# Fuzzing

sost@fabletech.com

Fabletech.com

6-6-2011

# Fuzzing

- Fuzzing was first described by Prof. Barton Miller in 1989.
- Fuzzing is in its basic form very simple. Understand the programs valid input and randomize it.

# Fuzzing methods

- Random
- Manual testing (Bad)
- Pregenerated testcases
- Mutation or Bruteforce Fuzzing
- Automatic protocol testing

# Fuzzing types

- Local fuzzing
  - Commandline
  - Enviroment variable
  - File format
- Remote fuzzing
  - Web
  - Network protocol
  - Web browser
- In memory fuzzing

# Fuzzing process

- Identify target
- Identify inputs
- Generate fuzzed data
- Execute fuzzed data
- Monitor exception
- Determine exploitability

# Tools

- Peach: Python based and the fuzzing is setup in xml.
- Spike: Blockbased fuzzer written in c
- Powerfuzzer
- OWASP JBroFuzz web application fuzzer
- OWASP WSFuzzer webservice fuzzer
- Sulley: python based block fuzzer
- Quickcheck

# Setup

download, unpack, and if python 2.5 is installed you are ready to go.

# Why is sulley so powerfull

Sulley is build up as modules, this means that even though Sulley is ideal for network protocol fuzzing it can be used in other fuzzing assigments.

```
from sulley import primitives
st = primitives.string('test')
for i in range(10):
    if st.mutate():
        print st.value
```

# Sulley primitives

- Strings: `s_string('test')`
- Delimiters: `s_delim(';')`
- Static: `s_static('dont touch')`
- Random:
  `s_random("text",min_length=5,max_length=10,num_mutatio`
- Binary: `s_binary('\\x41\\x42\\x43')`
- Integers: `s_int()`, `s_dword()`, `s_byte()`

# Fuzz library Extentions

Sulley has en internal fuzz library. If you for some reason do not wish to use this you can define your own. Using the fielnames .fuzz_strings or .fuzz_ints.

# Blocks

- Blocks is a smart way of defining entities
- Blocks have a s_block_start() and an s_block_end()
- Each block have a name associated to it
- Blocks is not very usefull on its own. But combined with groups and repeaters it becomes a powerfull tool.

# Blocks example

```
s_initialize("HTTP BASIC")
s_group("verbs", values=["GET", "HEAD", "POST", "TRACE"])
if s_block_start("body", group="verbs"):
    s_delim(" ")
    s_delim("/")
    s_string("index.html")
    s_delim(" ")
    s_string("HTTP")
    s_delim("/")
    s_string("1")
    s_delim(".")
    s_string("1")
    s_static("\r\n\r\n")
s_block_end("body")
```

# Sessions

Until now we have only been able to fuzz request/response protocols. That is no states have en dealt with. Sessions enalbles us to do so. By using s_initializers() and session.commect() we can create states.

# sessions

```
sess = sessions.session()
sess.connect(s_get('test0')
sess.connect(s_get('test0',s_get('test1.1'))
sess.connect(s_get('test0's_get('test1.2'))
sess.connect(s_get('test1.1',s_get('test2'))
sess.connect(s_get('test1.2'),s_get('test2'))
```

# Agents

- Network monitor
- Process monitor
- Virtual machine monitor

# Doing fuzzing well

- Attach a debugger, in order to do code coverage
- When ever a crash occurs dump as much info af possible
- cpu for the win
- use virtualization
- use pin or dynamorio for code coverage
- !exploitable

# fuzzing pitfalls

- Race conditions
- ”intended” use
- Some fuzzers like sulley are only written for certain oprating systems

# Fuzzing

Questions?