Fuzzing 101

NYU/Poly.edu

October 23, 2008

Mike Zusman

Protocol Fuzzing

- Find as much data as you can about the target application
  - Google is your friend
  - Maybe someone has fuzzed it
  - Maybe it uses some standard protocol

Protocol Fuzzing

- What is the transport layer?
  - TCP or UDP?
    - Effects anomaly detection

Protocol Fuzzing

- What type of protocol?
  - SIMPLE
    - Text Based
  - COMPLEX
    - Binary

Protocol Fuzzing

- What type of protocol?
  - SIMPLE
    - Text Based
  - COMPLEX
    - Binary

Protocol Fuzzing

- Do we need to authenticate?
  - What authentication protocol?
- Scoping your assessment
  - You may only care about pre-auth

Protocol Fuzzing

- Reversing the Protocol
  - Generate Traffic and Sniff
  - Use wireshark (check for plug-ins!)
  - It never hurts to ask Google

Protocol Fuzzing

- Reversing the Protocol
  - Establish syntax (authenticate first, then command1, followed by command2)
  - Establish a list of commands
  - Establish a list of arguments

Protocol Fuzzing

- Reversing the Protocol
  - Build Command Prototypes

    <argument> : required

    [argument] : optional

    {CONSTANT1|CONSTANT2 …}: Required constant argument

- Example:
  - PASS {SYS | USER <Username>} <Password>

Protocol Fuzzing

- Once you understand how to communicate with a service, you can send packets to it.
- Simple Protocols
  - Use telnet, nc.exe, openssl
- Complex Protocols
  - Write Code

Protocol Fuzzing

- Now that you can communicate with the protocol…
- Fuzzing Strategy
  - How would you fuzz it?
- What can you fuzz in this prototype?
  - PASS {SYS | USER <Username>} <Password>

Protocol Fuzzing

- Fuzzing is repetitive
  - Open/Close connections to hosts
  - Build a UDP packet
  - Write data to a socket
  - Read Data from a socket
  - Loop through a sequence
  - Fuzz each parameter
  - etc

Protocol Fuzzing

- If you try to write a network protocol fuzzer, you will eventually end up re-inventing the wheel
- SPIKE is a fuzzing framework/API
  - Written in C by Dave Aitel
- It takes care of the busy work

SPIKE

- If you try to write a network protocol fuzzer, you will eventually end up re-inventing the wheel
- SPIKE is a fuzzing framework/API
  - Written by Dave Aitel
- It takes care of the busy work

SPIKE

- Simple Text Based Protocol Fuzzing
  - line_send_tcp.c
    - Accepts a “script” of SPIKE commands
    - Example:

```
s_string_variable("PASS");
s_string(" ");
s_string_variable("USER");
s_string(" ");
s_string_variable("devel_user");
s_string(" ");
s_string_variable("secretpassword");
s_string("\r\n");
```

SPIKE

- Simple Text Based Protocol Fuzzing
  - line_send_tcp.c
    - ./line_send_tcp <IP> <PORT> script.spk 00

SPIKE

- SPIKE’s real value
  - Complex Protocols have length fields and data fields
  - Tracking length fields while fuzzing data is complicated
  - SPIKE does this for you
  - Block Based Protocol Representation

SPIKE

- What is a SPIKE?
  - “A SPIKE is a simple list of structures which contain block size information and a queue of bytes.”

```
s_block_size_binary_bigendian_word(“somepacketdata”);
s_block_start(“somepacketdata”)
s_binary(“01020304”);
s_block_end(“somepacketdata”);
```

SPIKE

```
s_block_size_binary_bigendian_word("somepacketdata");
s_block_start("somepacketdata")
s_binary("01020304");
s_block_end("somepacketdata");
```

- Push 4 NULLs onto BYTE queue (size place holder)
- Then a new BLOCK listener is allocated named "somepacketdata"

SPIKE

s_block_size_binary_bigendian_word(“somepacketdata”);
s_block_start(“somepacketdata”)
s_binary(“01020304”);
s_block_end(“somepacketdata”);

- Script starts searching the block listeners for one named “somepacketdata”
- Block “start” pointers are updated to reflect the blocks position in the queue

SPIKE

```
s_block_size_binary_bigendian_word("somepacketdata");
s_block_start("somepacketdata")
s_binary("01020304");
s_block_end("somepacketdata");
```

- 4 bytes of data are pushed onto the queue

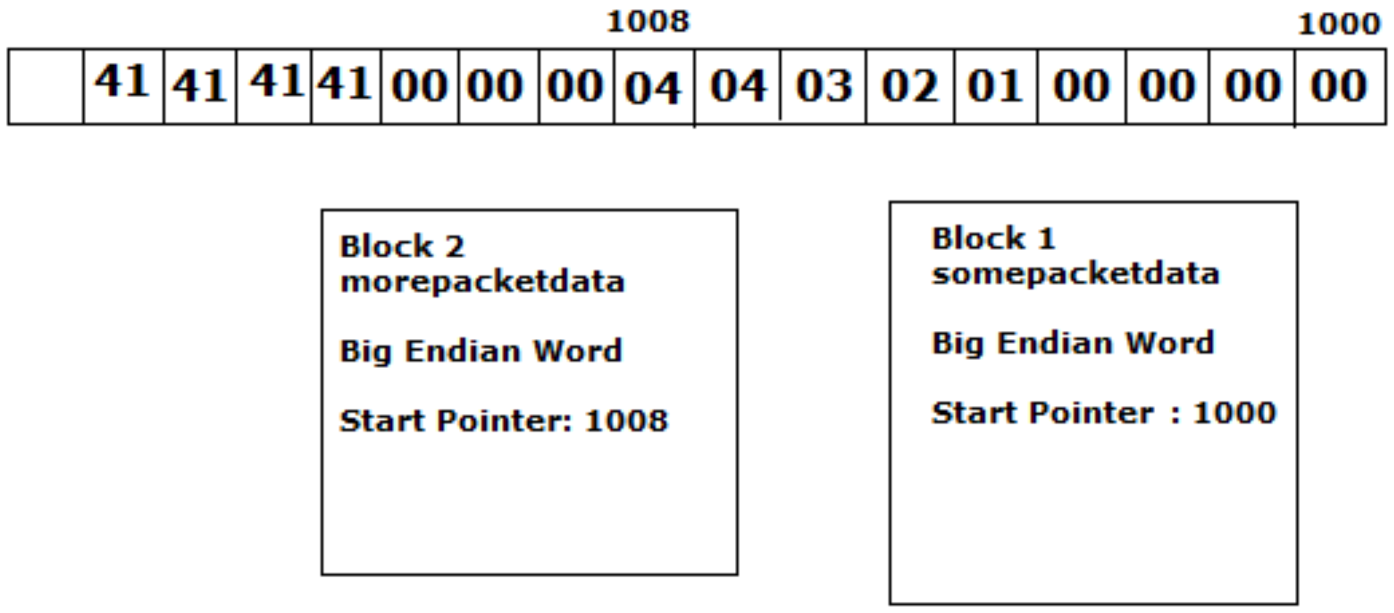
SPIKE

```
s_block_size_binary_bigendian_word(“somepacketdata”);
s_block_start(“somepacketdata”)
s_binary(“01020304”);
s_block_end(“somepacketdata”);
```

- The block is ended, and the sizes are finalized
- The original 4 null bytes are updated with the appropriate size value

SPIKE

s_block_size_binary_bigendian_word("somepacketdata");
s_block_start("somepacketdata")
s_binary("01020304");
s_block_end("somepacketdata");



SPIKE

- Given Prototype

```
Data (length 100 byte)
{ Element1 (length 75 bytes)
        {
              B x 50
              SubElement1(length 25 bytes)
                    {A x 25}
        }
}
```

Writing SPIKE

- Walk Through the Code
  - Citrix.c

Writing SPIKE

- Walk Through the Code
  - line_send_tcp.c

Writing SPIKE

- That’s it!