CSD4002:Ethical Hacking-Unit 4:Vulnerability Analysis

Dr. Ajit Kumar

November 2, 2019

What is reverse engineering?

At the highest level, it is simply taking a product apart to understand how it works.

- Understand the capabilities of the product's manufacturer
- Understand the functions of the product in order to create compatible components
- Determine whether vulnerabilities exist in a product
- Determine whether an application contains any undocumented functionality

Static / Passive Reverse Engineering

Static (also called passive) reverse engineering techniques in which you attempt to discover potential flaws and vulnerabilities simply by examining source or compiled code.

Ethical Reverse Engineering

Reverse engineering is often viewed as the craft of the cracker who uses her skills to remove copy protection from software or media. You are allowed to conduct such research as long as you have the permission of the owner of the subject system and you are acting in good faith to discover and secure potential vulnerabilities.

Vulnerability Analysis v/s Reverse Engineering

- One Should be interested in reverse engineering if want to extend their vulnerability assessment skills beyond the use of the pentester's standard bag of tricks.
- Nessus
- Pentesting tools can only report on **what they know**. They can't report on **undiscovered vulnerabilities**,
- Reverse engineering helps to find new and Unknown vulnerability
- Vulnerability researchers use a variety of reverse engineering techniques to find new vulnerabilities in existing software.

Reverse Engineering Considerations

- Failure to check for error conditions
- Poor understanding of function behaviors
- Poorly designed protocols
- Improper testing for boundary conditions

Uninitialized and Null pointers

Uninitialized pointers

Uninitialized pointers contain unknown data.

Null Pointer

Null pointers have been initialized to point to nothing so that they are in a known state.

Dereferencing

In C / C++ programs, attempting to access data (dereferencing) through either usually causes a program to crash or, at minimum, causes unpredictable behavior.

Group Activity

Using various Vulnerability database, find 10 different type of Vulnerabilities reported in 2019. Find few recent vulnerability reported regarding social media applications and Android OS.

Source Code Analysis

- A number of tools exist that attempt to automatically scan source code for known poor programming practices.
- automated tools tend to catch common cases and provide no guarantee that an application is secure.
- Source Code Auditing Tools (Lynis, ITS4, RATS (Rough Auditing Tool for Security), Flawfinder, and Splint (Secure Programming Lint), Microsoft's PREfast tool)
- RATS can Scan: Perl,PHP, Python, and C code.

Lynis and RATS

Example

- $lynis audit system
- $./rats -i -w 1 -d rats-c.xml example.c
- **rats-c.xml** :vulnerability database

Source Code Auditing:White, Black and Grey

- The goal of a white hat reviewing the output of a source code auditing tool should be to make the software more secure.
- The black hat is by definition interested in finding out how to exploit a program.
- It is useful for the gray hat to understand how to make use of the audit results to locate actual vulnerabilities and develop proof-of-concept code to demonstrate the seriousness of these vulnerabilities.

RATS:Installation and Use

Lynis: Installation and Use

- Kali: pre-installed
- Ubuntu: sudo apt-get install lynis
- Lynis performs more than 200 tests

Lynis commands

- $lynis show commands
- $lynis show settings
- $lynis update info
- $sudo lynis audit system
- $sudo lynis show details test-id
- $sudo nano /etc/lynis/custom.prf
- https://www.digitalocean.com/community/tutorials/how-to-perform-security-audits-with-lynis-on-ubuntu-1

Yasca: Installation and Use

- C/C++
- Java source and class files
- JSP source files
- PHP source files
- Perl
- Python

Manual Source Code Auditing

Sources of User-Supplied Data

- Command-line parameters argv manipulation
- Environment variables getenv()
- Input data files read(), fscanf(), getc(), fgetc(), fgets(), vfscanf()
- Keyboard input/stdin read(), scanf(), getchar(), gets()
- Network data read(), recv(), recvfrom()

Manual Auditing of Binary Code

Disassembler	The purpose of a disassembler is to generate assembly language from a compiled binary.
Decompiler	the purpose of a decompiler is to attempt to generate source code from a compiled binary.

Everything is file

Duplicating I/O Interfaces

In C/C++ programs, file descriptors 0, 1, and 2 correspond to the standard input (stdin), standard output (stdout), and standard error (stderr) devices. The dup2() function can be used to make stdin become a copy of any other file descriptor, including network sockets. Once this has been done, a program no longer accepts keyboard input; instead, input is taken directly from the network socket.

Manual Auditing of Binary Code

- Disassembler
- Decompiler

Decompilation

Lossy operation Information is lost in the process of generating machine language.

One-to-many There are many valid translations of a single line of source code to equivalent machine language statements.

Optimization to optimize a program for **speed** will generate vastly different code from what it will generate if asked to optimize that same program for **size**

Platform C and C++ are compiled to platform-specific machine language and linked to operating system–specific libraries.

Decompilation Tools

- JReversePro and Jad (Java Decompiler).
- `http://www.javadecompilers.com/`
- IDA Pro
- Hex-rays

Disassemblers

- compiled program to execute, it must communicate some information to its host operating system
- PE and ELF

IDA Pro

Developer	Ilfak Guilfanov
Use	Decompiler
Structure	IDA Pro is actually a database application.
Working	When a binary is loaded for analysis, IDA Pro loads each byte of the binary into a database and associates various flags with each byte. These flags can indicate whether a byte represents code, data, or more specific information such as the first byte of a multibyte instruction.
.IDB	Disassemblies are saved as IDB files separate from the original binary

Dynamic Linking to analyze dynamically linked binaries, IDA Pro makes use of embedded symbol table information to recognize references to external functions. Within IDA Pro's disassembly listing, the use of standard library names helps make the listing far more readable.

Static Linking For statically linked C/C++ binaries, IDA Pro uses a technique termed Fast Library Identification and Recognition Technology (FLIRT), which attempts to recognize whether a given machine language function is known to be a standard library function, by matching disassembled code against signatures of standard library functions used by common compilers

Powerful Features

- Code graphing capabilities to chart function relationships
- Flowcharting capabilities to chart function flow
- A strings window to display sequences of ASCII or Unicode characters contained in the binary file
- A large database of common data structure layouts and function prototypes
- A powerful plug-in architecture that allows extensions to IDA Pro's capabilities to be easily incorporated
- A scripting engine for automating many analysis tasks
- Several integrated debuggers

IDA Pro Freeware

Installation and Basic Interface

IDA pro has multiple version, freeware, paid and evaluation for Windows, Linux and Mac

- Download installer from Hex-rays official site as per your OS.
- For Linux: make binary executable by $chmod +x idapro_XX.run
- Then run the installer
- Move to the folder example, /home/ajit/idaprofreeware
- $ ./ida64

Static Analysis Challenge

- Binaries that have been stripped of some or all of their symbol information
- Binaries that have been linked with static libraries
- Binaries that make use of complex, user-defined data structures
- Compiled C++ programs that make use of polymorphism
- Binaries that have been obfuscated in some manner to hinder analysis
- Binaries that use instruction sets with which IDA Pro is not familiar
- Binaries that use file formats with which IDA Pro is not familiar

Fuzzing

Blackbox	Black box testing works because you can apply some **external stimulus** to a program and observe how the program **reacts** to that stimulus.
Monitoring	Monitoring tools give you the capability to observe the program's reactions.
Fuzzing	Fuzzing tools are designed for the **rapid generation** of input cases designed to induce errors in a program.
Challenge	The real challenge of fuzzer development is building them in such a way that they generate interesting input in an intelligent, efficient manner.

Known and Unknown Protocols

- Known Protocol (HTTP, FTP etc. RFC)
- Unknown Protocol (proprietary protocol non-RFC)

SPIKE

- SPIKE is a fuzzer creation toolkit/API
- By: Dave Aitel of Immunity, Inc
- SPIKE provides a library of C functions for use by fuzzer developers
- SPIKE is designed to assist in the creation of network-oriented fuzzers and supports sending data via TCP or UDP.
- SPIKE provides several example fuzzers for protocols ranging from HTTP to Microsoft Remote Procedure Call (MSRPC)

Working of SPIKE

- The SPIKE API centers on the notion of a “spike” data structure.
- Various API calls are used to push data into a spike and ultimately send the spike to the application being fuzzed.
- Spikes can contain static data, dynamic fuzzing variables, dynamic length values, and grouping structures called blocks.
- A SPIKE “block” is used to mark the beginning and end of data whose length should be computed. Blocks and their associated length fields are created with name tags.

Spike Creation Primitives

- struct spike *new_spike() Allocate a new spike data structure.
- int spike_free(struct spike *old_spike) Release the indicated spike.
- int set_spike(struct spike *new_spike) Make newspike the current spike.

All future calls to data manipulation functions will apply to this spike.

SPIKE Static Content Primitives

- s_string(char *instring) Insert a static string into a spike.
- s_binary(char *instring) Parse the provided string as hexadecimal digits and add the corresponding bytes into the spike.
- s_bigword(unsigned int aword) Insert a big-endian word into the spike. Inserts 4 bytes of binary data into the spike.
- s_xdr_string(unsigned char *astring) Insert the 4-byte length of astring followed by the characters of astring into the spike. This function generates the XDR representation of astring

XDR is the External Data Representation standard, which describes a standard way in which to encode various types of data such as integers, floating-point numbers, and strings.

SPIKE Block Handling Primitives

- int_block_start(char *blockname) Start a named block. No new content is added to the spike. All content added subsequently up to the matching block_end call is considered part of the named block and contributes to the block's length.
- int s_block_end(char *blockname) End the named block. No new content is added to the spike. This marks the end of the named block for length computation purposes.

SPIKE Fuzzing Variable Declaration

A fuzzing variable is a string that SPIKE will manipulate in some way between successive transmissions of a spike.

- void s_string_variable(unsigned char *variable) Insert an ASCII string that SPIKE will change each time a new spike is sent.

When a spike contains more than one fuzzing variable, an iteration process is usually used to modify each variable in succession until every possible combination of the variables has been generated and sent.

SPIKE Script Parsing

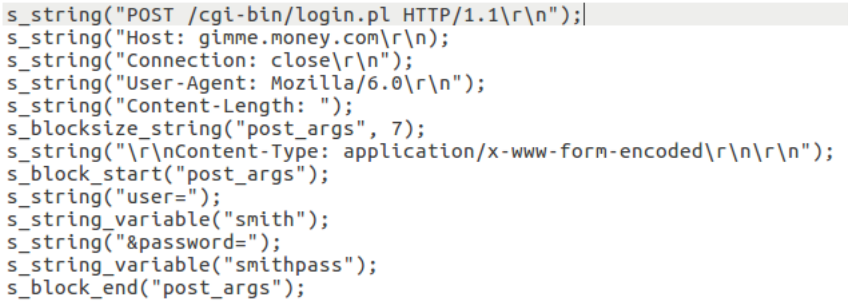
SPIKE statements can be placed in a text file and executed from within another SPIKE-based program. All of the work for executing scripts is accomplished by a single function.

- int s_parse(char *filename) Parse and execute the named file as a SPIKE script.

Example

```
POST /cgi-bin/login.pl HTTP/1.1
Host: gimme.money.com
Connection: close
User-Agent: Mozilla/6.0
Content-Length: 29
Content-Type: application/x-www-form-encoded
user=smith&password=smithpass
```

A Simple SPIKE Script:.spk



```
s_string("POST /cgi-bin/login.pl HTTP/1.1\r\n");
s_string("Host: gimme.money.com\r\n);
s_string("Connection: close\r\n");
s_string("User-Agent: Mozilla/6.0\r\n");
s_string("Content-Length: ");
s_blocksize_string("post_args", 7);
s_string("\r\nContent-Type: application/x-www-form-encoded\r\n\r\n");
s_block_start("post_args");
s_string("user=");
s_string_variable("smith");
s_string("&password=");
s_string_variable("smithpass");
s_block_end("post_args");
```

generic_send_tcp

It takes care of the details of initializing a spike,
parsing a script into the spike,
and iterating through all fuzzing variables in the spike.

1.Host	the host to be fuzzed,
2.Port	the port to be fuzzed,
3.SPIKE Script	the filename of the spike script
4.Do Skip Variable	information on whether any fuzzing variables should be skipped,
5.State Skip Variable	whether any states of each fuzzing variable should be skipped.

```
$./generic_send_tcp gimme.money.com 80 demo.spk 0 0
```

Other Fuzzer

- SPIKE Proxy: Web Application, Python
- Sharefuzz:fuzz set user ID (SUID) root binaries.
- **mangleme**:specifically target browser-based client-side vulnerabilities.(outdated)
- HTMLer: https://securiteam.com/tools/6Z00N1PBFK/ (Python Port of mangleme)
- jsfunfuzz: Javascript (Mozilla)
- css-grammar-fuzzer (CSS fuzzer)

Intelligent fuzzing

Instead of blindly throwing everything, techniques have been developed to analyze how a server works and to customize a fuzzer to get past the filters and reach deeper inside the server to discover even more vulnerabilities.

Intelligent fuzzing Steps

- need to conduct a protocol analysis of the target.
- need a way to fuzz that protocol and get feedback from the target as to how you are doing.

Sulley and Boo From Monsters Inc.

Sulley/Boofuzz

- The Sulley fuzzing framework automates this process and allows you to intelligently sling packets across the network.
- This tool is truly revolutionary in that it provides not only a great fuzzer and debugger, but also the infrastructure to manage a fuzzing session and conduct postmortem analysis.
- https://github.com/OpenRCE/sulley
- https://github.com/jtpereyda/boofuzz

Protocol Analysis

www.ietf.org/rfc/rfc1179.txt