSS

This attack relies on a website reflecting a user input back to the user without proper “sanitation” of the user-provided data. Reflected XSS requires the victim to click on a specially crafted link that takes them to a site vulnerable to XSS. This can either be a direct link or inside an IFRAME.

Consider the following example:

A site offers a search function and outputs the search term along with the search results. So a search for “Rubber ducky” returns:

<table>
<thead>
<tr>
<th>Application Output</th>
<th>HTML Source (partial/simplified)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Site search</td>
<td><code>&lt;html&gt;</code></td>
</tr>
<tr>
<td></td>
<td><code>&lt;head&gt;</code></td>
</tr>
<tr>
<td></td>
<td><code>&lt;title&gt;Site Search&lt;/title&gt;</code></td>
</tr>
<tr>
<td></td>
<td><code>&lt;/head&gt;</code></td>
</tr>
<tr>
<td></td>
<td><code>&lt;body&gt;</code></td>
</tr>
<tr>
<td></td>
<td><code>&lt;center&gt;</code></td>
</tr>
<tr>
<td></td>
<td><code>Site search&lt;/b&gt;</code></td>
</tr>
<tr>
<td></td>
<td><code>&lt;/center&gt;</code></td>
</tr>
<tr>
<td></td>
<td><code>&lt;form action=&quot;siteserach.php&quot; method=&quot;GET&quot;&gt;</code></td>
</tr>
<tr>
<td></td>
<td><code>&lt;input name=&quot;search&quot; type=&quot;text&quot;&gt;</code></td>
</tr>
<tr>
<td></td>
<td><code>&lt;input type=&quot;submit&quot; value=&quot;find&quot;&gt;</code></td>
</tr>
<tr>
<td>Search results for “Rubber ducky”:</td>
<td><code>&lt;/form&gt;</code></td>
</tr>
<tr>
<td>Ernie’s rubber ducky song</td>
<td><code>&lt;a href=&quot;result1.html&quot;&gt;Ernie's rubber ducky song&lt;/a&gt;</code></td>
</tr>
<tr>
<td>Rubber Ducks – Amazon.com</td>
<td><code>&lt;a href=&quot;result2.html&quot;&gt;Rubber Ducks – Amazon.com&lt;/a&gt;</code></td>
</tr>
<tr>
<td>Pictures of rubber ducks</td>
<td><code>&lt;a href=&quot;result3.html&quot;&gt;Pictures of rubber ducks&lt;/a&gt;</code></td>
</tr>
<tr>
<td>...</td>
<td><code>&lt;/body&gt;</code></td>
</tr>
<tr>
<td></td>
<td><code>&lt;/html&gt;</code></td>
</tr>
</tbody>
</table>
So, if an attacker enters:

```html
<script>alert("Pwned by XSS")>Test
</script>
```

into the form, the results page would contain a JavaScript directive, which would be executed upon displaying the results page.

---

<table>
<thead>
<tr>
<th>Application Output</th>
<th>HTML Source (partial/simplified)</th>
</tr>
</thead>
</table>
| Site search        | `<html>
|                    | <head>
|                    | <title>Site search</title>
|                    | </head>
|                    | <body>
|                    | <center><b>Site search</b></center>
|                    | <form action="sitesearch.php" method="GET">
|                    |   <input name="search" type="text">
|                    |   <br>
|                    |   <input type="submit" value="/f.ligind">
|                    | </form>
|                    | ...<br>
|                    | </body>
|                    | </html>` |

At this stage however, the attack only reflects on the attacker themselves, so an additional step is required:

This application's form sends data in an HTTP GET request. Therefore, it is also possible to directly pass the malicious search string in the URL:

```html
http://www.my-site.com/sitesearch.php?search=\<script\>alert(\"Pwned by XSS\")\>Test
```

This link can now be easily hidden in an email, crafted to lure the victim into clicking on it. Attackers sometimes attempt to add an additional layer of obfuscation by not delivering the JavaScript payload directly, but rather insert an IFRAME pointing to a server owned by them, which then delivers the JavaScript.
2 Stored/Persistent XSS

In stored/persistent XSS, the attacker manages to insert code into the database, for example via data submitted via a form. Consider the following scenario:
A simplistic web-based application asks users to provide their name for a “Who was here” list.

<table>
<thead>
<tr>
<th>Application Output</th>
<th>HTML Source (partial/simplified)</th>
</tr>
</thead>
</table>
| **The Who-Was-Here-Page**

[your name]

... was here

Visitors since Jan 1st, 2014:
Bart Simpson
Thomas Magnum
Parker Lewis
Ferris Bueller
...

Whenever a user enters his/her name, the application logic appends “<br>“ to it and adds it to a file stored on the server and then regenerates the page, embedding the content of the file (marked yellow) in the HTML source.

Now, consider a user entering the following JavaScript directive as their “name”:

```html
<script>alert("Pwned by XSS")</script>
```

As the attacker’s input will be embedded in all future versions of the page, the JavaScript directive will be executed in the browser of every subsequent visitor to the application, resulting in an ‘alert box’ being displayed.
<table>
<thead>
<tr>
<th>Application Output</th>
<th>HTML Source (partial/simplified)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>The Who-Was-Here-Page</strong></td>
<td></td>
</tr>
<tr>
<td>your name</td>
<td></td>
</tr>
<tr>
<td>Pwned by XSS</td>
<td></td>
</tr>
<tr>
<td>Visitors since Jan 1st, 2010</td>
<td></td>
</tr>
<tr>
<td>Bart Simpson</td>
<td></td>
</tr>
<tr>
<td>Thomas Magnum</td>
<td></td>
</tr>
<tr>
<td>Parker Lewis</td>
<td></td>
</tr>
<tr>
<td>Ferris Bueller</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>

### 3 DOM-based XSS

DOM-based XSS is the latest type of XSS attack. Unlike stored and reflected XSS, this type of attack does not modify the HTML code delivered by the application by the server. Instead, it relies on the variables provided to the client side JavaScript engine by the browser and the manipulation of these variables.

Let's consider the following static HTML code:

```html
<html>
<title>Welcome!</title>

Hi
<script>
var pos=document.URL.indexOf("name=")+5;
document.write(document.URL.substring(pos,document.URL.length));
</script>
<br>
Welcome to our system
...
</html>
```
The embedded script accesses the URL of the HTML document via the document. URL variable, then dynamically rewrites the document in the client's browser using document.write in order to incorporate the value of the "name" parameter, provided as part of the URL. Normally, this HTML page would be used for welcoming the user.

A URL such as:

http://www.vulnerable.site/welcome.html?name=Joe

would therefore result in the following benign HTML code rendered by the browser:

```html
<html>
<title>Welcome!</title>
Hi Joe
<br>
Welcome to our system
...
</html>
```

However, a request URL such as:

http://www.vulnerable.site/welcome.html?name=<script>alert(“Pwned by XSS”)</script>

results in an XSS condition, as the HTML code would look like this:

```html
<html>
<title>Welcome!</title>
Hi <script>alert(“Pwned by XSS”)</script>
<br>
Welcome to our system
...
</html>
```

As DOM-based XSS exploits browser flaws, the exposure to these attacks depends on the browser used to access the application.
XSS attack outcomes

XSS vulnerabilities are among the most prevalent problems in SAP applications. They account for roughly 25% of all SAP security notes and range high in terms of criticality. Attackers use XSS attacks primarily for the following:

- Modifying the application’s appearance. This could range from defacing the application to subtle display of non-legitimate information.
- Identity theft: by adding functional components to the application, such as an additional authentication request used to harvest legitimate users’ credentials.
- Taking over of an established authenticated session, which can be achieved by extracting the session ID and/or session cookie.
- Redirection: legitimate users can be redirected to a malicious site by attempting to infect them with malware.

SQL injection

In most modern web-based applications, the web page’s HTML, or parts of it, is generated dynamically with information retrieved from a database. SQL, the structured query language, is used to articulate the questions to the database, in order to generate the desired output. SQL injection attacks aim to manipulate these queries and add SQL commands to the application through user input used to customize the SQL queries.

Imagine a simple web-based application in which you enter the make and model of some device (let’s assume a flat-screen TV) and the application displays its technical data.

So, you enter:

SONY W850B

In order to find the right set of technical specifications, the application uses a pre-defined query string into which the user input is inserted to create the following SQL query:

```sql
SELECT size, resol, 3D, price FROM specs WHERE model='SONY W850B' order by price;
```
The query retrieves the size, resolution, 3D-option and the price of all models called “SONY W850B” and sorts the output based on the price.

In order to understand the attack, one needs to understand that in SQL, statements can be chained when separated by a semicolon and two minuses “--” qualify the rest of the line as a comment that need not be executed.

So, if an attacker enters:

```
XXX' OR 1==1; DROP TABLE specs; --
```

the resulting SQL query string becomes

```
SELECT size, resol, 3D, price FROM specs
WHERE model='XXX' OR 1=1;
DROP TABLE specs; --' order by price;
```

This statement will return details on all TVs in the table. Although model='XXX' will probably never be true, but 1=1 is always true, so the details are returned. The more evil command is then concatenated to the query: “DROP TABLE specs”. It causes the entire table “specs” to be deleted. The remaining part of the original query is ignored, because of the “--”. Obviously, in this example the attacker had information about the layout of the database (i.e. knowing the table name is “specs”), but even if they hadn’t, SQL injections can be used to find out those details before mounting the actual attack.

**SQL injection outcomes**

SQL injections are extremely critical as they can be used to gain access to virtually any information stored in the database. Not only can the attacker read data, they may also be able to manipulate it. The possibilities are endless; they can add administrator accounts, manipulate prices, and delete or falsify data which can lead to flawed or even fatal business decisions.
**Directory Traversal Attacks**

A Directory Traversal Attack is an attack that allows an attacker to traverse or move through one or more forbidden directories to gain access to restricted files.

Such an attack can be carried out by targeting either the web server or the web application. While SAP web servers are not vulnerable, a lot of application components still are.

Application-based directory traversal attacks will attempt to pass a specially crafted parameter to a vulnerable application in order to retrieve a file outside of the application's boundaries.

Consider the following example:

A web-based application, or a part thereof, includes a static HTML file in the output. The URL for that application module is:


An attacker could attempt 2 types of traversal attack:

Absolute path traversal by requesting:

http://www.myserver.com/webapp?ViewFile=/etc/passwd

Relative path traversal by requesting:

http://www.myserver.com/webapp?ViewFile=../../../etc/passwd

SAP servers are vulnerable to yet another form of directory traversal not directly related to web-based applications, but one related to the SAP proprietary SAPCAR archive format.

The SAPCAR format is commonly used to transfer and deploy support packages or other SAP components to servers. However, users or attackers can easily create SAPCAR archives of their own using the SAPCAR tool available on any SAP platform.

Unlike most other archive formats, SAPCAR allows absolute paths to be stored in the archive, while it also allows the storing of files in the archive whose relative positions are higher than the current work directory.
As such, a SAPCAR archive can contain files such as /etc/passwd or ../../../etc/profile. Upon unpacking of the archive, these files would be restored to the respective absolute or relative position.

Assuming the user, or service, unpacking the archive has the relevant access permissions, these files could potentially overwrite critical operating system files and insert new administrator accounts, or modify system startup settings in an effort to weaken or disable security on the system. In a worst-case scenario, an attacker could render a server unusable by overwriting critical operating system binaries.

---

**Open redirect attacks**

As web-based applications have to direct users to a multitude of sub-application modules, or even other applications; so-called redirects and forwards are frequently used and virtually everyone has experienced a redirect at one time or another. For example, when your session with a web-based application expires owing to inactivity, you are redirected to login again before you can continue to work with the application. Typically, the redirection target is coded as a URL parameter, such as:

```
http://www.myserver.com/myapp?url=login
```

Attackers may manipulate this parameter. They may use social engineering techniques to lure victims into clicking on the link, as the server appears to be well-known and trusted, yet they remain fully unaware they will actually be redirected to a malicious site which, in turn, could infect them with malware, steal credentials etc.

```
```

Attackers may obfuscate the redirection target URL by encoding it and/or embedding it within bogus parameters, as shown below:

```
http://www.myserver.com/myapp?appname=jfu77kjjfz?encryption=098382773665?url=%68%74%74%70%3A%2F%2F77%77%77%2E61%6C%69%63%69%6F%75%73%2D%73%69%74%65%2E63%6F%6D%2F%76%69%72%75%73?auth=003988477
```

The bogus data will be ignored by the application and the parameter “url” decodes to the actual malicious URL.
Unrestricted or unchecked file uploads

An increasing number of SAP applications support uploading unstructured content and files of any kind. These files are typically stored in the database or some content repository within the SAP system. Examples include applicant CVs in eRecruiting, proposal documents in SRM, PDFs, or images in KM, etc. From a security perspective, these uploads should be restricted to necessary file types or MIME types only. To ensure accuracy, the required filters need to analyze the files’ actual content and not only relying on the filename extension or the sending browser’s MIME-type declaration.

Without proper checks, uploaded files could carry the following risks:

1 Malware
Malware today is no longer limited to executables. Frequently, malware hides in seemingly benign file formats. Vulnerabilities in the displaying applications, however, can be exploited to execute arbitrary code on the viewing machine. So, while a malware infected file in the SAP repository does not necessarily harm the SAP application, it can damage the organization’s reputation when passed to an outsider. Ultimately, legal action for damages caused by a malware hit originating from an SAP system can lead to additional financial penalties.

2 Active content
Many file formats have the option to embed “interactive” elements. They are not malware by definition, so most virus scanners will not block them. However, for the security of your application, you may want to filter them. Such elements can be macros in office documents, JavaScript or ActiveAction in PDFs, JavaScript in HTML, generic java classes, XSLT, Flash and Silverlight etc.

3 Chameleon files
These files meet the description for multiple file types. One example of such a chameleon file is a GIFAR file:

A GIF image file consists of a file header at the beginning of the file containing an identifier and information about the image size, color palette etc. followed by the actual image data
A ZIP archive, on the other hand, begins with compressed data and has a table of contents at the end of the file.

**ZIP archive**

<table>
<thead>
<tr>
<th>Compressed data</th>
<th>TOC</th>
</tr>
</thead>
</table>

It is important to understand that a lot of common file formats use the ZIP archive format too, e.g. Office Open XML format (docx, pptx, xlsx files) and also Java archives (.jar files)

An attacker can create arbitrary Java code and package it as a Java archive (.jar or other variants). In order to fool content filters, attackers then concatenate the GIF image and the Java archive.

**GIFAR chameleon file**

<table>
<thead>
<tr>
<th>Header</th>
<th>Image Data</th>
<th>Compressed data</th>
<th>TOC</th>
</tr>
</thead>
</table>

The resulting file opens as both a GIF image AND a Java archive - a chameleon so content filters will determine the file type as being GIF, MIME-type image/gif; while the very same file, when opened with java -jar, will lead to the execution of the Java classes embedded in the file. These files become especially dangerous when referenced with an HTML `<APPLET>` tag during an XSS attack. This means an attacker can execute malicious Java code on a victim's machine.
Based on a similar approach, it is simple to hide Java classes in any format that uses the ZIP archive as containers, for example OOOXML Office documents (.docx, .xlsx, .pptx):

```
$ unzip -l Modified_xlsx_file.xlsx
Archive: Modified_xlsx_file.xlsx

<table>
<thead>
<tr>
<th>Length</th>
<th>Date</th>
<th>Time</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1565</td>
<td>01-01-80</td>
<td>00:00</td>
<td>[Content_Types].xml</td>
</tr>
<tr>
<td>735</td>
<td>01-01-80</td>
<td>00:00</td>
<td>_rels/.rels</td>
</tr>
<tr>
<td>980</td>
<td>01-01-80</td>
<td>00:00</td>
<td>xl/_rels/workbook.xml.rels</td>
</tr>
<tr>
<td>658</td>
<td>01-01-80</td>
<td>00:00</td>
<td>xl/workbook.xml</td>
</tr>
<tr>
<td>7079</td>
<td>01-01-80</td>
<td>00:00</td>
<td>xl/theme/theme1.xml</td>
</tr>
<tr>
<td>628</td>
<td>01-01-80</td>
<td>00:00</td>
<td>xl/worksheets/sheet2.xml</td>
</tr>
<tr>
<td>628</td>
<td>01-01-80</td>
<td>00:00</td>
<td>xl/worksheets/sheet3.xml</td>
</tr>
<tr>
<td>816</td>
<td>01-01-80</td>
<td>00:00</td>
<td>xl/worksheets/sheet1.xml</td>
</tr>
<tr>
<td>181</td>
<td>01-01-80</td>
<td>00:00</td>
<td>xl/sharedStrings.xml</td>
</tr>
<tr>
<td>1260</td>
<td>01-01-80</td>
<td>00:00</td>
<td>xl/styles.xml</td>
</tr>
<tr>
<td>842</td>
<td>01-01-80</td>
<td>00:00</td>
<td>docProps/app.xml</td>
</tr>
<tr>
<td>625</td>
<td>01-01-80</td>
<td>00:00</td>
<td>docProps/core.xml</td>
</tr>
<tr>
<td>940</td>
<td>01-01-80</td>
<td>00:00</td>
<td>docProps/custom.xml</td>
</tr>
<tr>
<td>18406</td>
<td>02-07-12</td>
<td>20:39</td>
<td>Pwned.class</td>
</tr>
</tbody>
</table>
```

35343 files
As with active content, such a file will not be detected by standard virus scanners as being malicious.

Prevalence of vulnerabilities and content-based attacks in SAP applications

SAP is constantly improving the functionality and security of their products. Furthermore, SAP internal and external security researchers constantly analyze SAP products and disclose vulnerabilities. One of these external security researchers, the Moscow-based company ERPscan, conducted a major task to quantify SAP security-related topics. Their white paper "12 years of SAP Security in Figures – a Global Survey" is a recommended read. ERPscan analyzed SAP security notes up until very recently, and they found that content-based attack vectors (XSS, SQL Injections, Directory Traversal) are 3 of the top 4 vulnerabilities and alone they account over 50% of all SAP vulnerabilities.

In absolute figures and for just the past 24 months, this translates to:

- 20 XSS vulnerabilities
- 10 directory traversal vulnerabilities
- 6 SQL injection vulnerabilities
- 3 Open redirect vulnerabilities

Numerous other content-related vulnerabilities, including cross-site request forgery and unchecked uploads.

With only two exceptions, every single one of these vulnerabilities was addressed by SAP with a “Correction with high priority” and customers would have been highly encouraged to deploy the fixes immediately. That is a lot of patching for mission-critical systems with very high availability requirements and SLAs, and additionally, these numbers refer to “out-of-the-box” SAP applications only. The number of vulnerabilities in custom-built applications is most definitely much, much greater.

These numbers only reflect vulnerabilities found and disclosed by white-hat security researchers. The nature of SAP systems and the value of the information processed by them suggest the black-hat community probably knows and trades information on zero-day SAP vulnerabilities as well as the malicious code needed to exploit them, enabling paying clients to attack specifically targeted SAP systems.

**Customized SAP applications introduce more vulnerabilities**

Lastly, the figures above concern out-of-the-box SAP products only. However, most customers will heavily customize and extend these products. One can safely assert that the same type of vulnerabilities do exist in customized SAP applications in numbers by far exceeding the figures above.
How to protect your SAP applications

The most efficient way to protect applications from content-based attacks is through proper programming. All vulnerabilities described in this whitepaper are the result of sloppy programming, missing so-called “sanitation” of user input. Consequently, fixing all user input processing functions in your applications will solve this problem long-term.

Unfortunately, when it comes to SAP applications, they are extremely large, extremely complex, and they are not entirely available in source code. As a result, fixing vulnerabilities depends on SAP learning about every single one of them, and then addressing each one with a specific fix – which in turn will result in massive patching activity for customers.

An alternative approach to sanitizing user input in the application is to look at all user input and parameters at server level – before data is processed by the applications. BowBridge “ApplicationSecurity Bridge for SAP solutions” takes exactly that approach. It hooks into the SAP Internet Communication Manager (ICM) and monitors all URLs and parameters passed to the application server for content-based threats.

In ApplicationSecurity Bridge for SAP Solutions, custom-built analysis modules search for:

- Cross-Site Scripting
- SQL Injections
- Open Redirects
- Directory Traversals
and block the offending data when detected. The SAP application remains uncompro-
mised, safe and available.

In order to secure file uploads, BowBridge developed AntiVirus Bridge for SAP Solutions, a SAP-certified solution that has become the de-facto standard for file-based content security for SAP applications. It leverages the NetWeaver “Virus Scan Interface” and provides extensive filtering capabilities, such as MIME-type whitelists and blacklists, detection of active content and chameleon files, while also handling SAPCAR archives, including detection of directory traversal attacks in archives. Lastly, AntiVirus Bridge comes with two leading virus scan engines built-in (McAfee and Sophos), in order to detect any malware embedded in files that are being uploaded into the SAP application.

Summary

When SAP experts address the topic of SAP security, they primarily think of roles and permissions, segregation of duties, and security mechanisms required by regulatory compliance frameworks such as SOX, HIPAA and PCI. Unfortunately for many SAP customers, content security for SAP applications is largely a blind spot in their security considerations. Regrettably this creates significant potential for disrupting the stability, availability and privacy of the SAP environment, ultimately putting a company’s reputation at stake. Neglecting these risks may have unforeseen consequences and a negative impact on any business or organization.