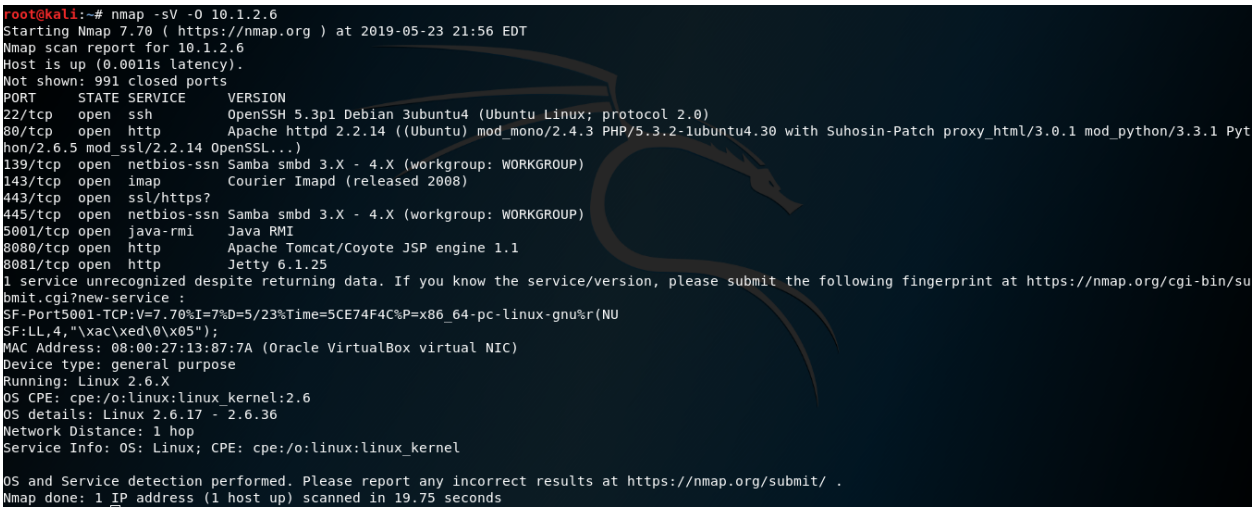
Segurança em Computação
Trabalho Individual IV

Gustavo Figueira Olegário

27 de maio de 2019

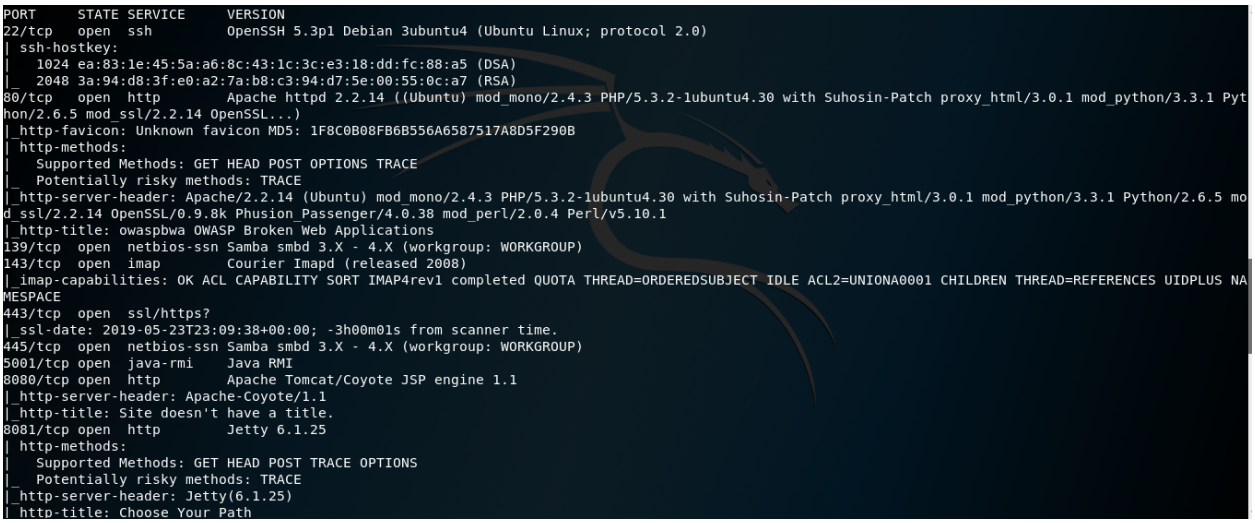
Parte 1 - NMAP

Questão 1



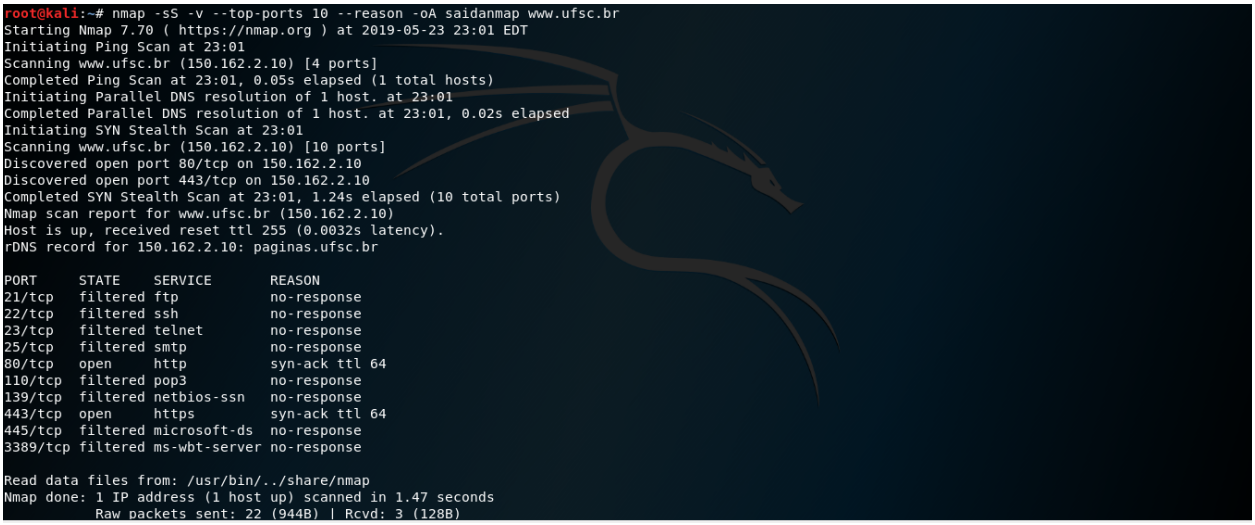
Das informações retornadas, pode se perceber que o programa detectou que a máquina escaneada era um Linux na versão 2.6.x, sendo esse uma distro Debian. Além disso, o programa identificou o MAC address com o valor de: 08:00:27:13:87:7A. Foi detectado também que a máquina estava executando o Apache juntamente com o Tomcat.

Questão 2



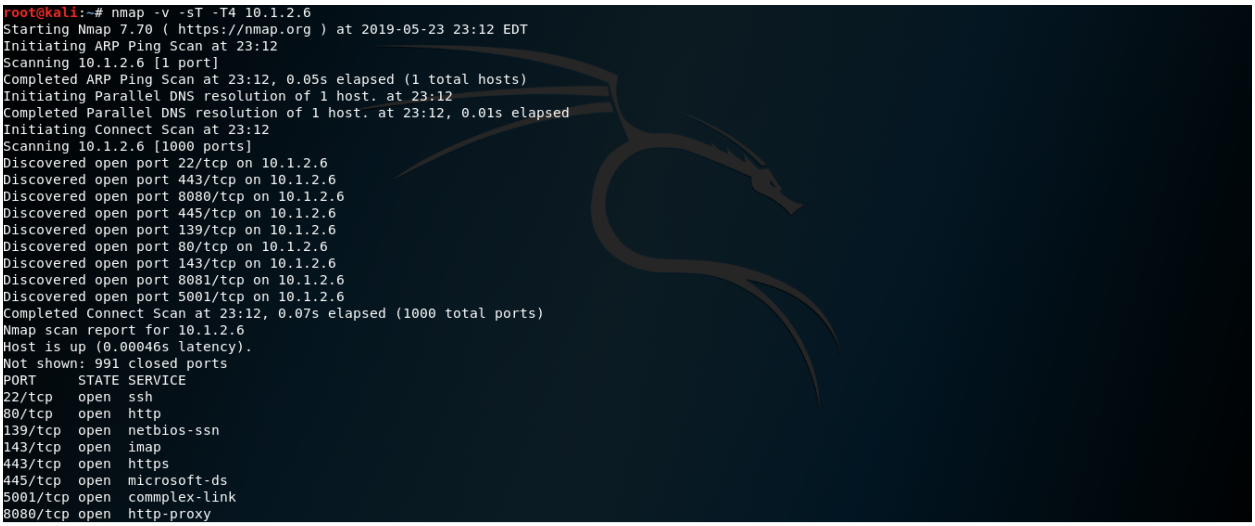
Nas informações detectadas dos headers HTTP, a máquina revela sua identidade: uma distro Ubuntu, além de apresentar outras informações como: versão de PHP, Python e Apache disponíveis na máquina

Questão 3



Listou as principais portas da máquina, serviços e estado das respectivas portas. A comunicação SYN ACK só foi possível nas portas 80 e 443, portas destinadas ao protocolo HTTP e HTTPS, disponíveis para qualquer pessoa se comunicar sem necessidade de autenticação.

Questão 4



Comando utilizado: *nmap -v -sT -T4 10.1.2.6* Listou as principais portas abertas na máquina e os serviços utilizadas por elas.

Questão 5

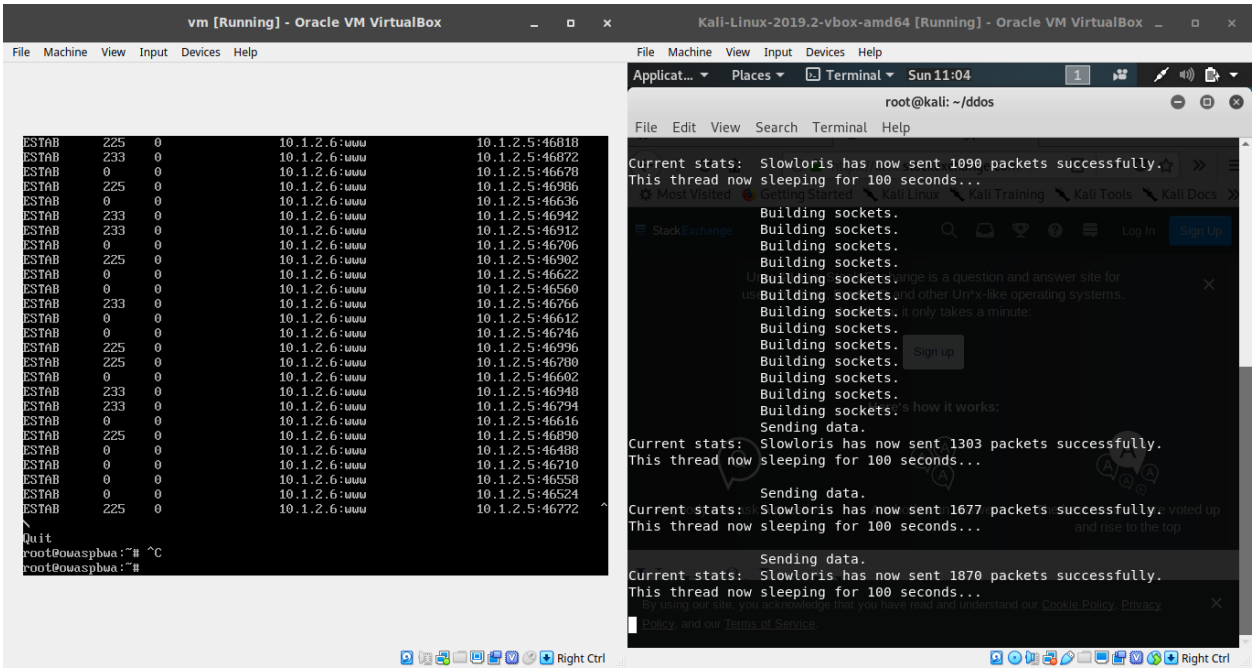
Item A

Na varradeura TCP SYN, o programa *nmap* envia um pacote SYN como se fosse iniciar uma conexão e quem recebeu o pacote responderá com a porta ou com um RST indicando que a porta está indisponível. Depois disso, o nmap envia imediatamente um pacote RST para encerrar a conexão. Enquanto que, a varradeura TCP irá estabelecer uma conexão real com cada porta disponível no servidor, caso ela esteja disponível.

Item B

A questão 3 usa varredura do tipo TCP SYN e a questão 1 usa conexão apenas TCP.

Item C

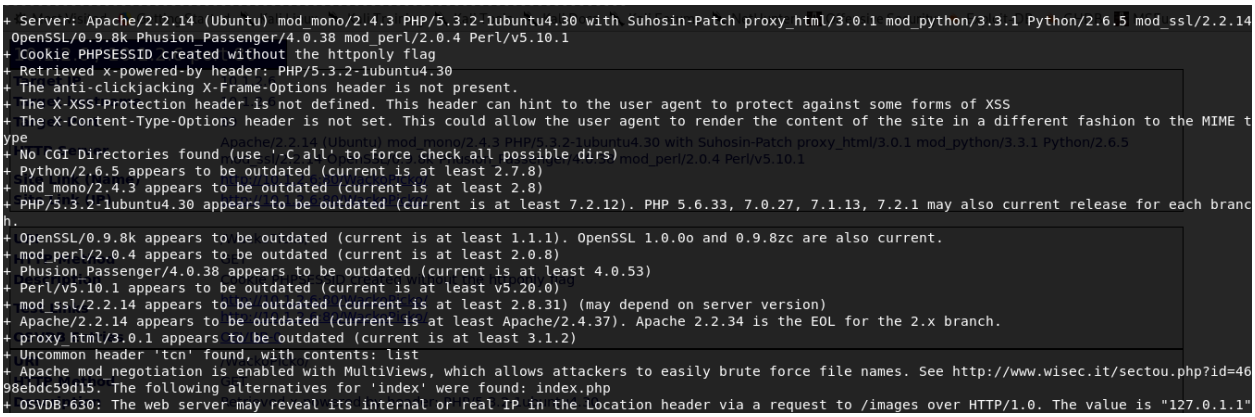


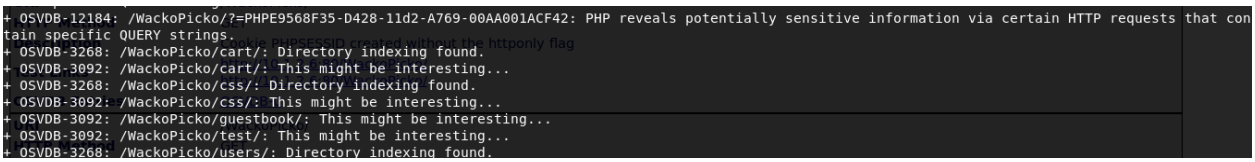
Como pode ser visto pela entrada CVE-2007-6750 (disponível em: http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2007-6750) é possível fazer um ataque de *Denial of Service* na máquina OWASPBWA uma vez que ela está utilizando a segunda versão do Apache. Esse ataque consiste em utilizar a ferramenta *slowloris*. Uma demonstração desse ataque pode ser visualizada na captura de tela acima.

Parte 2 - Nikto

Questão 6

Item A





Item B

O que mais chama atenção é que o Nikto é capaz de apontar algumas aplicações disponíveis na máquina e suas respectivas versões, indicando também se está atualizada ou não. Dessa forma, o atacante pode procurar vulnerabilidades nos pacotes desatualizados. Além disso, o programa indica potenciais arquivos com dados sensíveis.

URI	/WackoPicko/guestbook/?number=5&lng=%3Cscript%3Ealert(document.domain);%3C/script%3E
HTTP Method	GET
Description	/WackoPicko/guestbook/?number=5&lng=%3Cscript%3Ealert(document.domain);%3C/script%3E: MPM Guestbook 1.2 and previous are vulnreable to XSS attacks.
Test Links	http://10.1.2.6:80/WackoPicko/guestbook/?number=5&lng=%3Cscript%3Ealert(document.domain);%3C/script%3E http://10.1.2.6:80/WackoPicko/guestbook/?number=5&lng=%3Cscript%3Ealert(document.domain);%3C/script%3E
OSVDB Entries	OSVDB-2754

O Nikto detectou uma vulnerabilidade de injeção de código em uma das rotas do servidor como pode ser visto na rota testada acima.

Questão 7

A1: a injeção de código pode ocorrer em qualquer campo da aplicação no qual o usuário pode inserir informações. Nesse tipo de ataque, o atacante ao em vez de inserir dados, ele tenta colocar comandos (de SQL por exemplo). Nessa situação, caso a aplicação não esteja sanitizando o comando inserido pelo usuário (i.e. está tratando como dado e não como comando), o atacante terá liberdade para poder executar comandos direto na base de dados. Dessa forma, o atacante pode ter acesso à informações, as quais não deveria, ou pode até mesmo apagar o banco.

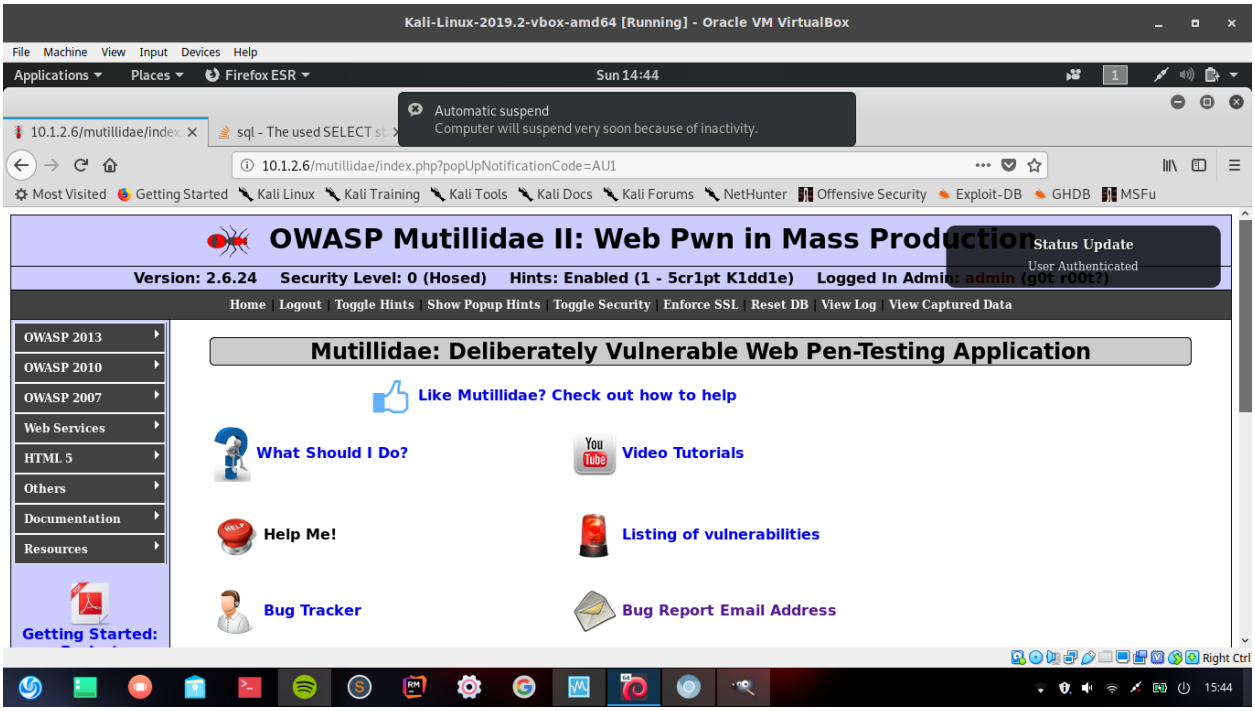
A2: a autenticação quebrada pode acontecer de várias formas. Uma delas é quando a aplicação não faz um bom gerenciamento de tokens.Propõe-se o seguinte centário: um usuário comum acessa um computador público e em vez de se desautenticar, ao sair, ele simplesmente fecha o navegador. Dessa forma, uma pessoa mal intencionada, ao acessar o mesmo computador, terá acesso total à conta do usuário. O atacante, então, pode falsificar sua real identidade, por exemplo. Outro possível ataque, é quando a aplicação não faz a proteção correta das credenciais. Isso é muito comum em aplicações que não bloqueiam temporariamente fontes que estejam tentando se logar e errando o par usuário e senha constantemente. Geralmente, nesse cenário, uma ferramenta automatizada com uma lista muito comum de senhas, estatisticamente, com um email previamente fornecido tenta exaustivamente logar com cada uma das senhas. Caso a aplicação não bloqueie essa fonte que está continuamente errando as credenciais, o atacante, eventualmente pode ter acesso à conta do usuário.

A3: a exposição de dados sensíveis é quando a aplicação não foi corretamente projetada para encriptar dados sensíveis. Uma das formas de isso acontecer, é quando a aplicação não encripta a senha dos usuários no banco, ou utiliza um algoritmo muito fraco para encriptar as senhas, como MD5 por exemplo, que pode ser facilmente quebrado. Uma outra possibilidade, é quando a aplicação encripta corretamente os dados no banco, mas faz a comunicação com o cliente sem criptografia (i.e. HTTPS). Isso permite que qualquer pessoa na rede consiga acompanhar e entender toda a comunicação entre cliente e servidor.

A7: Cross-site scripting é um ataque muito comum, principalmente em sites de fóruns. Esse ataque acontece em aplicações que não sanatizam HTML e salvam a informação no banco de dados sem nenhum tratamento. Dessa forma, um atacante insere código HTML combinado com JS no campo de comentários do fórum e envia para a aplicação. Caso o servidor não faça a sanatização desses dados, todas as pessoas que acessarem a página, verão a página alterada com os comandos do atacante e não a página original com um comentário do atacante. Dessa forma, através do JS inserido, o atacante pode ter total controle da aplicação.

Questão 8

Item A



Item B

O sistema autorizou a autenticação de um usuário, provavelmente, inexistente e sem senha. A vulnerabilidade explorada foi o SQL Injection. Nesse tipo de vulnerabilidade, o atacante pode utilizar comandos SQL em campos de informação que não foram corretamente sanitizados, o que acaba dando acesso ao banco de dados ao atacante

Item C

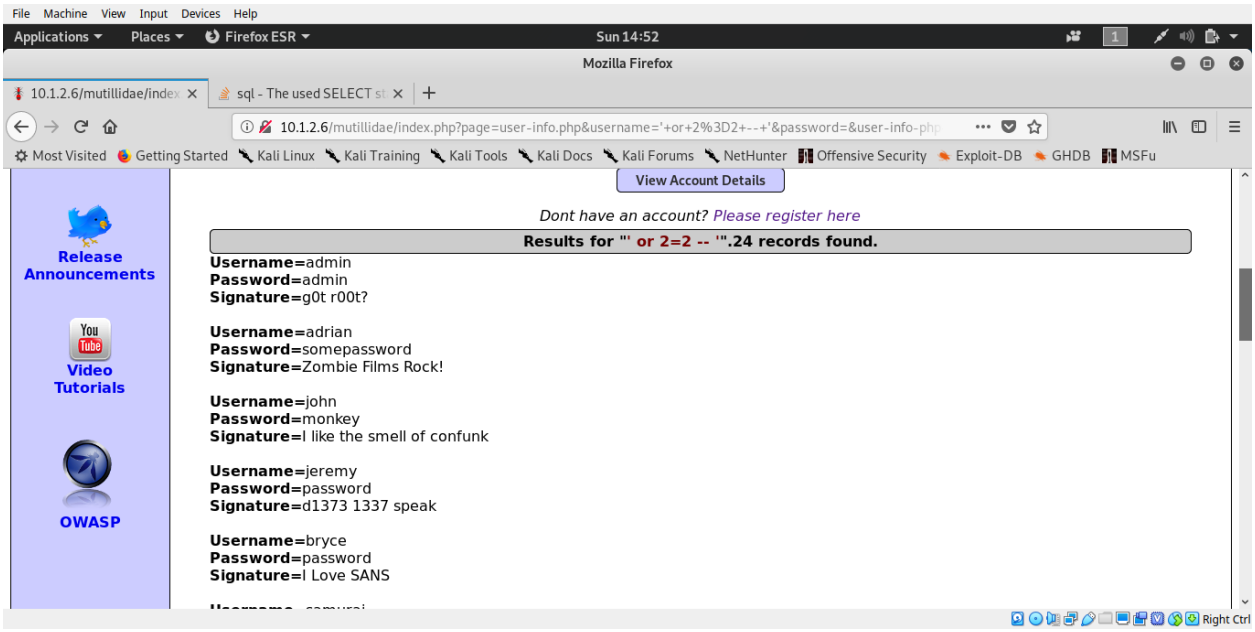
Os dados enviados ao servidor devem ser sanitizados através de um framework ou de funções que façam um escape nos comandos SQL para poderem ser interpretados como dados e não como comandos.

Questão 9

Item A

A vulnerabilidade explorada foi o SQL Injection. Nesse tipo de vulnerabilidade, o atacante pode utilizar comandos SQL em campos de informação que não foram corretamente sanitizados, o que acaba dando acesso ao banco de dados ao atacante

Item B



Item C

Os dados enviados ao servidor devem ser sanitizados através de um framework ou de funções que façam um escape nos comandos SQL para poderem ser interpretados como dados e não como comandos.

Questão 10

Item B

No relatório gerado é possível observer todas as vulnerabilidades detectadas pela ferramenta, grau de severidade, onde que a vulnerabilidade foi encontrada, de que forma e como o desenvolvedor pode corrigir cada uma delas. Foram encontradas um total de 13 fraquezas no sistema, sendo que dessas, 5 apresentam risco grave parao sistema, 4 são de risco médio e as demais de risco baixo. Dentre todas, destaca-se o XSS, SQL injection, remote OS command.

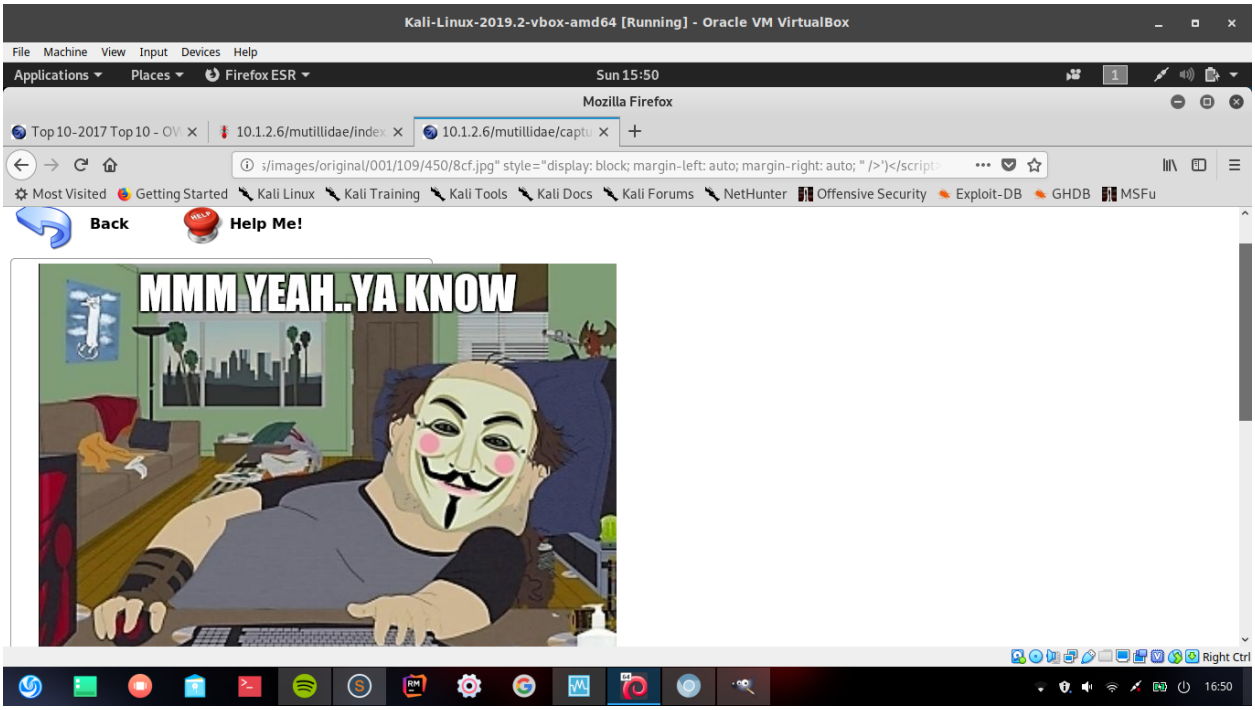
Item C

O relatório está anexado, dísponível no final do arquivo.

Questão 11

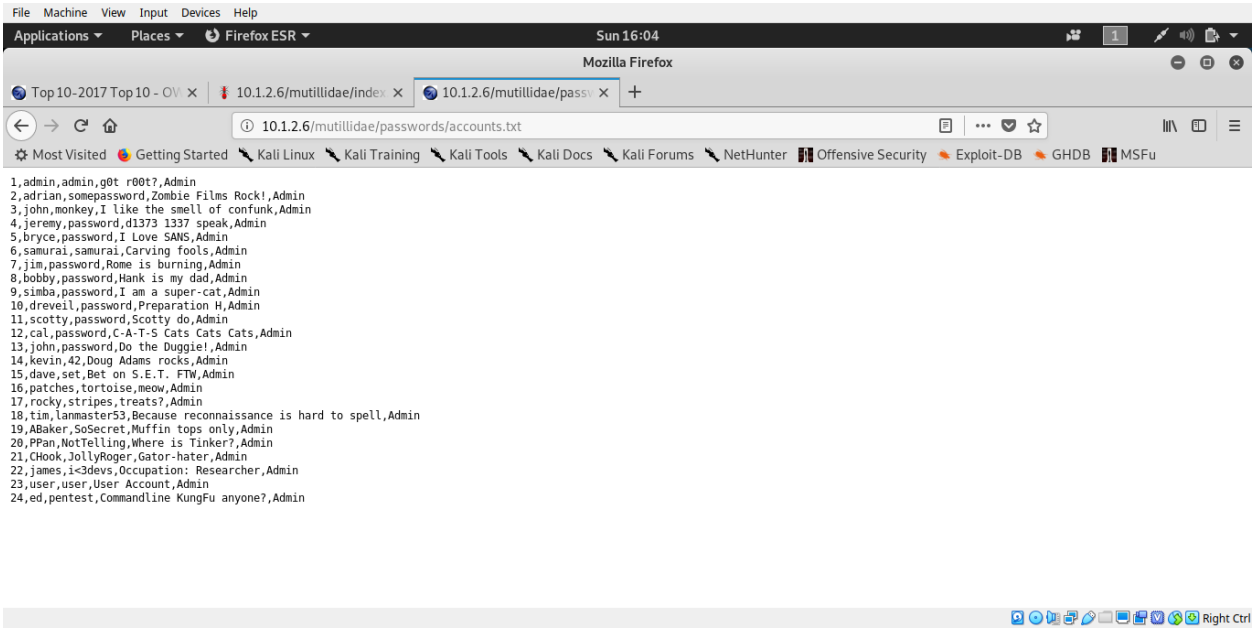
Os ataques escolhidos foram os Cross-site scripting (XSS) e Sensitive Data Exposure.

Para o XSS, foi utilizado a página */mutillidae/capture-data.php*. Como é possível, observar na documentação, essa página possui fraquezas XSS ao utilizar-se os vérbos HTTP GET e POST. Dessa forma, ao fazer uma requisição para essa mesma página utilizando o seguinte parâmetro na *query*: *<script>document.getElementById('idHintWrapperHeader').innerHTML=('')</script>*, será obtido o seguinte resultado:



Esse ataque decorre do fato de que a aplicação, no frontend, ao enviar a requisição, não está fazendo o parsing sanitizado dos dados. Dessa forma, o frontend fica suscetível a injeção de código, no caso de Javascript, o que permite alterar o conteúdo da página.

Para o seguinte ataque, ao visitar a página */robots.txt* ele exibe quais pastas não deveriam ser acessíveis para o usuário. Entretanto, ao tentar visitar algum desses diretórios, não há nenhum tipo de bloqueio. Dessa forma, é possível acessar a pasta password e visualizar senha e email dos usuários:



Essa falha permite que o atacante tenha acesso a dados sensíveis, quando não deveria. A vulnerabilidade ocorreu, primeiramente, pelo fato do arquivo robots.txt indicar diretórios onde ficam guardadas informações sigilosas. Além disso, apesar de os diretórios estarem listados como não acessíveis para o usuário, eles não estavam bloqueados de nenhuma forma, o que permite ao atacante visualizar informações secretas.

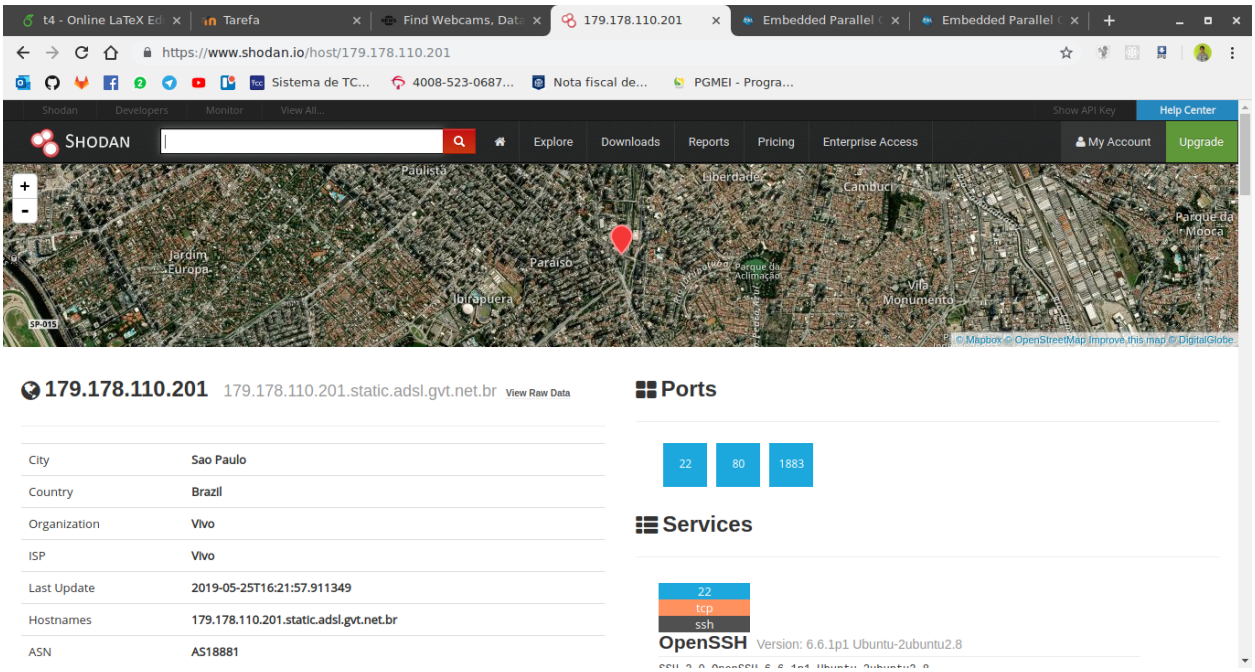
Questão 12

Item A

O Shodan é uma ferramenta online que permite localizar dispositivos IoT no geral como: semáforos, câmeras de segurança e até babás eletrônicas. Ele pode ser usado como ferramenta para verificar falhas em um determinado tipo de sistema, mas pode ser utilizado como search engine para encontrar dispositivos baseado em IP, tipo de equipamento e DNS.

Item B

O dispositivo buscado foi um ESP8266. Esse dispositivo é um placa de rede WiFi que permite que hardwares como o Arduino se conectem na rede sem fio e possam se comunicar através do protocol HTTP.



Questão 13

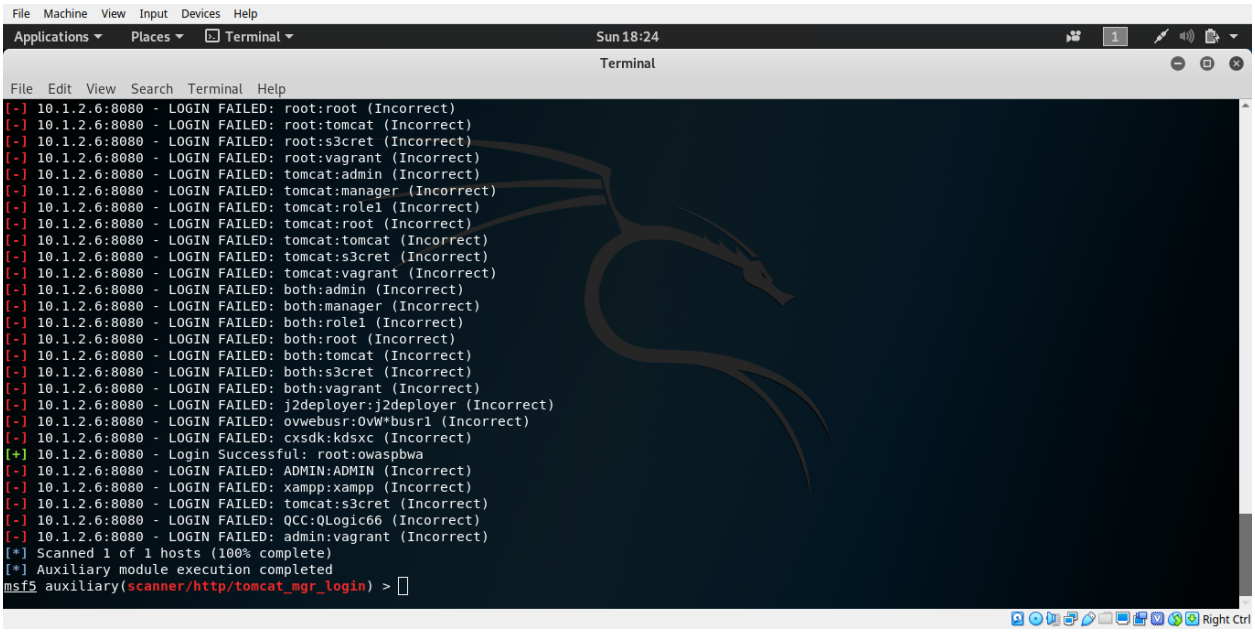
Item A

As imagens de uma câmera pública em tempo real.

Item B

Pode entender mais sobre o cotidiano do local, bem como as pessoas que o frequentam rotineiramente.

Questão 14



Item A

O ataque do dicionário é um ataque baseado em força bruta que utiliza uma ferramenta automatizada para tentar descobrir a senha ou chave de um sistema. Esse ataque pode tentar milhares ou até milhões de possibilidades, como as palavras de um dicionário.

Item B

Um par de senha e usuário que podem ser utilizados para se autenticar no servidor.

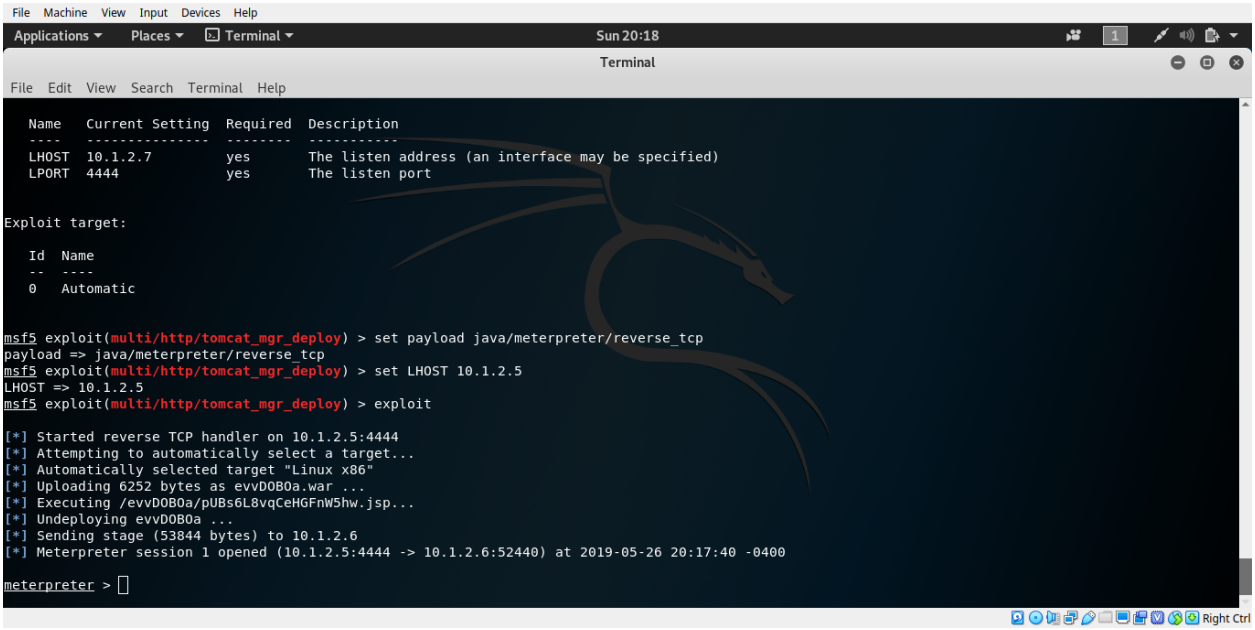
Item C

A vunerabilidade explorada foi a configuração incorreta de segurança já que o sistema veio com um usuário e senha padrão e esses não foram alterados.

Item D

Com as credenciais válidas, o atacante pode se logar no sistema e terá controle total sobre a máquina.

Questão 15



Item A

A vulnerabilidade é a mesma da questão anterior: configuração incorreta de segurança. Isso se deve ao fato de que na questão anterior foi possível descobrir usuário e senha padrão da aplicação. Dessa forma, apenas foi necessário configurar esses valores na ferramenta de ataque para que fosse possível conectar na máquina alvo.

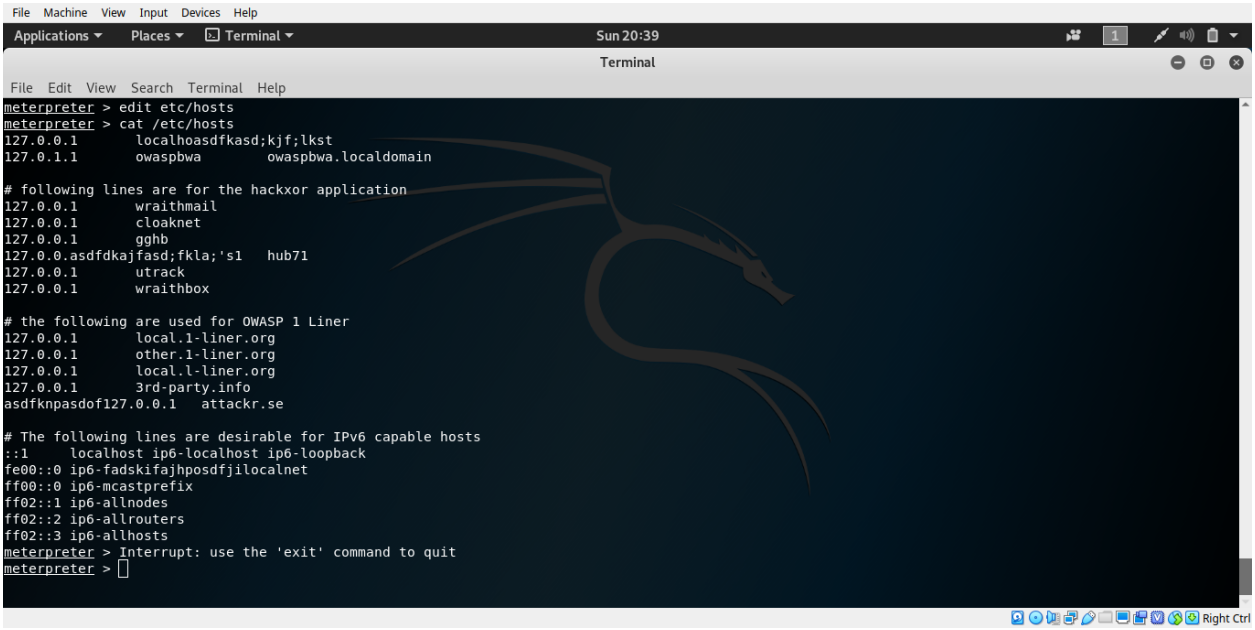
Item B

Ele conecta na máquina através de uma conexão TCP na porta 4444 utilizando login e senha fornecidos pelo usuário. Nesse caso, os valores utilizados, foram os descobertos pela aplicação na questão anterior.

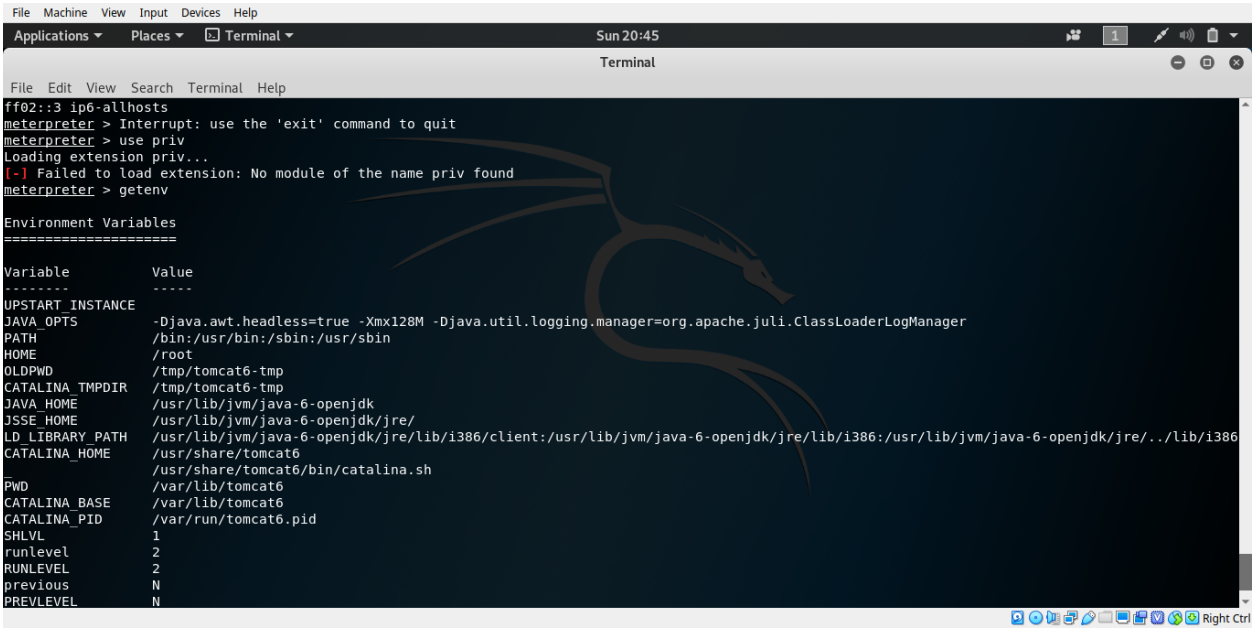
Item C

É uma ferramnta de conexão, similar ao SSH, que permite o cliente executar comandos na máquina alvo. Ele é baseado em injeção de staggers *in-memory* DLL.

Item D



O arquivo /etc/hosts foi desconfigurado com o comando *edit* como pode ser observado.



As variáveis de ambiente foram exibidas atraűes do comando *getenv*. Apesar de não haver nenhuma variável que configure a senha do banco de dados, isso é muito comum em um sistema em produção. Caso esse fosse o caso, o atacante teria acesso direto a senha (i.e. sem estar encriptada).

ZAP Scanning Report

Summary of Alerts

Risk Level	Number of Alerts
High	5
Medium	4
Low	4
Informational	0

Alert Detail

High (Medium)	Cross Site Scripting (Reflected)
Description	Cross-site Scripting (XSS) is an attack technique that involves echoing attacker-supplied code into a user's browser instance. A browser instance can be a standard web browser client, or a browser object embedded in a software product such as the browser within WinAmp, an RSS reader, or an email client. The code itself is usually written in HTML/JavaScript, but may also extend to VBScript, ActiveX, Java, Flash, or any other browser-supported technology. When an attacker gets a user's browser to execute his/her code, the code will run within the security context (or zone) of the hosting web site. With this level of privilege, the code has the ability to read, modify and transmit any sensitive data accessible by the browser. A Cross-site Scripted user could have his/her account hijacked (cookie theft), their browser redirected to another location, or possibly shown fraudulent content delivered by the web site they are visiting. Cross-site Scripting attacks essentially compromise the trust relationship between a user and the web site. Applications utilizing browser object instances which load content from the file system may execute code under the local machine zone allowing for system compromise. There are three types of Cross-site Scripting attacks: non-persistent, persistent and DOM-based. Non-persistent attacks and DOM-based attacks require a user to either visit a specially crafted link laced with malicious code, or visit a malicious web page containing a web form, which when posted to the vulnerable site, will mount the attack. Using a malicious form will oftentimes take place when the vulnerable resource only accepts HTTP POST requests. In such a case, the form can be submitted automatically, without the victim's knowledge (e.g. by using JavaScript). Upon clicking on the malicious link or submitting the malicious form, the XSS payload will get echoed back and will get interpreted by the user's browser and execute. Another technique to send almost arbitrary requests (GET and POST) is by using an embedded client, such as Adobe Flash. Persistent attacks occur when the malicious code is submitted to a web site where it's stored for a period of time. Examples of an attacker's favorite targets often include message board posts, web mail messages, and web chat software. The unsuspecting user is not required to interact with any additional site/link (e.g. an attacker site or a malicious link sent via email), just simply view the web page containing the code.
URL	http://10.1.2.6/WackoPicko/guestbook.php
Method	POST
Parameter	comment
Attack	<script>alert(1);</script>
Evidence	<script>alert(1);</script>
URL	http://10.1.2.6/WackoPicko/pictures/search.php?query=%22%3E%3Cscript%3Ealert%281%29%3B%3C%2Fscript%3E
Method	GET
Parameter	query
Attack	"><script>alert(1);</script>
Evidence	"><script>alert(1);</script>
Instances	2
Solution	Phase: Architecture and Design Use a vetted library or framework that does not allow this weakness to occur or provides constructs that make this weakness easier to avoid. Examples of libraries and frameworks that make it easier to generate properly encoded output include Microsoft's Anti-XSS library, the OWASP ESAPI Encoding module, and Apache Wicket. Phases: Implementation; Architecture and Design

	Understand the context in which your data will be used and the encoding that will be expected. This is especially important when transmitting data between different components, or when generating outputs that can contain multiple encodings at the same time, such as web pages or multi-part mail messages. Study all expected communication protocols and data representations to determine the required encoding strategies. For any data that will be output to another web page, especially any data that was received from external inputs, use the appropriate encoding on all non-alphanumeric characters. Consult the XSS Prevention Cheat Sheet for more details on the types of encoding and escaping that are needed. Phase: Architecture and Design For any security checks that are performed on the client side, ensure that these checks are duplicated on the server side, in order to avoid CWE-602. Attackers can bypass the client-side checks by modifying values after the checks have been performed, or by changing the client to remove the client-side checks entirely. Then, these modified values would be submitted to the server. If available, use structured mechanisms that automatically enforce the separation between data and code. These mechanisms may be able to provide the relevant quoting, encoding, and validation automatically, instead of relying on the developer to provide this capability at every point where output is generated. Phase: Implementation For every web page that is generated, use and specify a character encoding such as ISO-8859-1 or UTF-8. When an encoding is not specified, the web browser may choose a different encoding by guessing which encoding is actually being used by the web page. This can cause the web browser to treat certain sequences as special, opening up the client to subtle XSS attacks. See CWE-116 for more mitigations related to encoding/escaping. To help mitigate XSS attacks against the user's session cookie, set the session cookie to be HttpOnly. In browsers that support the HttpOnly feature (such as more recent versions of Internet Explorer and Firefox), this attribute can prevent the user's session cookie from being accessible to malicious client-side scripts that use document.cookie. This is not a complete solution, since HttpOnly is not supported by all browsers. More importantly, XMLHTTPRequest and other powerful browser technologies provide read access to HTTP headers, including the Set-Cookie header in which the HttpOnly flag is set. Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a whitelist of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does. Do not rely exclusively on looking for malicious or malformed inputs (i.e., do not rely on a blacklist). However, blacklists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright. When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if you are expecting colors such as "red" or "blue." Ensure that you perform input validation at well-defined interfaces within the application. This will help protect the application even if a component is reused or moved elsewhere.
Reference	http://projects.webappsec.org/Cross-Site-Scripting http://cwe.mitre.org/data/definitions/79.html
CWE Id	79
WASC Id	8
Source ID	1

High (Medium)	**SQL Injection**
Description	SQL injection may be possible.
URL	http://10.1.2.6/WackoPicko/users/login.php
Method	POST
Parameter	username
Attack	ZAP' AND '1'='1' --
Instances	1
Solution	Do not trust client side input, even if there is client side validation in place. In general, type check all data on the server side. If the application uses JDBC, use PreparedStatement or CallableStatement, with parameters passed by '?' If the application uses ASP, use ADO Command Objects with strong type checking and parameterized queries. If database Stored Procedures can be used, use them. Do *not* concatenate strings into queries in the stored procedure, or use 'exec', 'exec immediate', or equivalent functionality! Do not create dynamic SQL queries using simple string concatenation. Escape all data received from the client.

	Apply a 'whitelist' of allowed characters, or a 'blacklist' of disallowed characters in user input. Apply the principle of least privilege by using the least privileged database user possible. In particular, avoid using the 'sa' or 'db-owner' database users. This does not eliminate SQL injection, but minimizes its impact. Grant the minimum database access that is necessary for the application.
Other information	The page results were successfully manipulated using the boolean conditions [ZAP' AND '1'='1' --] and [ZAP' AND '1'='2' --] The parameter value being modified was NOT stripped from the HTML output for the purposes of the comparison Data was returned for the original parameter. The vulnerability was detected by successfully restricting the data originally returned, by manipulating the parameter
Reference	https://www.owasp.org/index.php/Top_10_2010-A1 https://www.owasp.org/index.php/SQL_Injection_Prevention_Cheat_Sheet
CWE Id	89
WASC Id	19
Source ID	1

High (Medium)	**Cross Site Scripting (Persistent)**
Description	Cross-site Scripting (XSS) is an attack technique that involves echoing attacker-supplied code into a user's browser instance. A browser instance can be a standard web browser client, or a browser object embedded in a software product such as the browser within WinAmp, an RSS reader, or an email client. The code itself is usually written in HTML/JavaScript, but may also extend to VBScript, ActiveX, Java, Flash, or any other browser-supported technology. When an attacker gets a user's browser to execute his/her code, the code will run within the security context (or zone) of the hosting web site. With this level of privilege, the code has the ability to read, modify and transmit any sensitive data accessible by the browser. A Cross-site Scripted user could have his/her account hijacked (cookie theft), their browser redirected to another location, or possibly shown fraudulent content delivered by the web site they are visiting. Cross-site Scripting attacks essentially compromise the trust relationship between a user and the web site. Applications utilizing browser object instances which load content from the file system may execute code under the local machine zone allowing for system compromise. There are three types of Cross-site Scripting attacks: non-persistent, persistent and DOM-based. Non-persistent attacks and DOM-based attacks require a user to either visit a specially crafted link laced with malicious code, or visit a malicious web page containing a web form, which when posted to the vulnerable site, will mount the attack. Using a malicious form will oftentimes take place when the vulnerable resource only accepts HTTP POST requests. In such a case, the form can be submitted automatically, without the victim's knowledge (e.g. by using JavaScript). Upon clicking on the malicious link or submitting the malicious form, the XSS payload will get echoed back and will get interpreted by the user's browser and execute. Another technique to send almost arbitrary requests (GET and POST) is by using an embedded client, such as Adobe Flash. Persistent attacks occur when the malicious code is submitted to a web site where it's stored for a period of time. Examples of an attacker's favorite targets often include message board posts, web mail messages, and web chat software. The unsuspecting user is not required to interact with any additional site/link (e.g. an attacker site or a malicious link sent via email), just simply view the web page containing the code.
URL	http://10.1.2.6/WackoPicko/guestbook.php
Method	POST
Parameter	comment
Attack	<script>alert(1);</script>
Instances	1
Solution	Phase: Architecture and Design Use a vetted library or framework that does not allow this weakness to occur or provides constructs that make this weakness easier to avoid. Examples of libraries and frameworks that make it easier to generate properly encoded output include Microsoft's Anti-XSS library, the OWASP ESAPI Encoding module, and Apache Wicket. Phases: Implementation; Architecture and Design Understand the context in which your data will be used and the encoding that will be expected. This is especially important when transmitting data between different components, or when generating outputs that can contain multiple encodings at the same time, such as web pages or multi-part mail messages. Study all expected communication protocols and data representations to determine the required encoding strategies. For any data that will be output to another web page, especially any data that was received from external inputs, use the appropriate encoding on all non-alphanumeric characters. Consult the XSS Prevention Cheat Sheet for more details on the types of encoding and escaping that are needed. Phase: Architecture and Design

	For any security checks that are performed on the client side, ensure that these checks are duplicated on the server side, in order to avoid CWE-602. Attackers can bypass the client-side checks by modifying values after the checks have been performed, or by changing the client to remove the client-side checks entirely. Then, these modified values would be submitted to the server. If available, use structured mechanisms that automatically enforce the separation between data and code. These mechanisms may be able to provide the relevant quoting, encoding, and validation automatically, instead of relying on the developer to provide this capability at every point where output is generated. Phase: Implementation For every web page that is generated, use and specify a character encoding such as ISO-8859-1 or UTF-8. When an encoding is not specified, the web browser may choose a different encoding by guessing which encoding is actually being used by the web page. This can cause the web browser to treat certain sequences as special, opening up the client to subtle XSS attacks. See CWE-116 for more mitigations related to encoding/escaping. To help mitigate XSS attacks against the user's session cookie, set the session cookie to be HttpOnly. In browsers that support the HttpOnly feature (such as more recent versions of Internet Explorer and Firefox), this attribute can prevent the user's session cookie from being accessible to malicious client-side scripts that use document.cookie. This is not a complete solution, since HttpOnly is not supported by all browsers. More importantly, XMLHTTPRequest and other powerful browser technologies provide read access to HTTP headers, including the Set-Cookie header in which the HttpOnly flag is set. Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a whitelist of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does. Do not rely exclusively on looking for malicious or malformed inputs (i.e., do not rely on a blacklist). However, blacklists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright. When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if you are expecting colors such as "red" or "blue." Ensure that you perform input validation at well-defined interfaces within the application. This will help protect the application even if a component is reused or moved elsewhere.
Other information	Source URL: http://10.1.2.6/WackoPicko/guestbook.php
Reference	http://projects.webappsec.org/Cross-Site-Scripting http://cwe.mitre.org/data/definitions/79.html
CWE Id	79
WASC Id	8
Source ID	1

High (Medium)	**Remote OS Command Injection**
Description	Attack technique used for unauthorized execution of operating system commands. This attack is possible when an application accepts untrusted input to build operating system commands in an insecure manner involving improper data sanitization, and/or improper calling of external programs.
URL	http://10.1.2.6/WackoPicko/passcheck.php
Method	POST
Parameter	password
Attack	ZAP&sleep 15&
Instances	1
Solution	If at all possible, use library calls rather than external processes to recreate the desired functionality. Run your code in a "jail" or similar sandbox environment that enforces strict boundaries between the process and the operating system. This may effectively restrict which files can be accessed in a particular directory or which commands can be executed by your software. OS-level examples include the Unix chroot jail, AppArmor, and SELinux. In general, managed code may provide some protection. For example, java.io.FilePermission in the Java SecurityManager allows you to specify restrictions on file operations. This may not be a feasible solution, and it only limits the impact to the operating system; the rest of your application may still be subject to compromise. For any data that will be used to generate a command to be executed, keep as much of that data out of external control as possible. For example, in web applications, this may require storing the command locally in the session's state instead of sending it out to the client in a hidden form field. Use a vetted library or framework that does not allow this weakness to occur or provides constructs that make this weakness easier to avoid. For example, consider using the ESAPI Encoding control or a similar tool, library, or framework. These will help the programmer encode outputs in a manner less prone to error. If you need to use dynamically-generated query strings or commands in spite of the risk, properly quote arguments and escape any special characters within those arguments. The most conservative approach is to escape or filter all characters that do not pass an extremely strict whitelist (such as everything that is not alphanumeric or white space). If some special characters are still needed, such as white space, wrap each argument in quotes after the escaping/filtering step. Be careful of argument injection.

	If the program to be executed allows arguments to be specified within an input file or from standard input, then consider using that mode to pass arguments instead of the command line. If available, use structured mechanisms that automatically enforce the separation between data and code. These mechanisms may be able to provide the relevant quoting, encoding, and validation automatically, instead of relying on the developer to provide this capability at every point where output is generated. Some languages offer multiple functions that can be used to invoke commands. Where possible, identify any function that invokes a command shell using a single string, and replace it with a function that requires individual arguments. These functions typically perform appropriate quoting and filtering of arguments. For example, in C, the system() function accepts a string that contains the entire command to be executed, whereas execl(), execve(), and others require an array of strings, one for each argument. In Windows, CreateProcess() only accepts one command at a time. In Perl, if system() is provided with an array of arguments, then it will quote each of the arguments. Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a whitelist of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does. Do not rely exclusively on looking for malicious or malformed inputs (i.e., do not rely on a blacklist). However, blacklists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright. When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if you are expecting colors such as "red" or "blue." When constructing OS command strings, use stringent whitelists that limit the character set based on the expected value of the parameter in the request. This will indirectly limit the scope of an attack, but this technique is less important than proper output encoding and escaping. Note that proper output encoding, escaping, and quoting is the most effective solution for preventing OS command injection, although input validation may provide some defense-in-depth. This is because it effectively limits what will appear in output. Input validation will not always prevent OS command injection, especially if you are required to support free-form text fields that could contain arbitrary characters. For example, when invoking a mail program, you might need to allow the subject field to contain otherwise-dangerous inputs like ";" and ">" characters, which would need to be escaped or otherwise handled. In this case, stripping the character might reduce the risk of OS command injection, but it would produce incorrect behavior because the subject field would not be recorded as the user intended. This might seem to be a minor inconvenience, but it could be more important when the program relies on well-structured subject lines in order to pass messages to other components. Even if you make a mistake in your validation (such as forgetting one out of 100 input fields), appropriate encoding is still likely to protect you from injection-based attacks. As long as it is not done in isolation, input validation is still a useful technique, since it may significantly reduce your attack surface, allow you to detect some attacks, and provide other security benefits that proper encoding does not address.
Reference	http://cwe.mitre.org/data/definitions/78.html https://www.owasp.org/index.php/Command_Injection
CWE Id	78
WASC Id	31
Source ID	1

High (Low)	**Cross Site Scripting (Reflected)**
Description	Cross-site Scripting (XSS) is an attack technique that involves echoing attacker-supplied code into a user's browser instance. A browser instance can be a standard web browser client, or a browser object embedded in a software product such as the browser within WinAmp, an RSS reader, or an email client. The code itself is usually written in HTML/JavaScript, but may also extend to VBScript, ActiveX, Java, Flash, or any other browser-supported technology. When an attacker gets a user's browser to execute his/her code, the code will run within the security context (or zone) of the hosting web site. With this level of privilege, the code has the ability to read, modify and transmit any sensitive data accessible by the browser. A Cross-site Scripted user could have his/her account hijacked (cookie theft), their browser redirected to another location, or possibly shown fraudulent content delivered by the web site they are visiting. Cross-site Scripting attacks essentially compromise the trust relationship between a user and the web site. Applications utilizing browser object instances which load content from the file system may execute code under the local machine zone allowing for system compromise. There are three types of Cross-site Scripting attacks: non-persistent, persistent and DOM-based. Non-persistent attacks and DOM-based attacks require a user to either visit a specially crafted link laced with malicious code, or visit a malicious web page containing a web form, which when posted to the vulnerable site, will mount the attack. Using a malicious form will oftentimes take place when the vulnerable resource only accepts HTTP POST requests. In such a case, the form can be submitted automatically, without the victim's knowledge (e.g. by using JavaScript). Upon clicking on the malicious link or submitting the malicious form, the XSS payload will get echoed back and will get interpreted by the user's browser and execute. Another technique to send almost arbitrary requests (GET and POST) is by using an embedded client, such as Adobe Flash. Persistent attacks occur when the malicious code is submitted to a web site where it's stored for a period of time. Examples of an attacker's favorite targets often include message board posts, web mail messages, and web chat software. The unsuspecting user is not required to interact with any additional site/link (e.g. an attacker site or a malicious link sent via email), just simply view the web page containing the code.
URL	http://10.1.2.6/WackoPicko/users/login.php
Method	POST
Parameter	username
Attack	'"<script>alert(1);</script>
Evidence	'"<script>alert(1);</script>
Instances	1

Solution	Phase: Architecture and Design Use a vetted library or framework that does not allow this weakness to occur or provides constructs that make this weakness easier to avoid. Examples of libraries and frameworks that make it easier to generate properly encoded output include Microsoft's Anti-XSS library, the OWASP ESAPI Encoding module, and Apache Wicket. Phases: Implementation; Architecture and Design Understand the context in which your data will be used and the encoding that will be expected. This is especially important when transmitting data between different components, or when generating outputs that can contain multiple encodings at the same time, such as web pages or multi-part mail messages. Study all expected communication protocols and data representations to determine the required encoding strategies. For any data that will be output to another web page, especially any data that was received from external inputs, use the appropriate encoding on all non-alphanumeric characters. Consult the XSS Prevention Cheat Sheet for more details on the types of encoding and escaping that are needed. Phase: Architecture and Design For any security checks that are performed on the client side, ensure that these checks are duplicated on the server side, in order to avoid CWE-602. Attackers can bypass the client-side checks by modifying values after the checks have been performed, or by changing the client to remove the client-side checks entirely. Then, these modified values would be submitted to the server. If available, use structured mechanisms that automatically enforce the separation between data and code. These mechanisms may be able to provide the relevant quoting, encoding, and validation automatically, instead of relying on the developer to provide this capability at every point where output is generated. Phase: Implementation For every web page that is generated, use and specify a character encoding such as ISO-8859-1 or UTF-8. When an encoding is not specified, the web browser may choose a different encoding by guessing which encoding is actually being used by the web page. This can cause the web browser to treat certain sequences as special, opening up the client to subtle XSS attacks. See CWE-116 for more mitigations related to encoding/escaping. To help mitigate XSS attacks against the user's session cookie, set the session cookie to be HttpOnly. In browsers that support the HttpOnly feature (such as more recent versions of Internet Explorer and Firefox), this attribute can prevent the user's session cookie from being accessible to malicious client-side scripts that use document.cookie. This is not a complete solution, since HttpOnly is not supported by all browsers. More importantly, XMLHTTPRequest and other powerful browser technologies provide read access to HTTP headers, including the Set-Cookie header in which the HttpOnly flag is set. Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a whitelist of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does. Do not rely exclusively on looking for malicious or malformed inputs (i.e., do not rely on a blacklist). However, blacklists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright. When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if you are expecting colors such as "red" or "blue." Ensure that you perform input validation at well-defined interfaces within the application. This will help protect the application even if a component is reused or moved elsewhere.
Reference	http://projects.webappsec.org/Cross-Site-Scripting http://cwe.mitre.org/data/definitions/79.html
CWE Id	79
WASC Id	8
Source ID	1

Medium (Medium)	**X-Frame-Options Header Not Set**
Description	X-Frame-Options header is not included in the HTTP response to protect against 'ClickJacking' attacks.
URL	http://10.1.2.6/WackoPicko/calendar.php?date=1559232048
Method	GET
Parameter	X-Frame-Options
URL	http://10.1.2.6/WackoPicko/calendar.php?date=1559059248
Method	GET
Parameter	X-Frame-Options
URL	http://10.1.2.6/WackoPicko/passcheck.php
Method	GET
Parameter	X-Frame-Options

URL	http://10.1.2.6/WackoPicko/cart/review.php
Method	GET
Parameter	X-Frame-Options
URL	http://10.1.2.6/WackoPicko/users/register.php
Method	GET
Parameter	X-Frame-Options
URL	http://10.1.2.6/WackoPicko/calendar.php?date=1558972848
Method	GET
Parameter	X-Frame-Options
URL	http://10.1.2.6/WackoPicko/users/login.php
Method	GET
Parameter	X-Frame-Options
URL	http://10.1.2.6/WackoPicko/pictures/recent.php
Method	GET
Parameter	X-Frame-Options
URL	http://10.1.2.6/WackoPicko/passcheck.php
Method	POST
Parameter	X-Frame-Options
URL	http://10.1.2.6/WackoPicko/calendar.php
Method	GET
Parameter	X-Frame-Options
URL	http://10.1.2.6/WackoPicko/tos.php
Method	GET
Parameter	X-Frame-Options
URL	http://10.1.2.6/WackoPicko/pictures/search.php?query=ZAP
Method	GET
Parameter	X-Frame-Options
URL	http://10.1.2.6/WackoPicko/guestbook.php
Method	GET
Parameter	X-Frame-Options
URL	http://10.1.2.6/WackoPicko/users/sample.php?userid=1
Method	GET
Parameter	X-Frame-Options
URL	http://10.1.2.6/WackoPicko/guestbook.php
Method	POST
Parameter	X-Frame-Options
URL	http://10.1.2.6/WackoPicko/
Method	GET
Parameter	X-Frame-Options
URL	http://10.1.2.6/WackoPicko/admin/index.php?page=login
Method	GET

Parameter	X-Frame-Options
URL	http://10.1.2.6/WackoPicko/calendar.php?date=1559145648
Method	GET
Parameter	X-Frame-Options
URL	http://10.1.2.6/WackoPicko/admin/index.php?page=login
Method	POST
Parameter	X-Frame-Options
Instances	19
Solution	Most modern Web browsers support the X-Frame-Options HTTP header. Ensure it's set on all web pages returned by your site (if you expect the page to be framed only by pages on your server (e.g. it's part of a FRAMESET) then you'll want to use SAMEORIGIN, otherwise if you never expect the page to be framed, you should use DENY. ALLOW-FROM allows specific websites to frame the web page in supported web browsers).
Reference	http://blogs.msdn.com/b/ieinternals/archive/2010/03/30/combating-clickjacking-with-x-frame-options.aspx
CWE Id	16
WASC Id	15
Source ID	3

Medium (Medium)	**Directory Browsing**
Description	It is possible to view the directory listing. Directory listing may reveal hidden scripts, include files , backup source files etc which can be accessed to read sensitive information.
URL	http://10.1.2.6/WackoPicko/cart/
Method	GET
Attack	Parent Directory
URL	http://10.1.2.6/WackoPicko/upload/doggie/
Method	GET
Attack	Parent Directory
URL	http://10.1.2.6/WackoPicko/upload/
Method	GET
Attack	Parent Directory
URL	http://10.1.2.6/WackoPicko/upload/waterfall/
Method	GET
Attack	Parent Directory
URL	http://10.1.2.6/WackoPicko/upload/house/
Method	GET
Attack	Parent Directory
URL	http://10.1.2.6/WackoPicko/users/
Method	GET
Attack	Parent Directory
URL	http://10.1.2.6/WackoPicko/pictures/
Method	GET
Attack	Parent Directory
URL	http://10.1.2.6/WackoPicko/css/blueprint/
Method	GET
Attack	Parent Directory

URL	http://10.1.2.6/WackoPicko/upload/toga/
Method	GET
Attack	Parent Directory
URL	http://10.1.2.6/WackoPicko/css/
Method	GET
Attack	Parent Directory
URL	http://10.1.2.6/WackoPicko/upload/flowers/
Method	GET
Attack	Parent Directory
Instances	11
Solution	Disable directory browsing. If this is required, make sure the listed files does not induce risks.
Reference	http://httpd.apache.org/docs/mod/core.html#options http://alamo.satlug.org/pipermail/satlug/2002-February/000053.html
CWE Id	548
WASC Id	48
Source ID	1

Medium (Medium)	Buffer Overflow
Description	Buffer overflow errors are characterized by the overwriting of memory spaces of the background web process, which should have never been modified intentionally or unintentionally. Overwriting values of the IP (Instruction Pointer), BP (Base Point
URL	http://10.1.2.6/WackoPicko/admin/index.php?page=login
Method	POST
Parameter	page
Attack	POST http://10.1.2.6/WackoPicko/admin/index.php? page=fZfRkxWhvPmXKLMbkjtntbSfawZDDBiqYFEEEwXpYpJgrUMIwxGiWicBEwxWprBPgDrGdeQvSblcvdORXVuHkKpSDuAigBKvYAsdyKMlkPMnLFqQgUTbXigyFGpqCphvTZeqjrMjxWbVHoLkkJZZNvVetvlPTBchoBlWFnRWwkteoOJiutlhMWM\ HTTP/1.1 User-Agent: Mozilla/5.0 (Windows NT 6.3; WOW64; rv:39.0) Gecko/20100101 Firefox/39.0 Pragma: no-cache Cache-Control: no-cache Content-Type: application/x-www-form-urlencoded Content-Length: 26 Referer: http://10.1.2.6/Wack
URL	http://10.1.2.6/WackoPicko/admin
Method	GET
Parameter	query
Attack	GET http://10.1.2.6/WackoPicko/admin? query=ncWPGRDyJDugZrHOYpKtQpbncxquqGEanjFickDpquUXrfdfQCDfwCeqtuwemhPXCVjLaqGPDOYEgEhBRgOujvuTLANauORcjZwUijMQakZpTwXNPiQfGfaIGxpUxrNuBoGxLdRHlKZcdusytCxheZIPUQBWrCbNhdtFcNBHQnDFAUasleyXF(Mozilla/5.0 (Windows NT 6.3; WOW64; rv:39.0) Gecko/20100101 Firefox/39.0 Pragma: no-cache Cache-Control: no-cache Referer: http://10.1.2.6/WackoPicko/ Cookie: PHPSESSID=knn2ggc3jhebm30jfnm8m434f4 Content-Length: 0 Host: 10.1.2
URL	http://10.1.2.6/WackoPicko/admin/index.php?page=login
Method	GET
Parameter	page
Attack	GET http://10.1.2.6/WackoPicko/admin/index.php? page=eHlTmrBiyIKOemoxRnhOkqWusygGopRiYSxBhbYCyoMhSydTeMaKNZrEfHRbfhwhFMLXSxcLcCAlrsHYfNOoSvulxFZpRoiSvEVdAyLaKWlcGymTnyhovYpNeeTltiXXxQcWSiNwugCWcrpypNrFiXMYiieOYRFgxBJCGWYAdUGCYJmTlyDWs(Mozilla/5.0 (Windows NT 6.3; WOW64; rv:39.0) Gecko/20100101 Firefox/39.0 Pragma: no-cache Cache-Control: no-cache Content-Length: 0 Referer: http://10.1.2.6/WackoPicko/ Cookie: PHPSESSID=knn2ggc3jhebm30jfnm8m434f4 Host: 10.1.2
Instances	3
Solution	Rewrite the background program using proper return length checking. This will require a recompile of the background executable.
Other information	Potential Buffer Overflow. The script closed the connection and threw a 500 Internal Server Error
Reference	https://www.owasp.org/index.php/Buffer_overflow_attack
CWE Id	120
WASC Id	7
Source ID	1

Medium (Medium)	Format String Error
Description	A Format String error occurs when the submitted data of an input string is evaluated as a command by the application.
URL	http://10.1.2.6/WackoPicko/admin/index.php?page=ZAP%25n%25s%0A
Method	GET
Parameter	page
Attack	ZAP%n%s%n%s%n%s%n%s%n%s%n%s%n%s%n%s%n%s%n%s%n%s%n%s%n%s%n%s%n%s%n%s%n%s%n%s%n%s%n%s%n%s
URL	http://10.1.2.6/WackoPicko/admin?query=ZAP%25n%25s%0A
Method	GET
Parameter	query
Attack	ZAP%n%s%n%s%n%s%n%s%n%s%n%s%n%s%n%s%n%s%n%s%n%s%n%s%n%s%n%s%n%s%n%s%n%s%n%s%n%s%n%s%n%s
URL	http://10.1.2.6/WackoPicko/admin/index.php?page=ZAP%25n%25s%0A
Method	POST
Parameter	page
Attack	ZAP%n%s%n%s%n%s%n%s%n%s%n%s%n%s%n%s%n%s%n%s%n%s%n%s%n%s%n%s%n%s%n%s%n%s%n%s%n%s%n%s%n%s
Instances	3
Solution	Rewrite the background program using proper deletion of bad character strings. This will require a recompile of the background executable.
Other information	Potential Format String Error. The script closed the connection on a /%s
Reference	https://www.owasp.org/index.php/Format_string_attack
CWE Id	134
WASC Id	6
Source ID	1

Low (Medium)	X-Content-Type-Options Header Missing
Description	The Anti-MIME-Sniffing header X-Content-Type-Options was not set to 'nosniff'. This allows older versions of Internet Explorer and Chrome to perform MIME-sniffing on the response body, potentially causing the response body to be interpreted and displayed as a content type other than the declared content type. Current (early 2014) and legacy versions of Firefox will use the declared content type (if one is set), rather than performing MIME-sniffing.
URL	http://10.1.2.6/WackoPicko/
Method	GET
Parameter	X-Content-Type-Options
URL	http://10.1.2.6/WackoPicko/tos.php
Method	GET
Parameter	X-Content-Type-Options
URL	http://10.1.2.6/WackoPicko/pictures/search.php?query=ZAP
Method	GET
Parameter	X-Content-Type-Options
URL	http://10.1.2.6/WackoPicko/css/blueprint/screen.css
Method	GET
Parameter	X-Content-Type-Options
URL	http://10.1.2.6/WackoPicko/guestbook.php
Method	GET

Parameter	X-Content-Type-Options
URL	http://10.1.2.6/WackoPicko/css/stylings.php
Method	GET
Parameter	X-Content-Type-Options
URL	http://10.1.2.6/WackoPicko/css/blueprint/print.css
Method	GET
Parameter	X-Content-Type-Options
URL	http://10.1.2.6/WackoPicko/pictures/recent.php
Method	GET
Parameter	X-Content-Type-Options
URL	http://10.1.2.6/WackoPicko/guestbook.php
Method	POST
Parameter	X-Content-Type-Options
URL	http://10.1.2.6/WackoPicko/css/blueprint/ie.css
Method	GET
Parameter	X-Content-Type-Options
URL	http://10.1.2.6/WackoPicko/users/sample.php?userid=1
Method	GET
Parameter	X-Content-Type-Options
URL	http://10.1.2.6/WackoPicko/calendar.php
Method	GET
Parameter	X-Content-Type-Options
URL	http://10.1.2.6/WackoPicko/upload/doggie/Dog.jpg.128_128.jpg
Method	GET
Parameter	X-Content-Type-Options
URL	http://10.1.2.6/WackoPicko/upload/toga/togasfs.128_128.jpg
Method	GET
Parameter	X-Content-Type-Options
URL	http://10.1.2.6/WackoPicko/upload/house/hodjjgld.128_128.jpg
Method	GET
Parameter	X-Content-Type-Options
URL	http://10.1.2.6/WackoPicko/upload/flowers/flweofoee.128_128.jpg
Method	GET
Parameter	X-Content-Type-Options
URL	http://10.1.2.6/WackoPicko/calendar.php?date=1558972848
Method	GET
Parameter	X-Content-Type-Options
URL	http://10.1.2.6/WackoPicko/users/login.php
Method	GET
Parameter	X-Content-Type-Options
URL	http://10.1.2.6/WackoPicko/admin/index.php?page=login

Method	POST
Parameter	X-Content-Type-Options
URL	http://10.1.2.6/WackoPicko/calendar.php?date=1559145648
Method	GET
Parameter	X-Content-Type-Options
Instances	32
Solution	Ensure that the application/web server sets the Content-Type header appropriately, and that it sets the X-Content-Type-Options header to 'nosniff' for all web pages. If possible, ensure that the end user uses a standards-compliant and modern web browser that does not perform MIME-sniffing at all, or that can be directed by the web application/web server to not perform MIME-sniffing.
Other information	This issue still applies to error type pages (401, 403, 500, etc) as those pages are often still affected by injection issues, in which case there is still concern for browsers sniffing pages away from their actual content type. At "High" threshold this scanner will not alert on client or server error responses.
Reference	http://msdn.microsoft.com/en-us/library/ie/gg622941%28v=vs.85%29.aspx https://www.owasp.org/index.php/List_of_useful_HTTP_headers
CWE Id	16
WASC Id	15
Source ID	3

Low (Medium)	**Cookie No HttpOnly Flag**
Description	A cookie has been set without the HttpOnly flag, which means that the cookie can be accessed by JavaScript. If a malicious script can be run on this page then the cookie will be accessible and can be transmitted to another site. If this is a session cookie then session hijacking may be possible.
URL	http://10.1.2.6/WackoPicko/
Method	GET
Parameter	PHPSESSID
Evidence	Set-Cookie: PHPSESSID
Instances	1
Solution	Ensure that the HttpOnly flag is set for all cookies.
Reference	http://www.owasp.org/index.php/HttpOnly
CWE Id	16
WASC Id	13
Source ID	3

Low (Medium)	**Password Autocomplete in Browser**
Description	The AUTOCOMPLETE attribute is not disabled on an HTML FORM/INPUT element containing password type input. Passwords may be stored in browsers and retrieved.
URL	http://10.1.2.6/WackoPicko/passcheck.php
Method	POST
Parameter	password
Evidence	<input type="password" name="password" />
URL	http://10.1.2.6/WackoPicko/users/register.php
Method	GET
Parameter	password
Evidence	<input type="password" name="password" />
URL	http://10.1.2.6/WackoPicko/admin/index.php?page=login

Method	POST
Parameter	password
Evidence	<input type="password" name="password" />
URL	http://10.1.2.6/WackoPicko/users/login.php
Method	GET
Parameter	password
Evidence	<input type="password" name="password" />
URL	http://10.1.2.6/WackoPicko/passcheck.php
Method	GET
Parameter	password
Evidence	<input type="password" name="password" />
URL	http://10.1.2.6/WackoPicko/admin/index.php?page=login
Method	GET
Parameter	password
Evidence	<input type="password" name="password" />
URL	http://10.1.2.6/WackoPicko/users/register.php
Method	GET
Parameter	againpass
Evidence	<input type="password" name="againpass" />
Instances	7
Solution	Turn off the AUTOCOMPLETE attribute in forms or individual input elements containing password inputs by using AUTOCOMPLETE='OFF'.
Reference	http://www.w3schools.com/tags/att_input_autocomplete.asp https://msdn.microsoft.com/en-us/library/ms533486%28v=vs.85%29.aspx
CWE Id	525
WASC Id	15
Source ID	3

Low (Medium)	**Web Browser XSS Protection Not Enabled**
Description	Web Browser XSS Protection is not enabled, or is disabled by the configuration of the 'X-XSS-Protection' HTTP response header on the web server
URL	http://10.1.2.6/WackoPicko/users/login.php
Method	GET
Parameter	X-XSS-Protection
URL	http://10.1.2.6/WackoPicko/calendar.php
Method	GET
Parameter	X-XSS-Protection
URL	http://10.1.2.6/WackoPicko/users/sample.php?userid=1
Method	GET
Parameter	X-XSS-Protection
URL	http://10.1.2.6/WackoPicko/guestbook.php
Method	POST
Parameter	X-XSS-Protection

URL	http://10.1.2.6/WackoPicko/passcheck.php
Method	GET
Parameter	X-XSS-Protection
URL	http://10.1.2.6/WackoPicko/guestbook.php
Method	GET
Parameter	X-XSS-Protection
URL	http://10.1.2.6/sitemap.xml
Method	GET
Parameter	X-XSS-Protection
URL	http://10.1.2.6/WackoPicko/
Method	GET
Parameter	X-XSS-Protection
URL	http://10.1.2.6/WackoPicko/passcheck.php
Method	POST
Parameter	X-XSS-Protection
URL	http://10.1.2.6/WackoPicko/calendar.php?date=1558972848
Method	GET
Parameter	X-XSS-Protection
URL	http://10.1.2.6/WackoPicko/pictures/search.php?query=ZAP
Method	GET
Parameter	X-XSS-Protection
URL	http://10.1.2.6/WackoPicko/tos.php
Method	GET
Parameter	X-XSS-Protection
URL	http://10.1.2.6/robots.txt
Method	GET
Parameter	X-XSS-Protection
URL	http://10.1.2.6/WackoPicko/calendar.php?date=1559059248
Method	GET
Parameter	X-XSS-Protection
URL	http://10.1.2.6/WackoPicko/calendar.php?date=1559232048
Method	GET
Parameter	X-XSS-Protection
URL	http://10.1.2.6/WackoPicko/users/register.php
Method	GET
Parameter	X-XSS-Protection
URL	http://10.1.2.6/WackoPicko/pic'%20+%20'check'%20+%20'.php
Method	POST
Parameter	X-XSS-Protection
URL	http://10.1.2.6/WackoPicko/admin/index.php?page=login
Method	POST

Parameter	X-XSS-Protection
URL	http://10.1.2.6/WackoPicko/calendar.php?date=1559145648
Method	GET
Parameter	X-XSS-Protection
URL	http://10.1.2.6/WackoPicko/pictures/recent.php
Method	GET
Parameter	X-XSS-Protection
Instances	22
Solution	Ensure that the web browser's XSS filter is enabled, by setting the X-XSS-Protection HTTP response header to '1'.
Other information	The X-XSS-Protection HTTP response header allows the web server to enable or disable the web browser's XSS protection mechanism. The following values would attempt to enable it: X-XSS-Protection: 1; mode=block X-XSS-Protection: 1; report=http://www.example.com/xss The following values would disable it: X-XSS-Protection: 0 The X-XSS-Protection HTTP response header is currently supported on Internet Explorer, Chrome and Safari (WebKit). Note that this alert is only raised if the response body could potentially contain an XSS payload (with a text-based content type, with a non-zero length).
Reference	https://www.owasp.org/index.php/XSS_(Cross_Site_Scripting)_Prevention_Cheat_Sheet https://blog.veracode.com/2014/03/guidelines-for-setting-security-headers/
CWE Id	933
WASC Id	14
Source ID	3