

**Sistemas de Software Seguros (SSS)**

Mestrado em Segurança Informática

**Segurança de Software (SS)**

Mestrado em Engenharia Informática

Mestrado em Informática

**2024/2025**

**Projeto**

Gustavo Henriques Nº 64361

Leonardo Monteiro Nº 58250

Maria Figueirinhas Nº 46494

Índice

[Primeira Parte 3](#_Toc184166236)

[2. vulnApp 3](#_Toc184166237)

[01 – Characterize the attack surface of each application. TODO – FAZER O MIME 4](#_Toc184166238)

[01.1 – Map\_path 4](#_Toc184166239)

[01.2 – MIME 4](#_Toc184166240)

[01.3 – Mysig 5](#_Toc184166241)

[02 – Map\_path ( buffer overflows ) 5](#_Toc184166242)

[03 – Map\_path ( input that exploits the identified vulnerabilities ) 6](#_Toc184166243)

[04 – *Mime* ( buffer overflows ) 6](#_Toc184166244)

[05 – Mime ( solution ) 6](#_Toc184166245)

[06 – Mysig ( integer overflow ) 7](#_Toc184166246)

[07 – Mysig ( solution ) 7](#_Toc184166247)

[3. SSS-DB 7](#_Toc184166248)

[08 – SQL Injection 7](#_Toc184166249)

[09 – Stored XSS Attack 8](#_Toc184166250)

[10 – Reflected XSS Attack 9](#_Toc184166251)

[11 – Reflected XSS Attack ( solution ) 9](#_Toc184166252)

[12 – Command injection 10](#_Toc184166253)

[13 – Command injection ( solution ) 11](#_Toc184166254)

[Segunda Parte 11](#_Toc184166255)

[4. FlawFinder 11](#_Toc184166256)

[14 - Error report 11](#_Toc184166257)

[15 - Same command with the -F option 13](#_Toc184166258)

[6. AFL 13](#_Toc184166259)

[16 - Using the AFL fuzzer tool to discover vulnerabilities 13](#_Toc184166260)

[7. Large Language Models 13](#_Toc184166261)

[17 – Analysing if each code is vulnerable or secure 13](#_Toc184166262)

[18 – Prompt engineering with ChatGPT 13](#_Toc184166263)

[19 – Ask ChatGPT to fix/patch the codes 13](#_Toc184166264)

íNDICE DE FIGURAS

[Figura 1. Code execution result for map\_path attack. 5](#_Toc184166675)

[Figura 2. Solution for the SQL Injection. 7](#_Toc184166676)

[Figura 3. Results of the SQL Injection attack. 7](#_Toc184166677)

[Figura 4. Results of the stored XSS attack. 8](#_Toc184166678)

[Figura 5. Solution of the Reflected XSS attack. 8](#_Toc184166679)

[Figura 6. Result of the Reflected XSS attack. 9](#_Toc184166680)

[Figura 7. Field of input that can suffer a command. 10](#_Toc184166681)

[Figura 8. File to be attacked. 10](#_Toc184166682)

[Figura 9. Input solution for the command injection. 10](#_Toc184166683)

[Figura 10. Results of the FlawFinder report for Mysig. 11](#_Toc184166684)

# Primeira Parte

## 2. vulnApp

### 01 – Characterize the attack surface of each application. TODO – FAZER O MIME

### 01.1 – Map\_path

A *attack surface* do *map\_path* é o conteúdo do ficheiro input (file system). Uma vez que nada dentro do programa verifica se o utilizador tem acesso às diretorias indicadas pelo ficheiro input no terminal, correr o programa permite a entrada em qualquer diretoria desejada. A partir daí, é possível entrar onde o atacante quiser e, eventualmente, aceder aos ficheiros aí presentes. Aqui, a função vai concatenando o caminho dos diretórios à variável *mapped\_path*, sem fazer uma verificação de segurança adequada no conteúdo de *dir*. Isto permite que um atacante insira partes do caminho como ".." para manipular o caminho e aceder a diretórios superiores.

strcat(mapped\_path, "/");

strcat(mapped\_path, dir);

if ((ret = chdir(mapped\_path)) < 0){

printf("couldn't chdir to %s !\n", mapped\_path);

strcpy(mapped\_path, old\_mapped\_path);

}

Aqui, a função tenta mudar o diretório para o valor acumulado em *mapped\_path*, o que pode ser controlado por um atacante se *orig\_path* tiver valores manipulados. Isto abre a possibilidade de mudar entre diretórios, permitindo acesso a áreas não autorizadas do sistema de ficheiros.

f = fopen(argv[1], "r");

fgets(orig\_path, MAXPATHLEN + 20, f);

fclose(f);

map\_dir\_chdir(orig\_path);

O código lê o caminho original de um ficheiro (através de *fopen* e *fgets*) e depois tenta mudar de diretoria para esse caminho com *map\_dir\_chdir*. Se um atacante puder modificar o conteúdo do ficheiro entre a abertura e a leitura, ou se manipular o sistema de ficheiros entre as chamadas a *fopen* e *chdir*, este pode alterar o comportamento do programa, levando a um aumento de privilégios ou à execução de operações indesejadas.

### 01.2 – MIME

A superfície de ataque do MIME é o conteúdo do ficheiro de entrada. Este e um tipo de ataque de files system /User interface. O programa lê diretamente o arquivo fornecido pelo utilizador, sem validar adequadamente o conteúdo, permitindo que um atacante forneça um ficheiro malicioso. Isso pode resultar em ataques como **Path Traversal**, **execução de código malicioso** ou **negação de serviço (DoS)** através de ficheiros manipulados. A linha de código que permite isso é:

temp = fopen(argv[1], "r");

### 01.3 – Mysig

Agora em relação ao mysig. Este código implementa uma parte de um servidor DNS (Domain Name System), responsável por lidar com consultas de resolução de nomes de domínio. O código constrói pacotes de consulta DNS, empacotando informações como nomes de domínio em um formato adequado para serem enviados a servidores DNS. Além disso, o servidor envia essas consultas para servidores específicos e processa as respostas que recebe, extraindo registos de recursos, como endereços IP, associados aos domínios consultados. O código faz uso de alocação de memória dinâmica para armazenar temporariamente dados, como os nomes de domínio e as respostas DNS, e realiza operações de manipulação de strings e buffers. Em resumo, a principal função do código é atuar como um cliente de resolução DNS, montando e enviando consultas, recebendo as respostas e retornando informações sobre os domínios consultados.

No caso deste código, a attack surface está principalmente relacionada ao sistema operacional , especialmente devido à manipulação de memória dinâmica, interações com o sistema de arquivos e rede e uso de funções para manipulação de strings e buffers. O código contém interações com o sistema de arquivos (por exemplo, com malloc para alocar buffers para armazenar dados de rede) e com a rede (enviando pacotes DNS). Em sistemas operacionais, essa interação pode ser uma linha de ataque, pois o acesso a essas camadas envolve manipulação de buffers e de dados externos que podem ser comprometidos. Para além, como o código usa funções como strcpy, memcpy e dn\_expand podem existir vulnerabilidades de buffer overflow devido a estas funções não fazerem verificações de limite de tamanho.

### 02 – Map\_path ( buffer overflows )

No programa *map\_path*, há cinco *buffer overflows*. Acontecem nas linhas 30, 68, 69, 83, 86 e 112. Na linha 30 e 86, ocorrem pois está a ser utilizada a função *strcpy* (copia uma string). O problema de utilizar esta função é que não é feita nenhuma verificação de tamanho da string que vai ser copiada em relação ao local de destino.

Na linha 30, *strcpy* copia *mapped\_path* para *path*. A variável *mapped\_path* é uma string contida dentro do input de main, que representa o caminho passado como argumento. Esta variável pode dar overflow no buffer criado para *path*.

strcpy(path, mapped\_path);

Caso *mapped\_path* seja maior ou igual a *path* (\0 será acrescentado no final da string, causando buffer overflow nestes casos também), ocorrerá overflow.

A função *strcat* concatena *strings* sem verificar se o tamanho do resultado ultrapassa o espaço reservado para a operação. É este o caso na linha 68 e 69, com as seguintes linhas:

strcat ( mapped\_path, “/” );

strcat ( mapped\_path, dir );

A variável *mapped\_path* é concatenada com “/” e depois com *dir*. O resultado pode eventualmente originar um *buffer overflow*, dependendo do input inicial.

Na linha 83, o buffer overflow ocorre devido a um strcpy, de acordo com o código seguinte:

83. strcpy (old\_mapped\_path, mapped\_path);

A variável old\_*mapped*\_path está definida na linha 36 como sendo um buffer de tamanho igual a MAXPATHLEN. A variável *mapped*\_path é um buffer com tamanho igual a MAXPATHLEN também, inicializada apenas contendo “/”. No entanto, a partir da linha 41, há uma série de verificações sobre o conteúdo de dir e de mapped\_path, culminando no possível buffer overflow mencionado nas linhas 68 e 69. A variável mapped\_path pode já estar em buffer overflow por esta fase, dado que dir é fornecido pelo utilizador e o seu tamanho nunca é verificado. Ao copiar esta string em possível overflow para o old\_mapped\_path, cujo tamanho é fixo, e, usando strcpy, que não verifica o tamanho da string a ser copiada, então há a forte possibilidade de gerar um buffer overflow também nesta variável.

Na linha 86, a variável orig\_path é copiada para a variável path. Esta variável é definida na linha 84 como sendo um pointer para a lista pathspace, que tem de tamanho maxpathlen, definido, na linha 35. Orig\_path é definida na linha 168, na main, com tamanho superior a maxpathlen e o seu conteúdo é o caminho original para o ficheiro em questão, que pode ser maior que maxpathlen. Uma vez que a cópia é feita utilizando strcpy, não é feita nenhuma verificação de tamanho em relação ao que é copiado e para onde, pelo que pode acontecer um buffer overflow como resultado.

### 03 – Map\_path ( input that exploits the identified vulnerabilities )

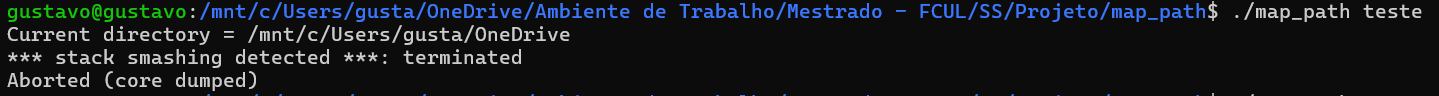
Para atacarmos as vulnerabilidades do Map\_path criamos um ficheiro teste com mais de 20 caracteres que era o máximo de caracteres que as variáveis podiam ter ( 19 caracteres mais o /0 ). E com isto causamos um buffer overflow tendo o código mudado para a diretoria que metemos no input e corrompido a pilha de execução do programa, como podemos ver na imagem abaixo:

Figura 1. Code execution result for map\_path attack.

### 04 – *Mime* ( buffer overflows )

*Este programa é um descodificador básico para mensagens MIME codificadas em base64. Lê um arquivo especificado pelo utilizador, verifica se está codificado em base64 (usando o cabeçalho MIME), e, se estiver, descodifica o conteúdo de volta para um formato de 8 bits.*

*Possível buffer overflow:*

*136. if (\*fbufp++ == '\n' || fbuf >= &fbuf[MAXLINE]) {*

*...*

*}*

Na função *MIME\_func*, o ponteiro *fbufp* é utilizado para gravar dados descodificados no buffer *fbuf*. Este buffer tem um tamanho fixo, definido por *MAXLINE* + 1. *fbufp* começa por apontar para o início de *fbuf*, mas é continuamente incrementado à medida que os dados são gravados. O problema está na verificação indireta e inadequada que o código faz para garantir que *fbufp* não ultrapasse o tamanho do buffer. A condição atual verifica apenas o ponteiro base *fbuf*, e não *fbufp*, o que permite que *fbufp* continue a ser incrementado e acabe por escrever fora dos limites de *fbuf*. Isso causa uma vulnerabilidade de *buffer overflow*, permitindo que o programa escreva em áreas não alocadas da *heap*, corrompendo dados e afetando variáveis e estruturas adjacentes. Esta vulnerabilidade pode ser explorada por um invasor para alterar o comportamento do programa, escrevendo fora do espaço alocado para o programa.

### 05 – Mime ( solution )

Para evitar o *buffer overflow*, precisamos garantir que o ponteiro *fbufp* nunca ultrapasse o limite do *buffer fbuf*. A sugestão é alterar a condição de verificação para comparar o *fbufp* diretamente com o final do *buffer*.

Em vez disto:

if (\*fbufp++ == '\n' || fbuf >= &fbuf[MAXLINE]) {

Fazer:

if \*(fbufp - 1) == '\n' || fbufp >= &fbuf[MAXLINE]) {

Esta versão garante que *fbufp* não ultrapassa o limite de *fbuf*: (fbufp >= &fbuf[MAXLINE]) impede que *fbufp* aponte para fora do *buffer*. Do mesmo modo, a primeira condição na solução verifica se o último caracter dentro de fbuf antes da posição atual é o newline (\n). Isto serve para ter a certeza de que o buffer está a ser resetted na altura correta.

### 06 – Mysig ( integer overflow )

Na função *RRextract*, o valor de *dlen* (comprimento dos dados do RR) é extraído diretamente da mensagem DNS recebida: *GETSHORT(dlen, cp)*; aqui, *dlen* é extraído de uma posição na mensagem DNS (o ponteiro *cp*), que está fora do controlo da aplicação. Isso significa que um atacante pode manipular esse valor no pacote DNS para ser maior do que o esperado, causando assim um *integer overflow*. Após ler *dlen*, o código tenta garantir que a área referida por *cp* tem espaço suficiente para ler os dados, usando a macro *BOUNDS\_CHECK: BOUNDS\_CHECK*(*cp, dlen*); A macro *BOUNDS\_CHECK* verifica se o ponteiro *cp* mais o valor *dlen* não ultrapassa o fim da mensagem (eom), mas isso não verifica se o *buffer* onde os dados serão armazenados tem espaço suficiente para os conter.

Esta verificação é limitada à estrutura da mensagem DNS, mas não protege contra um *buffer overflow* quando os dados são copiados ou processados. A cópia dos dados do RR para o buffer é feita logo após a verificação de limites, sem verificar se o espaço disponível no *buffer* é suficiente para os dados que estão a ser copiados: *rdatap = cp*;

Neste ponto, *rdatap* (um ponteiro para a posição atual no *buffer*) é configurado para apontar para os dados que serão lidos de *cp*. Se o valor de *dlen* for maior do que o espaço disponível no *buffer*, a função poderá copiar mais dados do que o *buffer* suporta, causando um *buffer overflow* nesta variável rdatap*.*

### 07 – Mysig ( solution )

Uma alteração que pode impedir a utilização desta vulnerabilidade é verificar o valor de *dlen* antes de o usar:

if (dlen > sizeof(data)) {

printf("Data length exceeds buffer size\n");

hp->rcode = FORMERR; // Código de erro

return -1;

}

Isso garantiria que qualquer operação de leitura ou escrita não excederia o tamanho do *buffer*, prevenindo o *buffer overflow*.

## 3. SSS-DB

### 08 – SQL Injection

No nível de segurança 0, o código permite a execução de *SQL Injection* porque as variáveis não são escapadas ou saneadas antes de serem usadas na *query*. Este é um caso típico de vulnerabilidade onde um utilizador mal-intencionado pode explorar essa fraqueza para obter acesso não autorizado ao sistema ou até mesmo manipular os dados na base de dados.

$lQuery = "SELECT \* FROM accounts WHERE username='".

$lUsername .

"' AND password='" .

$lPassword .

"'";

Ao analisar este código SQL é possível fazer uma SQL Injection ao meter no Username a seguinte string:

' OR '1'='1' --

Figura 2. Solution for the SQL Injection.

Uma imagem com texto, documento, captura de ecrã, recibo

Descrição gerada automaticamente Ao inserir isto conseguimos obter a seguinte lista de nomes:

Figura 3. Results of the SQL Injection attack.

### 09 – Stored XSS Attack

Durante a criação de um novo utilizador, não ocorre qualquer saneamento do *input* que é fornecido pelo utilizador antes de exibir a página onde este fica guardado. Isto torna a opção *Show Log* vulnerável a ataques de *stored XSS*. Um atacante pode colocar no username um script malicioso que será guardado e chamado de novo quando qualquer utilizador aceder à página do *Show Log* e então, será executado como parte do script da página.

Para realizar o ataque, no momento de criação de user, no campo de input Username, inserimos o seguinte input:

<img src=x onerror=alert(&#39;hello&#39;)>

Com isto criamos um *XSS stored attack* que irá fazer com que apareça um *popup*, tanto no momento de criação de *user* como quando se tenta procurar o nome do utilizador na opção *Show Log* no *Core Control*.

Uma imagem com texto, captura de ecrã, software, Software de multimédia

Descrição gerada automaticamente

Figura 4. Results of the stored XSS attack.

### 10 – Reflected XSS Attack

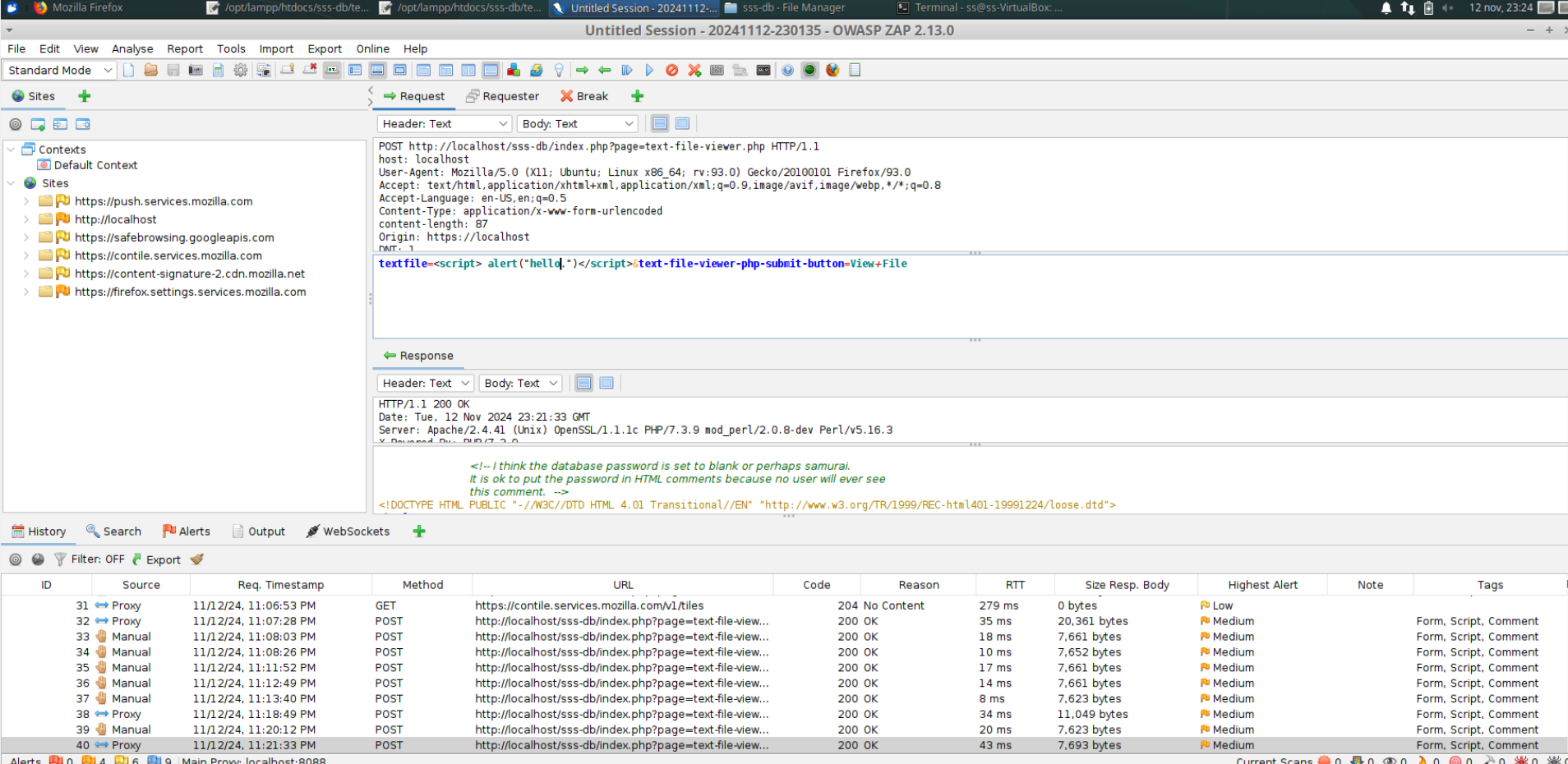
O ataque funciona alterando o nome do textfile no request que o cliente faz ao servidor. O servidor não faz qualquer saneamento do request e simplesmente corre o nome do .txt como um script, criando o pop up.

Figura 5. Solution of the Reflected XSS attack.

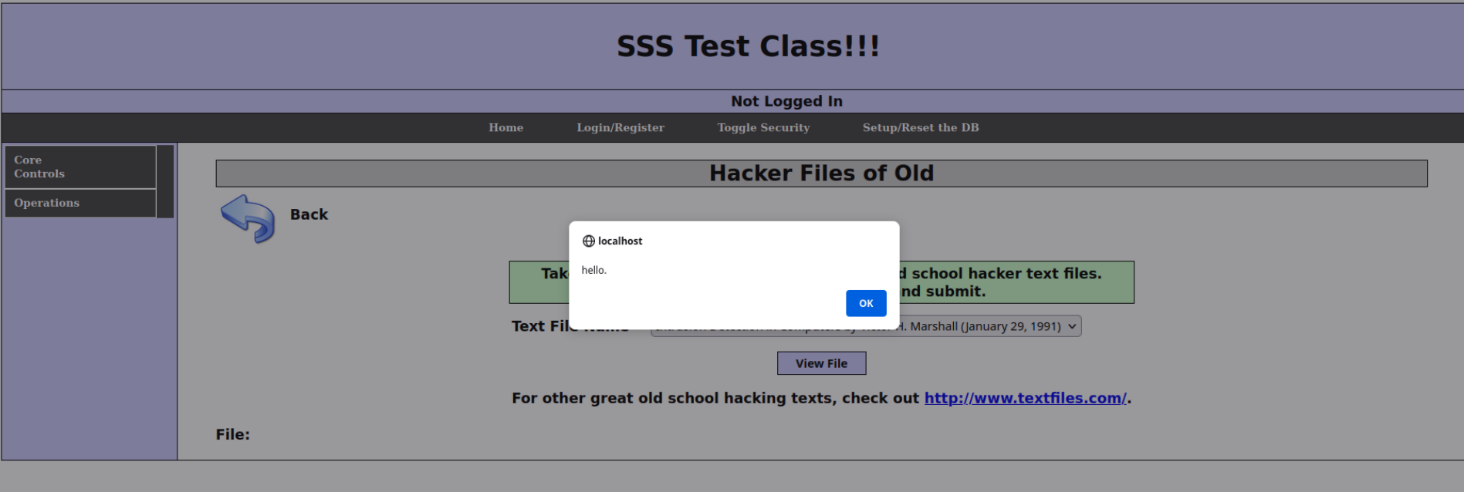


Figura 6. Result of the Reflected XSS attack.

### 11 – Reflected XSS Attack ( solution )

A vulnerabilidade existe porque o parâmetro "textfile" não é corretamente validado ou escapado antes de ser utilizado na página web. Quando o utilizador seleciona um ficheiro, o valor do parâmetro "textfile" é incluído diretamente na resposta HTML sem qualquer sanitização. Isto permite que um atacante manipule a requisição, inserindo um script malicioso que será executado no navegador do utilizador quando a página for carregada. Para mitigar esta vulnerabilidade, a aplicação deve garantir que qualquer entrada do utilizador, como o valor do parâmetro "textfile", seja rigorosamente validada e escapada antes de ser renderizada no HTML.

Em vez de permitir que o parâmetro "textfile" contenha qualquer valor, a aplicação poderia restringi-lo a identificadores específicos que correspondam a ficheiros válidos. Assim, ao invés de aceitar diretamente caminhos ou nomes de ficheiros, o sistema deveria utilizar um esquema de IDs para representar os ficheiros, reduzindo a possibilidade de injeção de código malicioso. Além disso, a implementação de tokenização para níveis de segurança mais altos contribui para proteger o sistema, uma vez que assegura que o valor do parâmetro apenas corresponda a números predefinidos que representam ficheiros legítimos, prevenindo um reflected XSS attack.

### 12 – Command injection

Figura 7. Field of input that can suffer a command.

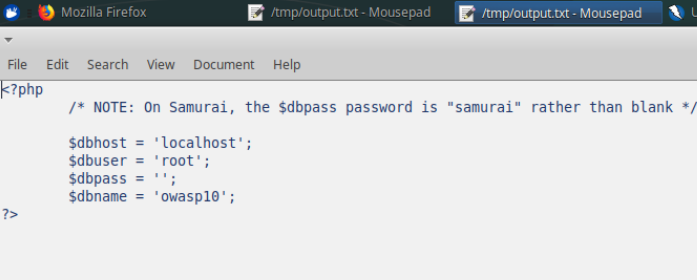


Figura 8. File to be attacked.

Para fazer este ataque descobrimos que o ficheiro que queremos é o config.inc, mostrado em cima. Com isto, primeiramente tentamos diretamente imprimir este ficheiro com o comando “; cat /opt/lampp/htdocs/sss-db/config.inc” mas, como o código do ficheiro está em php não era possível ele ser impresso na aplicação. Para solucionar este problema, decidimos copiar o código que estava neste ficheiro mas sem a primeira e última linha, de seguida, bastava imprimir este ficheiro e desta maneira conseguimos com que aparecesse na app, como podemos ver na imagem abaixo. Assim, conseguimos arranjar uma maneira de ver o conteúdo deste ficheiro sem sequer utilizar o proxy:

Figura 9. Input solution for the command injection.

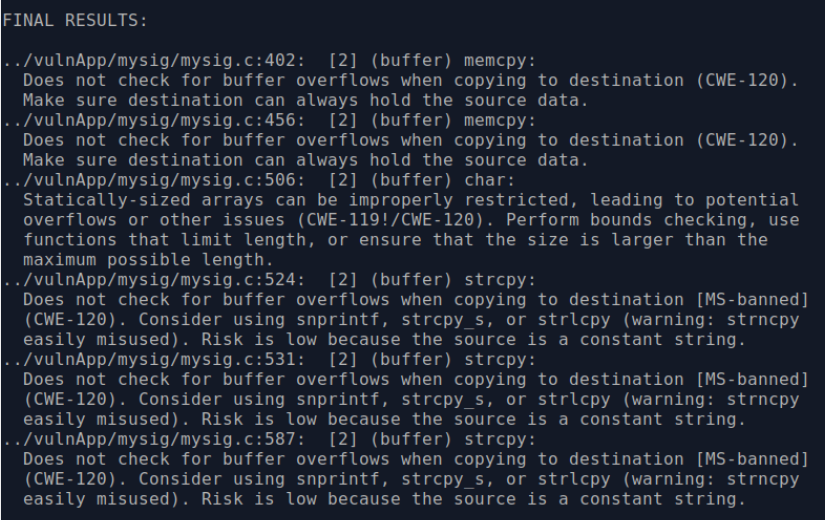
### 13 – Command injection ( solution )

A vulnerabilidade de injeção de comandos existe porque a aplicação passa o valor do parâmetro "target\_host" diretamente para o comando dnslookup sem a devida sanitização, especialmente nos níveis de segurança baixos. Nos níveis de segurança 0 e 1, o valor de "target\_host" é obtido diretamente através de $\_REQUEST e não é validado, o que permite a injeção de caracteres especiais como ; ou & para encadear comandos adicionais. Assim, um atacante pode manipular o campo "target\_host" para executar comandos no servidor, obtendo, por exemplo, informações sensíveis como as credenciais de acesso à base de dados, explorando a vulnerabilidade para executar comandos arbitrários.

Para proteger a aplicação contra este tipo de ataque, é crucial validar e sanitizar adequadamente os inputs do utilizador. Uma forma eficaz é permitir apenas valores específicos, como endereços IP ou nomes de domínio válidos, utilizando expressões regulares para verificar o formato do input, o que já é feito nos níveis de segurança mais altos. Além disso, a aplicação deve evitar o uso direto de funções que executam comandos do sistema operacional com inputs fornecidos pelo utilizador. Em vez disso, poderia recorrer a APIs ou bibliotecas específicas para resolver DNS sem necessidade de comandos externos, mitigando o risco de injeção de comandos.

# Segunda Parte

## 4. FlawFinder



### 14 - Error report

Figura 10. Results of the FlawFinder report for Mysig.

|  |  |  |
| --- | --- | --- |
| Error | Rating | Why |
| Mysig 402  memcpy(cp1, cp, 18); | FALSE POSITVE | O tamanho que estamos a passar buffer de destino (cp1) é validado através do BOUNDS\_CHECK na linha anterior. Logo apesar de nesta linha não ser feita nenhuma verificação, pela linha anterior conclui-se que esta vulnerabilidade não pode ser explorada. |
| Mysig 456  memcpy(cp1, cp, n); | TRUE POSITIVE | O tamanho que estamos a passar para o cp1 pode ser maior do que esta variável consegue aceitar, causando assim um bufferoverflow. |
| Mysig 506  char exp\_dn[200], exp\_dn2[200]; | FALSE POSITIVE | As verificações de tamanho antes de acessar o buffer, garantem que não há risco de overflow. |
| Mysig 524  strcpy(temp, "HEADER JUNK:"); | FALSE POSITIVE | O tamanho do buffer é maior do que a string copiada. Temp é um char pointer com capacidade para 400 caracteres e a string que estamos a copiar tem 12 caracteres. |
| Mysig 531  strcpy(exp\_dn, "lcs.mit.edu"); | FALSE POSITIVE | O tamanho do buffer é maior do que a string copiada. Exp\_dn é um char com capacidade para 200 caracteres e a string que estamos a copiar tem 11 caracteres. |
| Mysig 587  strcpy(exp\_dn2, "ccs.ocs.fcul.pt"); | FALSE POSITIVE | O tamanho do buffer é maior do que a string copiada. Exp\_dn2 é um char com capacidade para 200 caracteres e a string que estamos a copiar tem 14 caracteres. |

### Uma imagem com texto, captura de ecrã, Tipo de letra15 - Same command with the -F option

A ferramenta Flawfinder, ao executar o comando com a opção -F, filtra potenciais False Positives (FP), mas nem sempre de forma precisa. Observámos que, a linha 506 identificada anteriormente como False Positives foi corretamente removida do output. Contudo, as outras que também analisámos como FP (402, 524, 531, 587) permanecem no resultado, mostrando que a ferramenta não consegue identificar consistentemente todas as situações que não representam vulnerabilidades reais. Com isto, concluímos que a ferramenta não é 100% precisa na distinção entre True Positives e False Positives. Este comportamento é esperado, pois Flawfinder é uma análise estática que não compreende totalmente o contexto do código.

## 6. AFL

### 16 - Using the AFL fuzzer tool to discover vulnerabilities

## 7. Large Language Models

### 17 – Analysing if each code is vulnerable or secure

### 18 – Prompt engineering with ChatGPT

### 19 – Ask ChatGPT to fix/patch the codes