# A blue and white sign with textSistemas de Segurança de Software

## Mestrado em Engenharia Informática

# Segurança de Software

## Mestrado em Informática

# Projeto – vulnApp

Gustavo Henriques Nº 64361

Leonardo Monteiro Nº 58250

Maria Figueirinhas Nº 46494

1 – Characterize the attack surface of each application. Justify your answer by relating it to the code.

A *attack surface* do *map\_path* é o conteúdo do ficheiro input. 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.

No caso de *mysig*, a attack surface é

if (exptime <= now) { printf("ignoring SIG: expiration %s is in the past", p\_secstodate (exptime)); return ((cp - rrp) + dlen); }

Através do código acima, se um atacante manipular os campos de tempo (como o tempo de expiração e o tempo de assinatura), pode tentar fazer com que a aplicação aceite assinaturas expiradas ou inválidas. Isso pode permitir ataques de falsificação de registos DNS.

(FALTA ATTACK SURFACE MIME)

02 – Map\_path

No programa *map\_path*, há cinco *buffer overflows*. Acontecem nas linhas 30, 35, 68, 69 e 86. 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 *orig\_path* para *path*. A variável *orig\_path* é uma string contida dentro do input de main. Esta variável pode dar overflow no buffer criado para *path*.

strcpy(path, *orig\_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. Na linha 35, *pathspace* cria um buffer com tamanho [MAXPATHLEN], (definido em *include-map.h* como tendo 20 *bytes* de tamanho). Este *buffer*, como será usado depois para (fazer cenas idk), também pode receber *input* maior do que o seu tamanho.

char pathspace[MAXPATHLEN]

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.

03 – *Map\_path*

04 – *Mime*

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, solução*

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++ == '\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*.

06 – *Mysig*

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*:

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

...

}

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.*

if (dlen > sizeof(data)) {

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

hp->rcode = FORMERR;

return (-1);

}

07 – *Mysig*

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

if (dlen > MAXDATA) {

printf("dlen exceeds the maximum allowed size\n");

hp->rcode = FORMERR;

return (-1);

}

Isto pode ser aplicado imediatamente após o valor de *dlen* ser extraído para garantir que não haja qualquer risco de processamento de tamanho de dados incorretos.

#define BOUNDS\_CHECK(ptr, count) \

do { \

if ((ptr) + (count) > eom || (count) > sizeof(data)) { \

hp->rcode = FORMERR; \

return (-1); \

} \

} while (0)

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