Ciências ULisboa

Sistemas de Segurança de Software

Mestrado em Engenharia Informática

Segurança de Software

Mestrado em Informática

Class Project: Experiments with Static Analysis

Gustavo Henriques Nº 64361

Leonardo Monteiro Nº 58250

Maria Figueirinhas Nº 46494

2. FlawFinder

Para analisar as vulnerabillidades fomos até à diretoria /home/ss/apps/flawfinder.
Depois disto corremos o comando seguinte:

./flawfinder -m 2 ../vulnApp/qwik-smtpd.c

Com isto o FlawFinder irá analisar todas as vulnerabilidades existentes no ficheiro qwik-smtpd.c. O relatório que nos foi fornecido indicou que o código tinha 64 vulnerabilidades. Iremos agora analisar as vulnerabilidades das linhas 152, 211 e 422, tal como pedido no enunciado.

Linha 152:

strcpy(localIP,getConfig("localip"));

Como podemos ver, nesta linha é feito um strcpy do getConfig("localip") para a variável localIP. Mais acima no código, o tamanho definido para esta variável é de 64 caracteres, como se pode observar pela linha seguinte " char localIP[64]; ". A função getConfig basicamente vai retornar a primeira linha do ficheiro que estiver na diretoria CONFIG_DIR/localip ou então a string "" caso o ficheiro não exista. Com isto concluimos que esta vulnerabilidade pode ser explorada, visto que ao mudarmos o que estiver na primeira linha do ficheiro referido acima (assumindo que este ficheiro não é confiável) podemos escrever uma string com mais de 64 caracteres, causando assim um bufferoverflow.

Linha 211:

```
fprintf(fpout,clientRcptTo[x]);
```

Para esta vulnerabilidade fomos explorar a variável clientRcptTo para ser possível perceber se de facto isto é uma vulnerabilidade. Reparamos que a mesma é inicializada nesta função push:

```
int push(char *data)
{
  clientRcptTo[clientRcpts] = (char*) malloc(64);
  strcpy(clientRcptTo[clientRcpts],data);
  clientRcpts++;
}
```

Depois disto fomos onde é a que a função push é chamada para conseguirmos perceber que tipo de data é passada o clientRcptTo e, vimos que sempre que esta função era chamada era com o arg3 “push(arg3);”. De seguida fomos à procura de como é que a variável arg3 era inicializada e, descobrimos que era chamada na linha 256 “parseInput(inputLine,arg1,arg2,arg3)”. Nesta função é passado para o arg3 o email address passado na linha de comandos. Logo concluímos assim que podemos fazer um format string ataque. Reparamos ainda que existiam algumas verificações nesta variável como, por exemplo verificar se continha um @ (por essa um email address) mas de qualquer maneira seria ainda assim possível escrever algo como “@%s” pois essa verificação não é feita, causando assim um format string ataque.

Linha 422:

```
  sprintf(messageID, "Message-ID: <%d.%d.qwikmail@%s>\n", s, myPid, localHost);
```

Aqui começamos por analisar o tamanho da variável MessageID (char messageID[128];). De seguida ao analisar o tamanho da string que será copiada para o MessageID percebe-se que o tamanho pode muito bem ultrapassar os 128, dado que só o tamanho da variável localHost já é 128, por isso imagindo que essa variável está no tamanho máximo, tendo conta os restante caracteres estariamos deste modo a causar um bufferoverflow.

3. WAP

a) Checking that the vulnerability exists

O programa encontrou duas vulnerabilidades dentro do ficheiro index.php. Analisando a primeira instância descrita como vulnerável:

98: $evid_filter = "and evid='$_GET[evid]'"; // This clause is used later for refreshing the event div

99: $query = "update $cfg[db_tabl] set eventState = NOT eventState, ownerid='".$_GET[login] . "' where 1 $evid_filter limit 1";

100: $result = mysql_query($query);

Nestas linhas de código, o utilizador fornece o input 'login', que é diretamente atualizado como sendo o 'ownerId', e colocado no array cfg, onde estão todas as configurações de permissão [cfg db_tabl. O código é vulnerável a SQL injections porque o login do utilizador é guardado diretamente como ownerId sem sanear contra possíveis entradas maliciosas.

Analisando agora a segunda vulnerabilidade descrita por esta ferramenta:

102: $log_query = "insert into $cfg[db_log] values ('$_GET[evid]','".$_GET[login]."',concat(utc_date(),' ',utc_time()),'Event state changed to $state from ZiPEC')";

103: @mysql_query($log_query);

Como podemos ver, tal como na primeira vulnerabilidade, o utilizador pode fazer uma SQL injection nos campos evid ou login, fazendo assim com que esta vulnerabilidade possa ser explorada.

b) Checking that the correction is correct

A correção apresentada para a primeira vulnerabilidade apresenta um problema significativo: ela removeu a funcionalidade relacionada ao campo login, essencialmente ignorando a vulnerabilidade existente nesse campo, em vez de solucioná-la. Desta maneira o código foge ao SQL injection ataque, mas o código fica funcionalmente errado.

A correção proposta para a segunda vulnerabilidade, apesar de sanear o campo evid deixa o campo login descoberto para sofrer uma sql injection, logo apesar de haver um saneamento numa das variáveis como podemos ver aqui:
".san_sqli(0, san_sqli(0, $_GET)[evid])."

A outra continua desprotegida, o que significa que esta correção não é suficiente para que o código fique protegido contra SQL injections ataques.