binScan Design Document

A binary scanning utility

By Mark McCloskey

Table of Contents

Software Components

The software consists of 8 main components loosely broken into their respective functions. The components are C files with their corresponding header files and include: commandLine.c, disasm.c, entropy.c, err.c, main.c, md5.c, parseElf.c, and finally an assembly file unpass.asm.

commandLine.c

commandLine.c primarily focuses on managing input and output from the user.

disasm.c

disasm.c contains the code used to disassemble binaries.

entropy.c

entropy.c calculates the entropy of the binary file.

err.c

err.c is a simple error handling function.

md5.c

md5.c calculates the md5 hash of the .text section of the binary.

parseElf.c

parseElf.c is the workhorse of the group and handles parsing and delegating tasks related to finding information about the ELF file.

unpass.asm

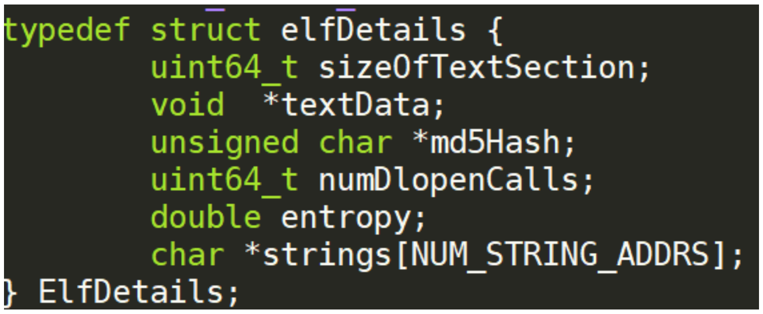
unpass.asm is an assembly function written to be used in binScan.

main.c

main.c is the driver of binScan and delegates work to all of the above components.

Application Storage Protocol

The storage protocol in the application is a structure named ElfDetails.



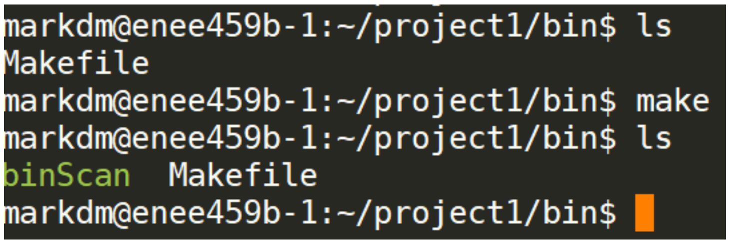
```
typedef struct elfDetails {
        uint64_t sizeOfTextSection;
        void  *textData;
        unsigned char *md5Hash;
        uint64_t numDlopenCalls;
        double entropy;
        char *strings[NUM_STRING_ADDRS];
} ElfDetails;
```

Storage Protocol

This structure is passed around and populated during analysis of a binary and the information is saved to disk in much the same form.

Build Instructions

Navigate to the /bin folder and run ‘make’, after the process completes the binScan executable will be in the directory.



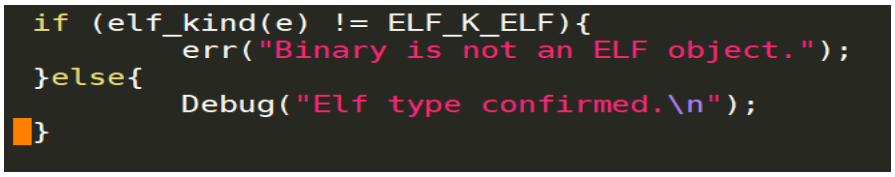
```
markdm@enee459b-1:~/project1/bin$ ls
Makefile
markdm@enee459b-1:~/project1/bin$ make
markdm@enee459b-1:~/project1/bin$ ls
binScan  Makefile
markdm@enee459b-1:~/project1/bin$ 
```

Building binScan

Requirements Mapping

This section will specify where in the project files individual requirements are met. When feasible a picture of the requirement will be included here otherwise line numbers in files or function names will have to suffice. I will number the requirements and restate what they were then map them.

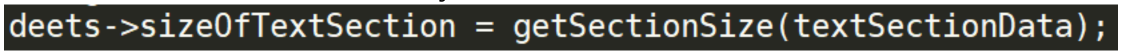
1. Analyze binaries built in the ELF format
2. The application should verify the format of the binary file before analyzing.
   All of parseElf.c serves this purpose and can be verified by running binScan on an ELF binary.



```
if (elf_kind(e) != ELF_K_ELF){
        err("Binary is not an ELF object.");
}else{
        Debug("Elf type confirmed.\n");
}
```

Requirements 1 and 2

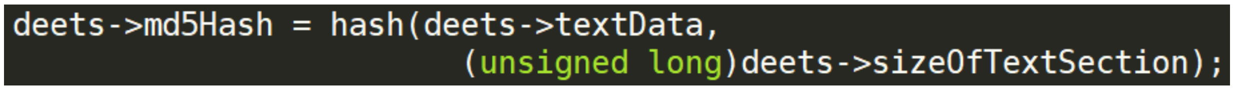
3. The application must collect 5 classes of information about each binary it analyzes.
   The binary collects the size of the text section, a hash of the text section, the entropy of the file, the number of calls to dlopen, and the strings passed to dlopen.
4. One of the attributes is the size in bytes of the .text section.



```
deets->sizeOfTextSection = getSectionSize(textSectionData);
```

Size of Text Section

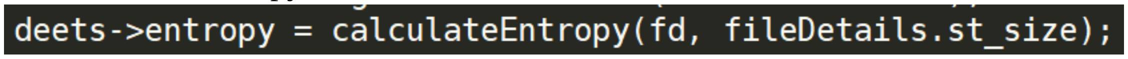
5. One must be an MD5 hash of the .text section.



```
deets->md5Hash = hash(deets->textData,
                        (unsigned long)deets->sizeOfTextSection);
```

Hash of text section

6. One must be the entropy of the file.

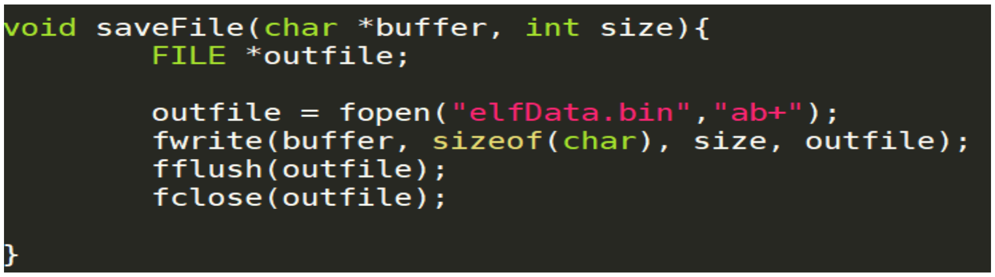


```
deets->entropy = calculateEntropy(fd, fileDetails.st_size);
```

Entropy Calculation of file done in entropy.c

7. One of the attributes is the number of calls to dlopen().
   The number of calls to dlopen is a hot mess starting on line 98 of parseElf.c, resolving those strings then begins on line 173.
8. The software shall use an original binary format for storage of data on disk.
   The function createFileBuffer in parseElf.c satisfies this requirement.
9. The software shall obfuscate the file contents on disk.
   The function fuzzFile in parseElf.c satisfies this requirement.
10. The software shall provide an authentication mechanism.
    The functions getUsername and getPassword in commandLine.c in conjunction with unpass.asm satisfy these requirements.

11. Information collected must be stored in a persistent manner.



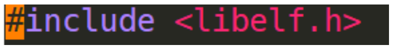
```
void saveFile(char *buffer, int size){
        FILE *outfile;

        outfile = fopen("elfData.bin","ab+");
        fwrite(buffer, sizeof(char), size, outfile);
        fflush(outfile);
        fclose(outfile);

}
```

Persistent storage

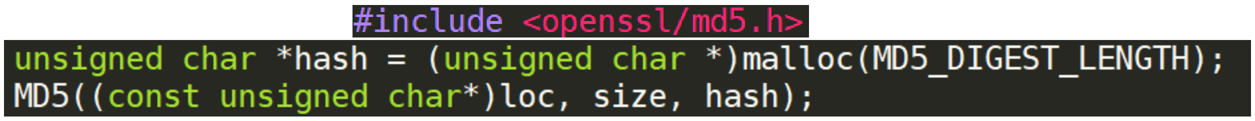
12. The software shall run on the Linux bash shell.
    This will be shown during use.
13. Software shall use libelf



```
#include <libelf.h>
```

Used libelf

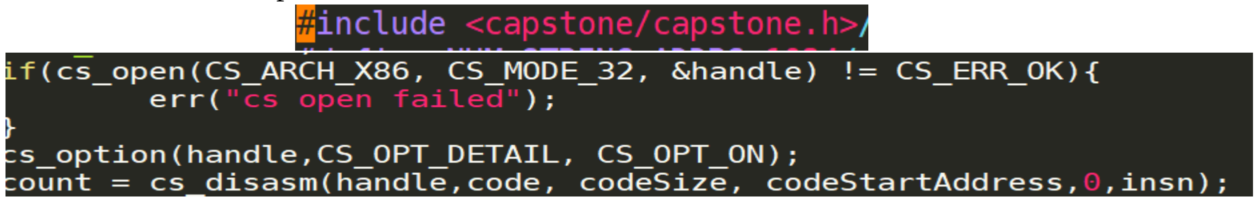
14. Software shall use openssl



```
#include <openssl/md5.h>
unsigned char *hash = (unsigned char *)malloc(MD5_DIGEST_LENGTH);
MD5((const unsigned char*)loc, size, hash);
```

Used openssl

15. Software shall use capstone



```
#include <capstone/capstone.h>
if(cs_open(CS_ARCH_X86, CS_MODE_32, &handle) != CS_ERR_OK){
        err("cs open failed");
}
cs_option(handle,CS_OPT_DETAIL, CS_OPT_ON);
count = cs_disasm(handle,code, codeSize, codeStartAddress,0,insn);
```

Used capstone

16. The software shall contain a vulnerability.
    The vulnerability included in binScan exists in authenticating users in the assembly function unpass.asm. The only way to be authenticated is to use the username/password combination root/toor. This should be found when reversing the binary.
17. Software shall contain one function written in assembly.
    The fuction unpass.asm satisfies this requirement.
18. All functions shall be written in C or x86 assembly.
    I don't think there's anything I can say that will convince you of this, you'll just have to look at source.
19. Software must be built and run on Linux for a 32 bit architecture with gcc.
    I built the software on the class VM's, the makefile contains gcc compiler.
20. The software must compile and run on the class VM's.
    I did all of my code writing, testing, and documenting using the class VM's.
21. Optional requirement: The software should resolve and store the strings passed to dlopen.

    Starts on line 173 of parseElf.c. Sorry, it's kind of ugly.