LUCIDEUS

Practical Insight Into Injections

Hanut Kumar Arora

Hanutarora@gmail.com

INTRODUCTION

PURPOSE

Injections are one of the most basic and common attacks and it is at the first position of the OWASP Top 10 Attacks. An unprotected website is a security risk to customers, other businesses, and public/government sites. It allows for the spread and escalation of malware, attacks on other websites, and even attacks against national targets and infrastructure. A single security breach could be a death-knell for a small business. By some estimates, about 30,000 to 50,000 websites get hacked every day. These numbers are growing every day and the importance of website security is increasing rapidly. Web security must be a primary focus in a time when a person can make purchases, pay bills, and even access bank accounts all from the convenience of a web browser. A lot of sensitive information is transmitted over the internet, which enables people to carry out all business from a web browser. So, to be aware of these attacks and protect our websites we should have a proper understanding of the working of these attacks and the preventive measures.

SCOPE

Untrusted data is most often data that comes from the HTTP request, in the form of URL parameters, form fields, headers, or cookies. But data that comes from databases, web services, and other sources is frequently untrusted from a security perspective. That is, untrusted data is the input that can be manipulated to contain a web attack payload. The OWASP Code Review Guide has a decent list of methods that return untrusted data in various languages, but you should be careful about your own methods as well.

OVERVIEW OF THE DOCUMENT

The project overview describes the attack Injections. It describes the meaning, working, implementation, the impact of the vulnerabilities, what can it be used for and how can they be found. Injections are very common, and Injections is probably the most frequently occurring web security vulnerability. This project aims to provide mitigation before and after the attack.

HOST HEADER INJECTION

DESCRIPTION

A web server commonly hosts several web applications on the same IP address, referring to each application via the virtual host. In an incoming HTTP request, web servers often dispatch the request to the target virtual host based on the value supplied in the Host header. Without proper validation of the header value, the attacker can supply invalid input to cause the web server to:

- dispatch requests to the first virtual host on the list
- cause a redirect to an attacker-controlled domain
- perform web cache poisoning
- manipulate password reset functionality

HOST HEADER BYPASS

Initial testing is as simple as supplying another domain (i.e. attacker.com) into the Host header field. It is how the web server processes the header value that dictates the impact. The attack is valid when the web server processes the input to send the request to an attacker-controlled host that resides at the supplied domain, and not to an internal virtual host that resides on the web server.

```
GET / HTTP/1.1
Host: www.attacker.com
[...]
```

In the simplest case, this may cause a 302 redirect to the supplied domain.

```
HTTP/1.1 302 Found
[...]
Location: http://www.attacker.com/login.php
```

Alternatively, the web server may send the request to the first virtual host on the list.

X-FORWARDED-HOST HEADER BYPASS

In the event that Host header injection is mitigated by checking for invalid input injected via the Host header, you can supply the value to the X-Forwarded-Host header.

```
GET / HTTP/1.1
Host: www.example.com
X-Forwarded-Host: www.attacker.com
...
```

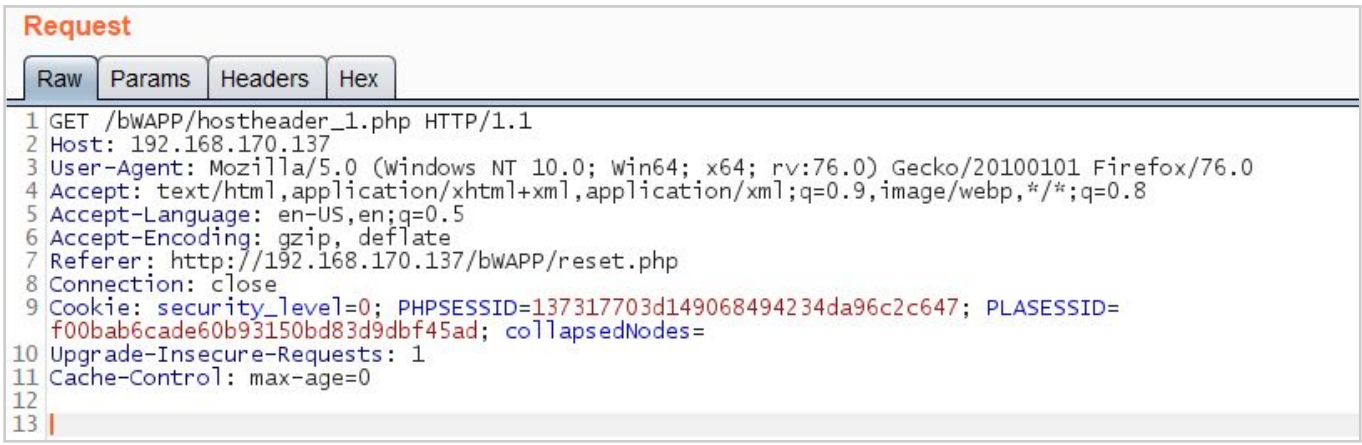
Potentially producing client-side output such as:

```
<link src="http://www.attacker.com/link" />
```

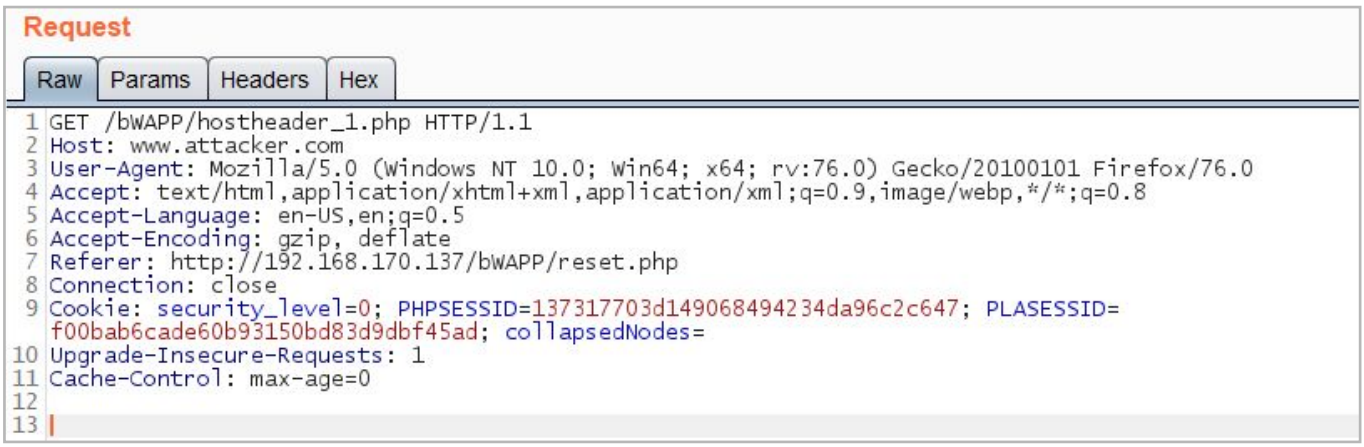
Once again, this depends on how the web server processes the header value.

WEB CACHE POISONING

Using this technique, an attacker can manipulate a web-cache to serve poisoned content to anyone who requests it. This relies on the ability to poison the caching proxy run by the application itself, CDNs, or other downstream providers. As a result, the victim will have no control over receiving the malicious content when requesting the vulnerable application.



Original Request to the Web Server



Request with Modified Host header value

The following Fig 2.2 will be served from the web cache instead of Fig 2.3, when a victim visits the vulnerable application.

```
20    <link rel="stylesheet" type="text/css" href="http://192.168.170.137/bWAPP/stylesheets/stylesheet.css" media="screen" />
21    <link rel="shortcut icon" href="images/favicon.ico" type="image/x-icon" />
22
23    <!--<script src="//html5shiv.googlecode.com/svn/trunk/html5.js"></script>-->
24    <script src="http://192.168.170.137/bWAPP/js/html5.js">
      </script>
```

Response to the Original Request

```
20    <link rel="stylesheet" type="text/css" href="http://www.attacker.com/bWAPP/stylesheets/stylesheet.css" media="screen" />
21    <link rel="shortcut icon" href="images/favicon.ico" type="image/x-icon" />
22
23    <!--<script src="//html5shiv.googlecode.com/svn/trunk/html5.js"></script>-->
24    <script src="http://www.attacker.com/bWAPP/js/html5.js">
      </script>
```

Response to the Modified Request

PASSWORD RESET POISONING

It is common for password reset functionality to include the Host header value when creating password reset links that use a generated secret token. If the application processes an attacker-controlled domain to create a password reset link, the victim may click on the link in the email and allow the attacker to obtain the reset token, thus resetting the victim's password.

... Email snippet ...
Click on the following link to reset your password:
http://192.168.170.1:1234/index.php?module=Login&action=resetPassword&token=<SECRET_TOKEN>
... Email snippet ...

REMEDIATION

Mitigating against the host header injection is simple — don't trust the host header. However in some cases, this is easier said than done (especially situations involving legacy code). If you must use the host header as a mechanism for identifying the location of the web server, it's highly advised to make use of a whitelist of allowed hostnames.

SQL INJECTION

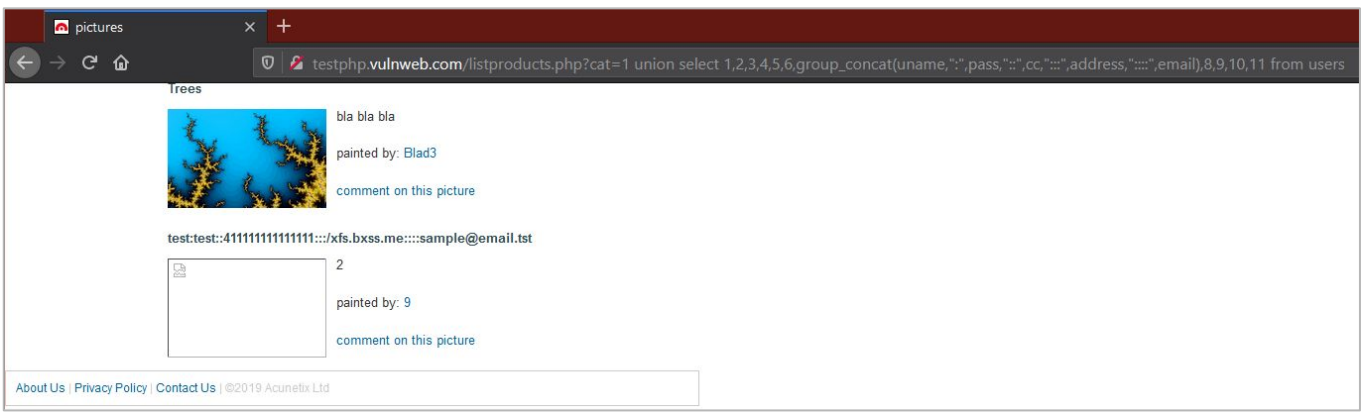
DESCRIPTION

SQL injection is a web security vulnerability that allows an attacker to interfere with the queries that an application makes to its database. It generally allows an attacker to view data that they are not normally able to retrieve. This might include data belonging to other users, or any other data that the application itself is able to access. In many cases, an attacker can modify or delete this data, causing persistent changes to the application's content or behavior.

In some situations, an attacker can escalate an SQL injection attack to compromise the underlying server or other back-end infrastructure, or perform a denial-of-service attack.

UNION BASED SQL INJECTION

The UNION operator is used in SQL injections to join a query, purposely forged by the tester, to the original query. The result of the forged query will be joined to the result of the original query, allowing the tester to obtain the values of columns of other tables.



http://testphp.vulnweb.com/listproducts.php?cat=1 union select 1,2,3,4,5,6,group_concat(uname,":",pass,"::",cc,":::",address,"::::",email),8,9,10,11 from users

BLIND SQL INJECTION

Many instances of SQL injection are blind vulnerabilities. This means that the application does not return the results of the SQL query or the details of any database errors within its responses. Blind vulnerabilities can still be exploited to access unauthorized data, but the techniques involved are generally more complicated and difficult to perform.

1. **Boolean Based Blind SQL Injection**

   Many instances of SQL injection are blind vulnerabilities. This means that the application does not return the results of the SQL query or the details of any database errors within its responses. Blind vulnerabilities can still be exploited to access unauthorized data, but the techniques involved are generally more complicated and difficult to perform.

2. **Time Based Blind SQL Injection**

   Time Based SQL Injection is the subcategory of Blind Based SQL Injection in which when we input a Query. They are often use to extracts the data when there no other way to retrieve the data from the database while executing a query in the database which creates a time delay if the query is right depending on the time it takes to get the server response. As you can guess, this type of inference approach is particularly useful for blind injection attacks. It is basically used by using queries which results in a delay of response.
   We basically use the functions such as :

```
sleep()
delay()
hibernate()
```

Basic syntax

```
select if(expression,true,false)
```

ERROR BASED SQL INJECTION

You can change the logic of the query to trigger a detectable difference in the application's response depending on the truth of a single condition. This might involve injecting a new condition into some Boolean logic, or conditionally triggering an error such as a divide-by-zero.

1. **Double Query Error Based SQL Injection**

   In this, we get the answers in the form of errors or we can say Dumping everything in the form of sql error.

AUTOMATED TOOLS FOR SQL INJECTION

SQLMAP is one of the most popular tools for Automated SQL Injection.

```
$ python sqlmap.py -u "http://debiandev/sqlmap/mysql/get_int.php?id=1" --batch
        ___
       __H__
 ___ ___[,]_____ ___ ___  {1.3.4.44#dev}
|_ -| . [,]     | .'| . |
|___|_  [,]_|_|_|__,|  _|
      |_|V...       |_|   http://sqlmap.org

[!] legal disclaimer: Usage of sqlmap for attacking targets without prior mutual consent i
s illegal. It is the end user's responsibility to obey all applicable local, state and fed
eral laws. Developers assume no liability and are not responsible for any misuse or damage
 caused by this program

[*] starting @ 10:44:53 /2019-04-30/

[10:44:54] [INFO] testing connection to the target URL
[10:44:54] [INFO] heuristics detected web page charset 'ascii'
[10:44:54] [INFO] checking if the target is protected by some kind of WAF/IPS
[10:44:54] [INFO] testing if the target URL content is stable
[10:44:55] [INFO] target URL content is stable
[10:44:55] [INFO] testing if GET parameter 'id' is dynamic
[10:44:55] [INFO] GET parameter 'id' appears to be dynamic
[10:44:55] [INFO] heuristic (basic) test shows that GET parameter 'id' might be injectable
 (possible DBMS: 'MySQL')
```

SQLmap

REMEDIATION

Most instances of SQL injection can be prevented by using parameterized queries (also known as prepared statements) instead of string concatenation within the query.

Parameterized queries can be used for any situation where untrusted input appears as data within the query, including the WHERE clause and values in an INSERT or UPDATE statement. They can't be used to handle untrusted input in other parts of the query, such as table or column names, or the ORDER BY clause. Application functionality that places untrusted data into those parts of the query will need to take a different approach, such as white-listing permitted input values, or using different logic to deliver the required behavior.

For a parameterized query to be effective in preventing SQL injection, the string that is used in the query must always be a hard-coded constant, and must never contain any variable data from any origin. Do not be tempted to decide case-by-case whether an item of data is trusted, and continue using string concatenation within the query for cases that are considered safe. It is all too easy to make mistakes about the possible origin of data, or for changes in other code to violate assumptions about what data is tainted.

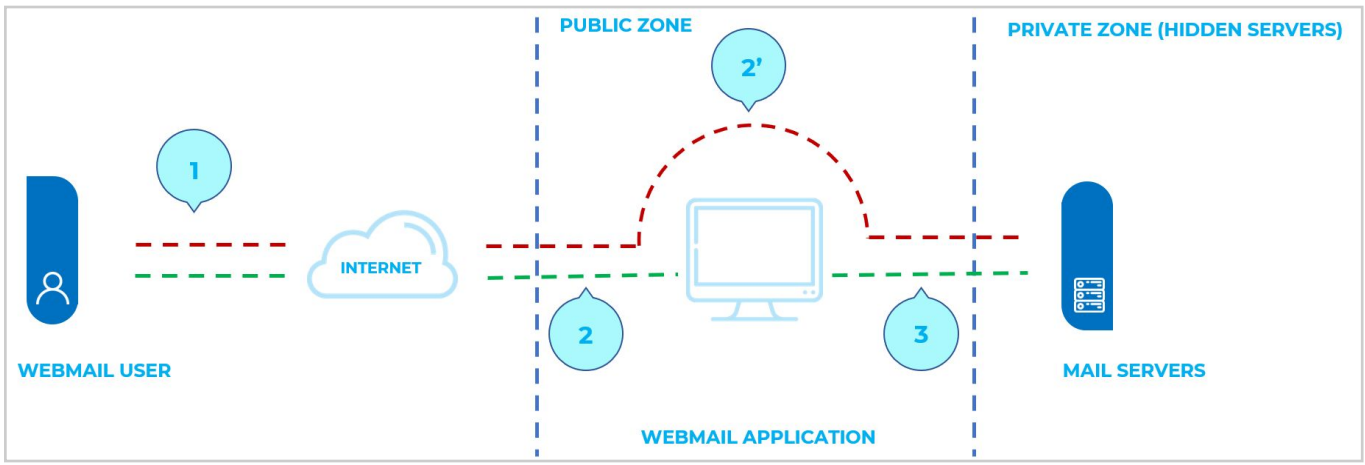
SMTP INJECTION

DESCRIPTION

This threat affects all applications that communicate with mail servers (IMAP/SMTP), generally webmail applications. The aim of this test is to verify the capacity to inject arbitrary IMAP/SMTP commands into the mail servers, due to input data not being properly sanitized.

The IMAP/SMTP Injection technique is more effective if the mail server is not directly accessible from the Internet. Where full communication with the backend mail server is possible, it is recommended to conduct direct testing.

An IMAP/SMTP Injection makes it possible to access a mail server which otherwise would not be directly accessible from the Internet. In some cases, these internal systems do not have the same level of infrastructure security and hardening that is applied to the front-end web servers. Therefore, mail server results may be more vulnerable to attacks by end users (see the scheme presented in Figure 1).



Communication with the mail servers using the IMAP/SMTP Injection technique

Step 1 and 2 is the user interacting with the webmail client, whereas step 2' is the tester bypassing the webmail client and interacting with the back-end mail servers directly.

This technique allows a wide variety of actions and attacks. The possibilities depend on the type and scope of injection and the mail server technology being tested.

Some examples of attacks using the IMAP/SMTP Injection technique are:

- Exploitation of vulnerabilities in the IMAP/SMTP protocol
- Application restrictions evasion
- Anti-automation process evasion
- Information leaks
- Relay/SPAM

DEMO

```
POST /contact.php HTTP/1.1
Host: www.example2.com

name=Anna Smith&replyTo=anna@example.com&message=Hello
```

A typical genuine POST request

```
POST /contact.php HTTP/1.1
Host: www.example2.com

name=Best Product\nbcc: everyone@example3.com&replyTo=blame_anna@example.com&message=Buy my product!
```

An attacker could abuse this contact form by sending this POST request

REMEDIATION

Validate that user input conforms to a whitelist of safe characters before placing it into email headers. In particular, input containing newlines and carriage returns should be rejected. Alternatively, consider switching to an email library that automatically prevents such attacks.

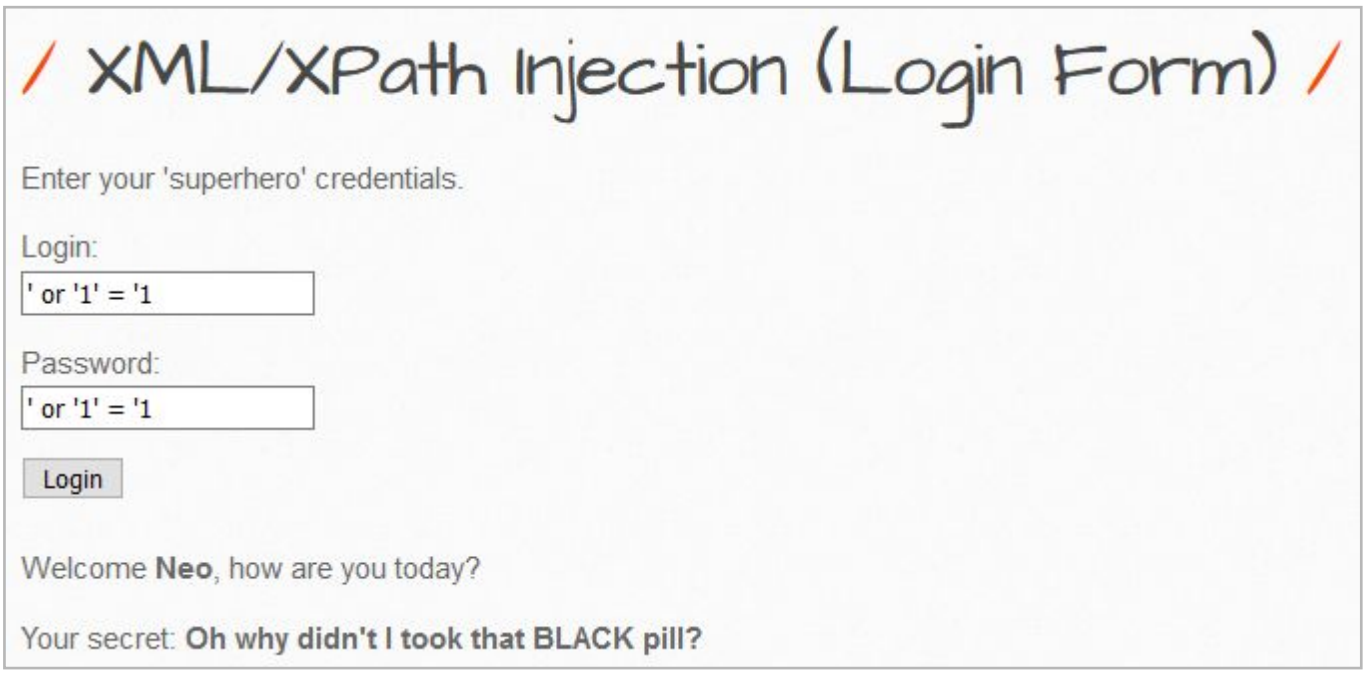
XPATH INJECTION

DESCRIPTION

XPath Injection attacks occur when a website uses user-supplied information to construct an XPath query for XML data. By sending intentionally malformed information into the web site, an attacker can find out how the XML data is structured, or access data that he may not normally have access to. He may even be able to elevate his privileges on the web site if the XML data is being used for authentication (such as an XML based user file).

Querying XML is done with XPath, a type of simple descriptive statement that allows the XML query to locate a piece of information. Like SQL, you can specify certain attributes to find, and patterns to match. When using XML for a web site it is common to accept some form of input on the query string to identify the content to locate and display on the page. This input must be sanitized to verify that it doesn't mess up the XPath query and return the wrong data.

DEMO



XPATH Injection

REMEDIATION

You need to use a parameterized XPath interface if one is available, or escape the user input to make it safe to include in a dynamically constructed query. If you are using quotes to terminate untrusted input in a dynamically constructed XPath query, then you need to escape that quote in the untrusted input to ensure the untrusted data can't try to break out of that quoted context.

OS COMMAND INJECTION

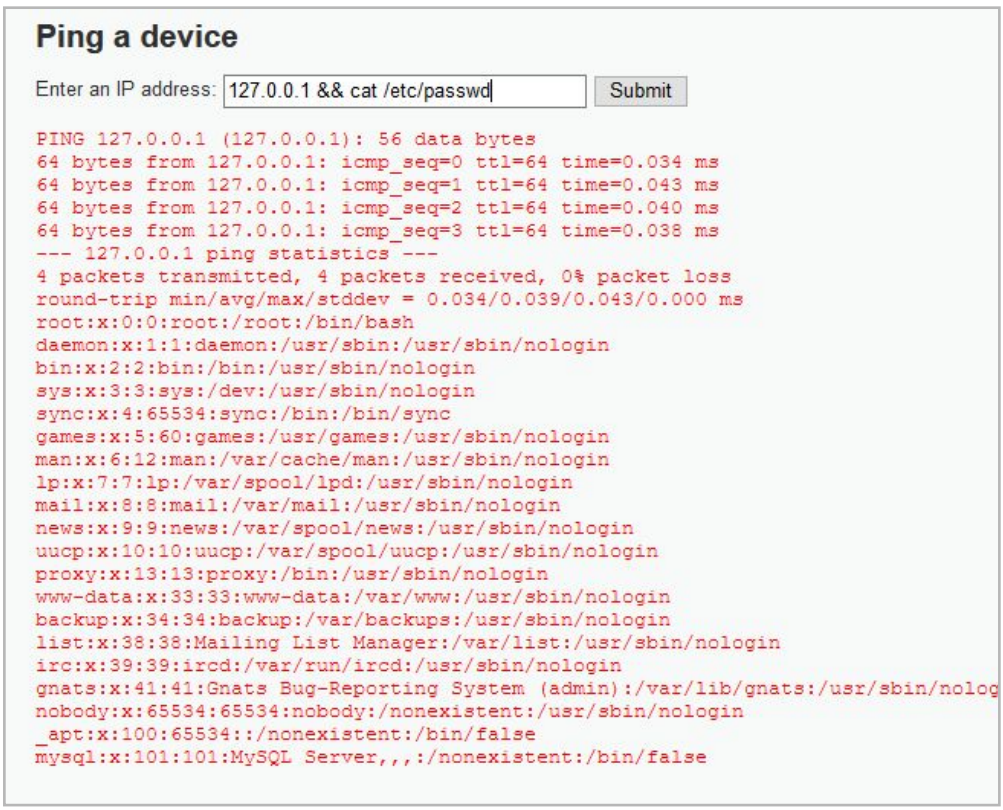
DESCRIPTION

Command injection is an attack in which the goal is execution of arbitrary commands on the host operating system via a vulnerable application. Command injection attacks are possible when an application passes unsafe user supplied data (forms, cookies, HTTP headers etc.) to a system shell. In this attack, the attacker-supplied operating system commands are usually executed with the privileges of the vulnerable application. Command injection attacks are possible largely due to insufficient input validation.

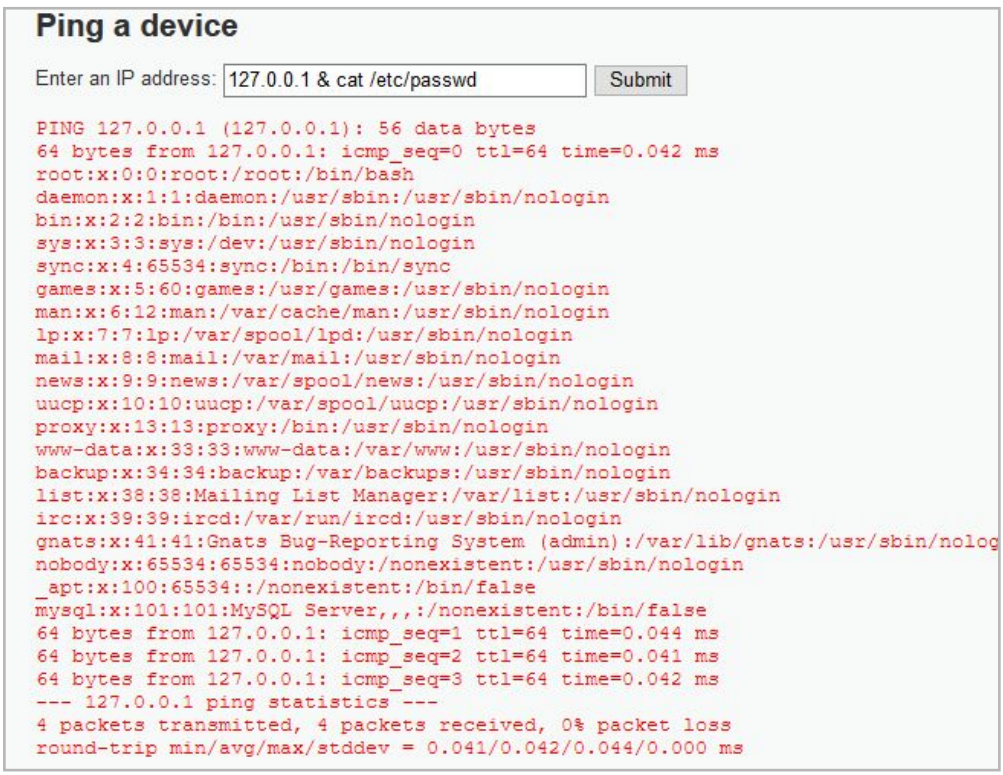
Command injection is an attack in which the goal is execution of arbitrary commands on the host operating system via a vulnerable application.

Command injection attacks are possible when an application passes unsafe user supplied data (forms, cookies, HTTP headers etc.) to a system shell. In this attack, the attacker-supplied operating system commands are usually executed with the privileges of the vulnerable application. Command injection attacks are possible largely due to insufficient input validation.

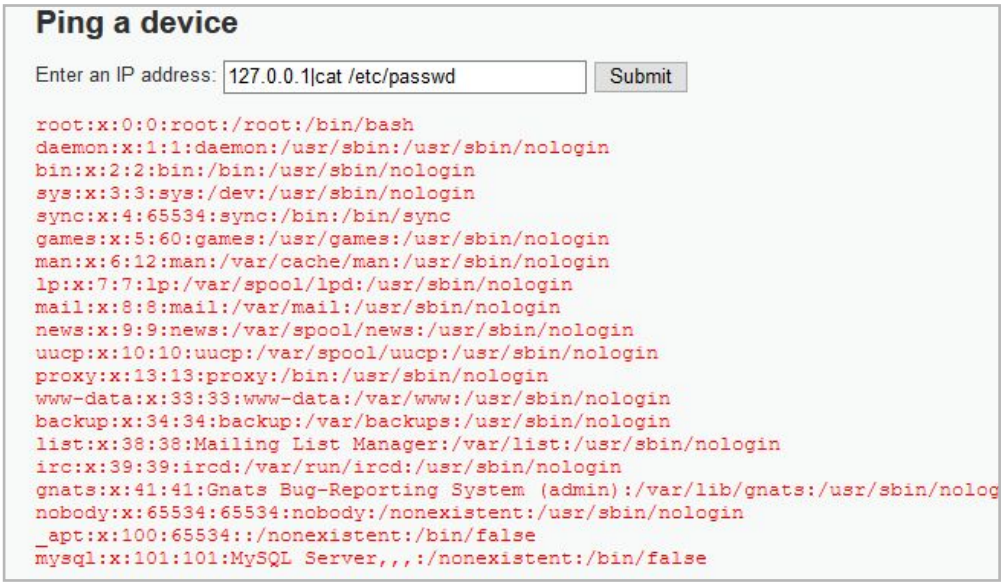
DEMO



OS Command Injection - DVWA - Low Security Level



OS Command Injection - DVWA - Medium Security Level



OS Command Injection - DVWA - High Security Level

REMEDIATION

The simplest and safest one is never to use calls such as shell_exec in PHP to execute any host operating system commands. Instead, you should use the equivalent commands from the programming language. For example, if a developer wants to send mail using PHP on Linux/UNIX, they may be tempted to use the mail command available in the operating system. Instead, they should use the mail() function in PHP.

This approach may be difficult if there is no equivalent command in the programming language. For example, there is no direct way to send ICMP ping packets from PHP. In such cases, you need to use input sanitization before you pass the value to a shell command. As with all types of injections, the safest way is to use a whitelist.

HTML INJECTION

DESCRIPTION

HTML injection is a type of injection vulnerability that occurs when a user is able to control an input point and is able to inject arbitrary HTML code into a vulnerable web page. This vulnerability can have many consequences, like disclosure of a user's session cookies that could be used to impersonate the victim, or, more generally, it can allow the attacker to modify the page content seen by the victims.

This vulnerability occurs when user input is not correctly sanitized and the output is not encoded. An injection allows the attacker to send a malicious HTML page to a victim. The targeted browser will not be able to distinguish (trust) legitimate parts from malicious parts of the page, and consequently will parse and execute the whole page in the victim's context.

1. STORED HTML INJECTION

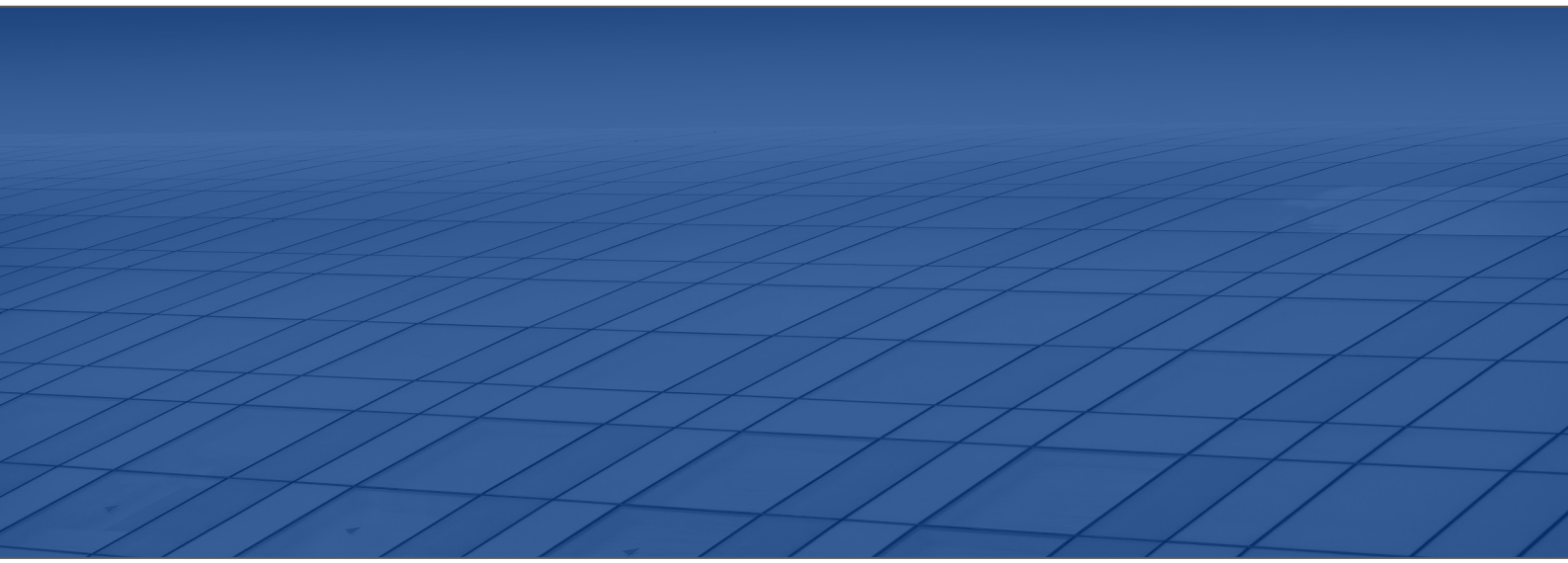
The stored injection attack occurs when malicious HTML code is saved in the web server and is being executed every time when the user calls an appropriate functionality.

2. REFLECTED HTML INJECTION

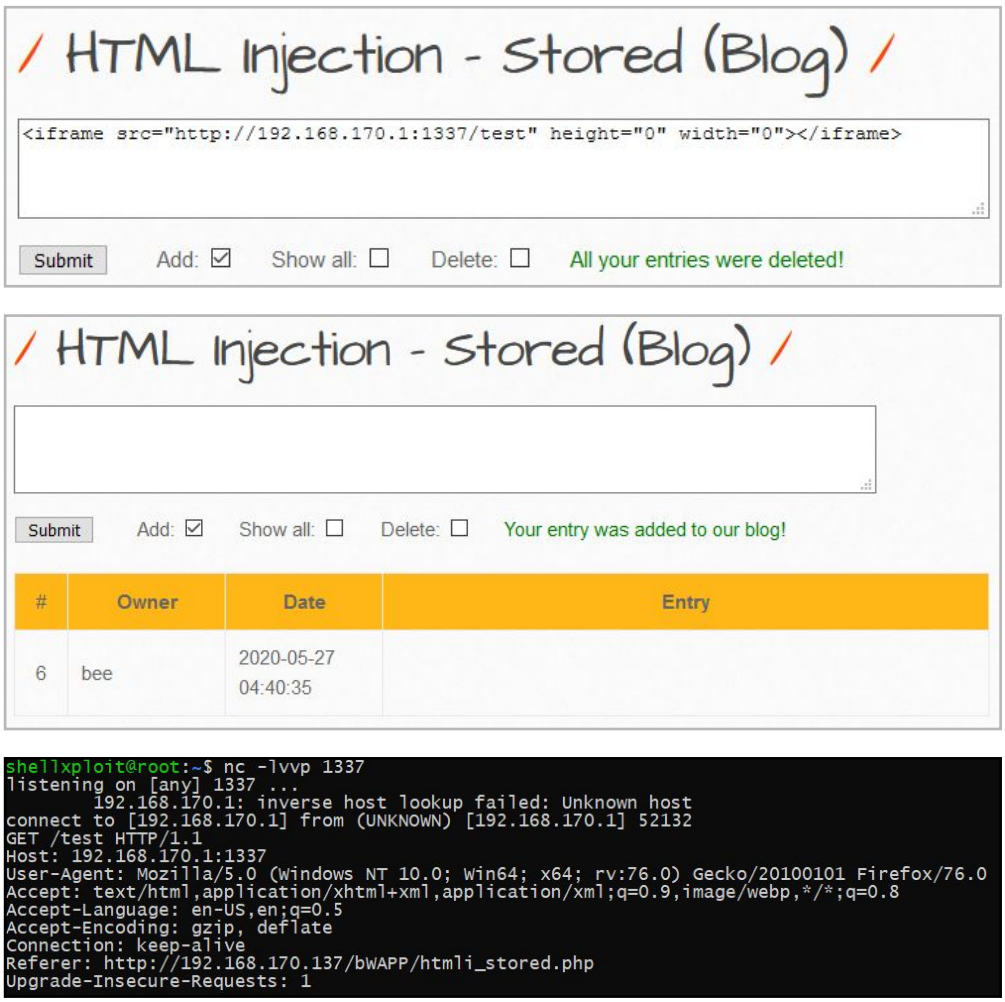
In the reflected injection attack case, malicious HTML code is not being permanently stored on the web server. Reflected Injection occurs when the website immediately responds to the malicious input. This can be again divided into more types:

- Reflected GET
- Reflected POST
- Reflected URL

Reflected GET Injection occurs, when our input is being displayed (reflected) on the website.
Reflected POST HTML Injection occurs when a malicious HTML code is being sent instead of correct POST method parameters.
Reflected URL happens, when HTML code is being sent through the website URL, displayed in the website and at the same time injected to the website.



DEMO



Information about connections who access the page.

REMEDIATION

You need to use a parameterized XPath interface if one is available, or escape the user input to make it safe to include in a dynamically constructed query. If you are using quotes to terminate untrusted input in a dynamically constructed XPath query, then you need to escape that quote in the untrusted input to ensure the untrusted data can't try to break out of that quoted context.

SSI INJECTION

DESCRIPTION

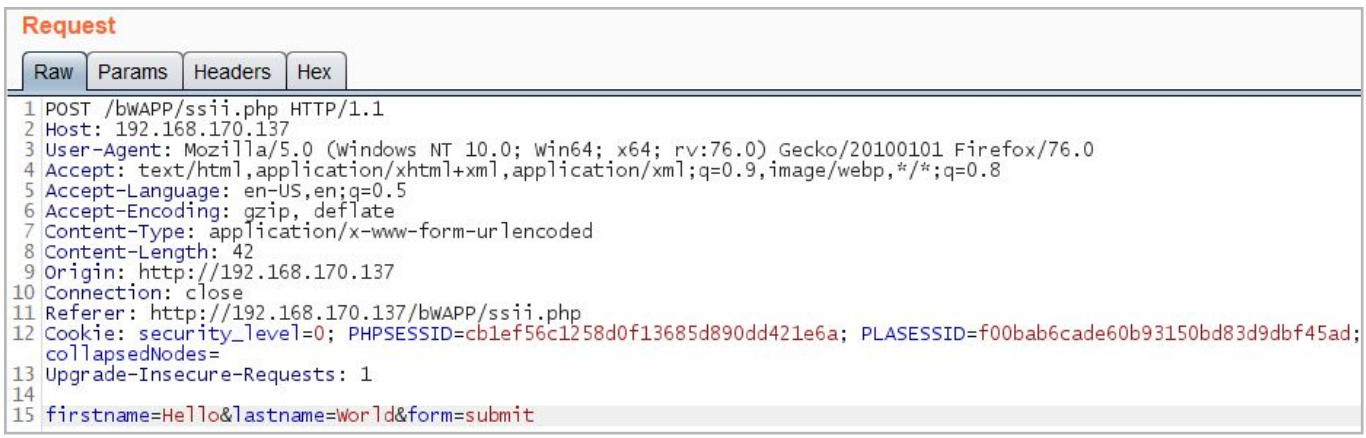
SSIs are directives present on Web applications used to feed an HTML page with dynamic contents. They are similar to CGIs, except that SSIs are used to execute some actions before the current page is loaded or while the page is being visualized. In order to do so, the web server analyzes SSI before supplying the page to the user.

The Server-Side Includes attack allows the exploitation of a web application by injecting scripts in HTML pages or executing arbitrary codes remotely. It can be exploited through manipulation of SSI in use in the application or force its use through user input fields.

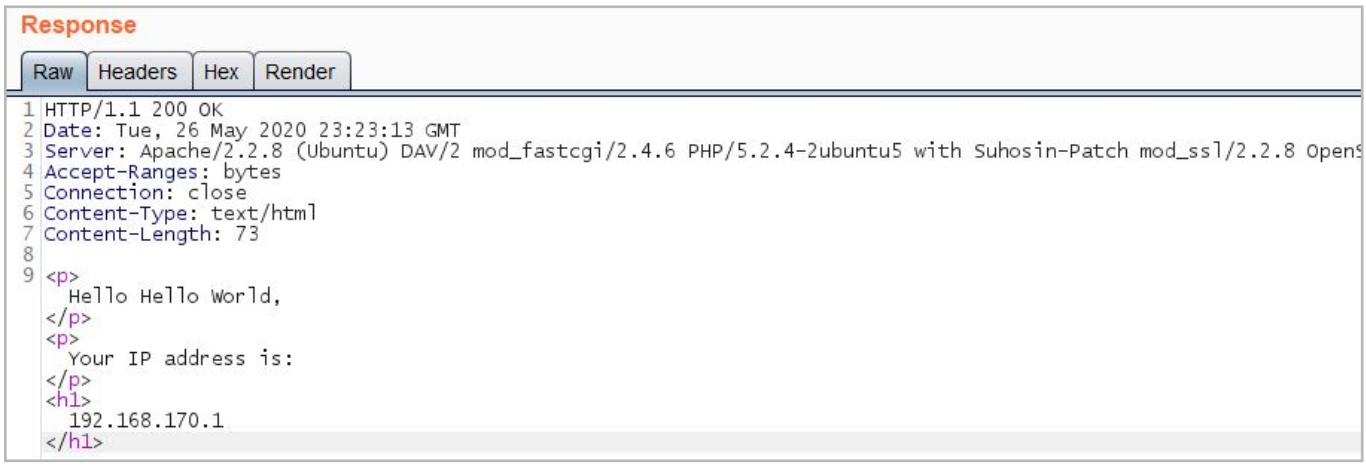
Another way to discover if the application is vulnerable is to verify the presence of pages with extension .stm, .shtm and .shtml. However, the lack of these type of pages does not mean that the application is protected against SSI attacks.

DEMO

This is a set of normal requests and responses.



Information about connections who access the page.



Response to the Original Request

The commands used to inject SSI vary according to the server operational system in use. The following commands represent the syntax that should be used to execute OS commands.

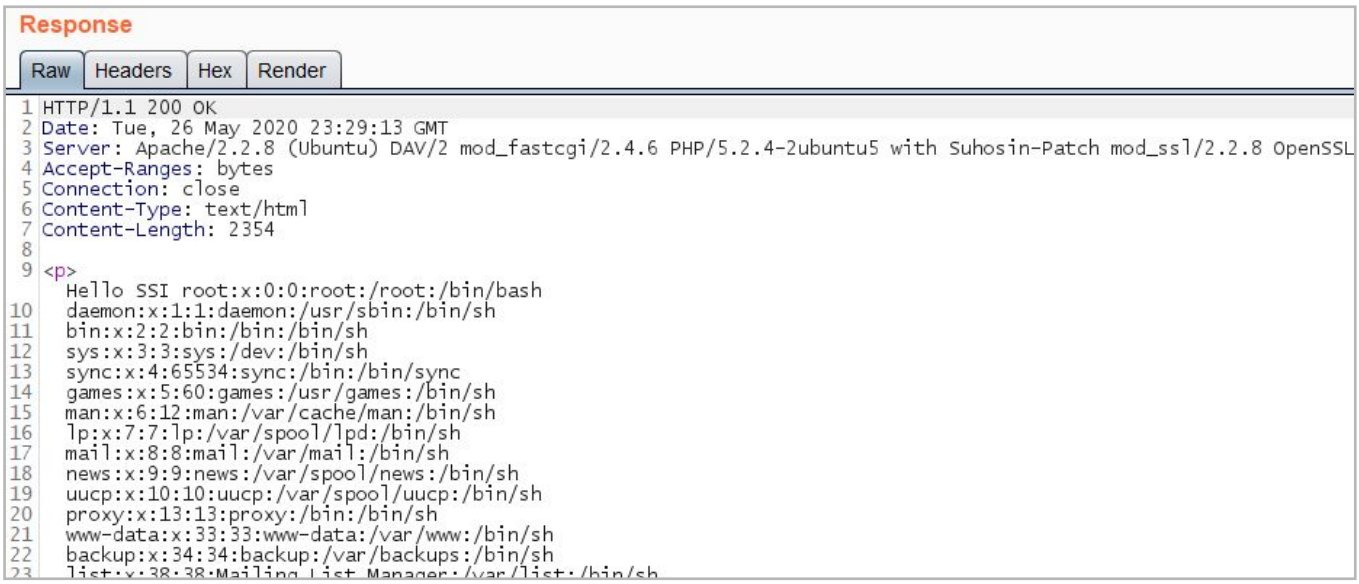
Linux:
List files of directory:

```
<!--#exec cmd="ls" →
```

Windows:
List files of directory:

```
<!--#exec cmd="dir" -->
```

Modified Request with added payload in "lastname"



Response to the Modified Request

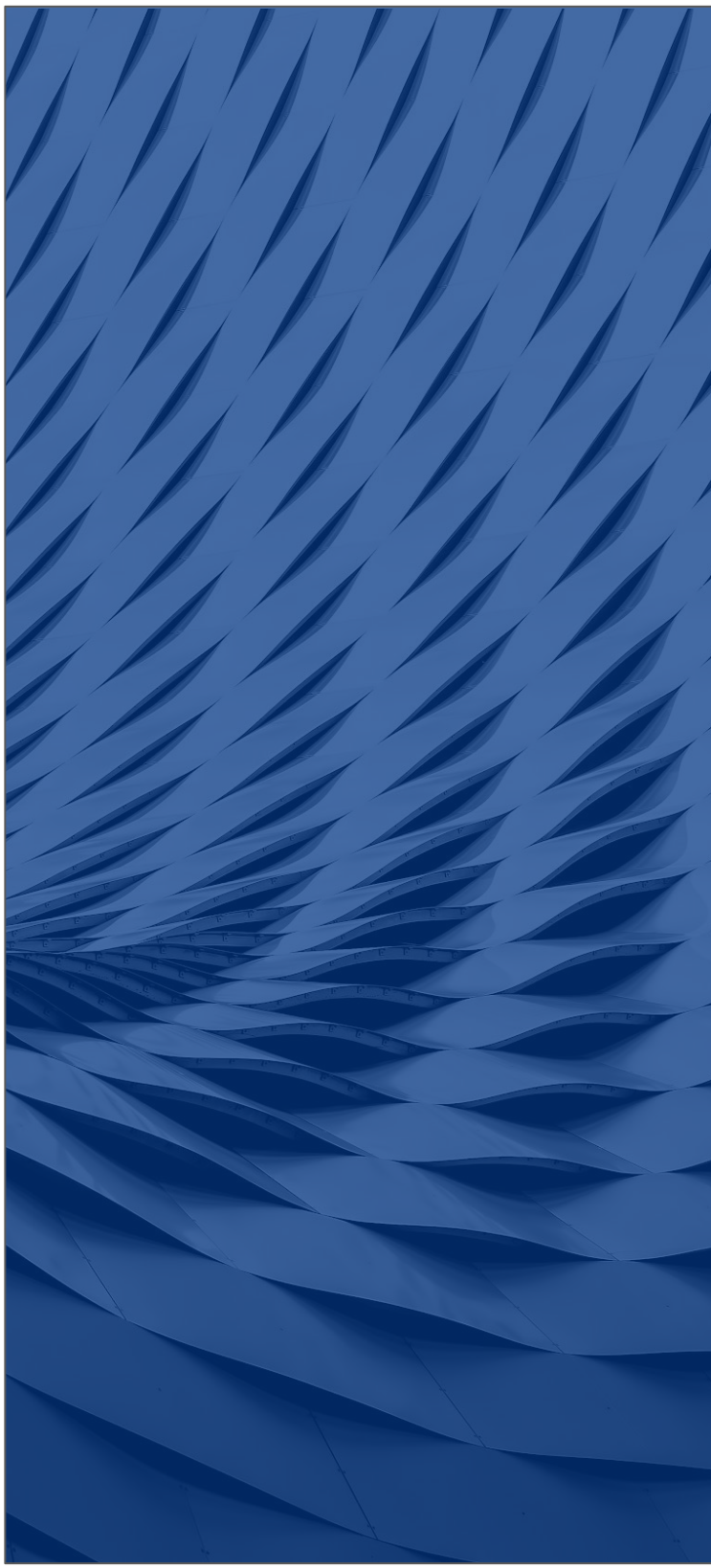
REMEDIATION

If possible, applications should avoid incorporating user-controllable data into pages that are processed for SSI directives. In almost every situation, there are safer alternative methods of implementing the required functionality. If this is not considered feasible, then the data should be strictly validated. Ideally, a whitelist of specific accepted values should be used. Otherwise, only short alphanumeric strings should be accepted. Input containing any other data, including any conceivable SSI metacharacter, should be rejected.

LDAP INJECTION

DESCRIPTION

The Lightweight Directory Access Protocol (LDAP) is used to store information about users, hosts, and many other objects. LDAP injection is a server side attack, which could allow sensitive information about users and hosts represented in an LDAP structure to be disclosed, modified, or inserted. This is done by manipulating input parameters afterwards passed to internal search, add, and modify functions.



A web application could use LDAP in order to let users authenticate or search other users' information inside a corporate structure. The goal of LDAP injection attacks is to inject LDAP search filters metacharacters in a query which will be executed by the application.

Boolean conditions and group aggregations on an LDAP search filter could be applied by using the following metacharacters: &, |, !, =, ~=, >=, <=, *, ()

DEMO

Intended Use of LDAP

Using Metacharacter to get the details of all the users

REMEDIATION

1. Escape all variables using the right LDAP encoding function
2. Escape any untrusted data that is added to any LDAP query
3. Use Frameworks that Automatically Protect from LDAP Injection
4. To minimize the potential damage of a successful LDAP injection attack, you should minimize the privileges assigned to the LDAP binding account in your environment
5. Input validation can be used to detect unauthorized input before it is passed to the LDAP query.


LUCIDEUS™