An Introduction to SPIKE, the Fuzzer Creation Kit

Dave Aitel

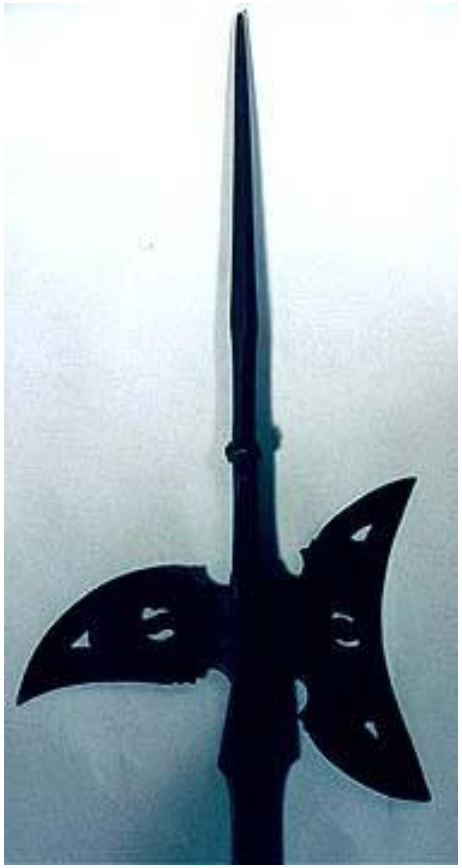
http://www.immunitysec.com

Agenda

- Demo and Vulnerability
- Theory
  - Goals
  - Using the SPIKE API
  - Useful samples included with SPIKE
- Questions throughout and at end

<GOBBLES> i used to laugh at fuzzers, but then you changed my whole outlook on life!

Demo of SPIKE in Action



Theory

- SPIKE is a GPL'd API and set of tools that allows you to quickly create network protocol stress testers
- Most protocols are built around extremely similar data formatting primitives
- Many of these are already supported in SPIKE
- Others soon will be. :>

Fes: “I'm always surprised at how effective fuzzers actually are!

The Goals of SPIKE

- Find new vulnerabilities by
  - Making it easy to quickly reproduce a complex binary protocol
  - Develop a base of knowledge within SPIKE about different kinds of bugclasses affecting similar protocols
  - Test old vulnerabilities on new programs
  - Make it easy to manually mess with protocols

How the SPIKE API works

- Unique SPIKE data structure supports lengths and blocks
  - s_block_start(), s_block_end(), s_blocksize_halfword_bigendian();
- SPIKE utility routines make dealing with binary data, network code, and common marshalling routines easy
  - s_xdr_string()
- SPIKE fuzzing framework automates iterating through all potential problem spots
  - s_string(“Host: “); s_string_variable(“localhost”);

The SPIKE Datastructure

- A SPIKE is a kind of First In First Out Queue or “Buffer Class”
- A SPIKE can automatically fill in “length fields”
  - s_size_string(“post”,5);
  - s_block_start(“Post”);
  - s_string_variable(“user=bob”);
  - s_block_end(“post”);

Length Fields

- Length fields come in many varieties
  - Word/halfword/string
  - Big endian, little endian
- More than one length field can “listen” for a particular block to be closed
- Blocks can be nested or intertwined

A few basic calls

- The main call is s_push(buffer,size) underneath everything
  - Currently, there is no s_pop();
- String calls:
  - s_string(“hi”);
  - s_string_variable(“hi”);
- s_binary(“\\x41 4141 0x41 41 00”);
  - Can take in all sorts of cut and pasted hexadecimal data without choking
  - Handles white space cleanly

Setting up/destroying a SPIKE

- Global variables you have to deal with:
  - set_current_spike(*struct spike);
  - spike_clear();
- Malloc fun
  - spike_new();
  - spike_free();

Network SPIKE calls

- Basic TCP connectivity
  - spike_tcp_connect(host,port);
  - spike_send();
  - spike_close_tcp();
- Basic UDP Connectivity
  - spike_udp_connect(host,port);
  - spike_send();

Fuzzing Framework SPIKE calls

- s_string_variable(“”);
- s_string_repeat(“A”,5000);
  - Equivalent to s_push("perl -e 'print “A” x 5000’”)
- While loop support
  - s_incrementfuzzstring();
  - s_incrementfuzzvariable();

Advantages to using SPIKE's fuzzing framework over a perl script

- Size values will automatically get updated
- Can handle binary data cleanly via s_binary();
- Already knows about many different types of interesting strings to use for fuzzstrings
- Integrates cleanly with libntlm or other GPL'd libraries in C for doing encryption or other things for which you don't already have perl modules

The Process of Using SPIKE on an unknown protocol

- Use Ethereal to cut and paste the packets into s_binary();
- Replace as much of the protocol as possible with deeper level spike calls
  - s_xdr_string(); s_word(); etc
- Find length fields and mark them out with size calls and s_block_start(), s_block_end();
- Make sure protocol still works :>
- Integrate with fuzzing framework (2 while() loops) and let the SPIKE fuzzer do the boring work
- Manually mess with the packets to see if you can cause any aberrant behaviour (attach ollydebug first)
- Write up the exploits

The SPIKE scripting language

- ...is C.
- s_parse(“filename.spk”);
  - Loads the file line by line and does limited C parsing on it
  - Uses dlopen() and dlsym() and some demarshalling to call any functions found within
    - printf(“Hi %s %s\n”,”dave”,”what's up?”);
    - s_clear();
    - s_binary(“41 42 43 44 45”);
- Typically a “generic” framework is built, then SPIKE script is used to quickly play with the protocol

Current Demo SPIKEs

- Web Focused
- MSRPC protocol support
- Miscellaneous other demos

SPIKE Programs for non Web Apps

- msrpcfuzz
- Citrixfuzz
- Quake,halflife (UDP demos)

Quickstart: msrpcfuzz

- First use DCEDUMP (basically rpcinfo against Windows)
- Then chose a program and port to fuzz
  - Sends valid, but random data structures to that program
- Watch it crash!

SPIKE Programs for Web Apps

- ntlm2/ntlm_brute
- webmitm
- makewebfuzz.pl
- webfuzz.c
- closed_source_web_server_fuzzer
- generic_web_server_fuzz

ntlm_brute and ntlm2

- Tries to do a dictionary attack on NTLM authenticating web servers
- Somewhat slow but easy to parallelize
- Very simple to use with provided do_ntlm_brute.sh
- Ntlm2 useful for doing “webfuzz” activity on a page that requires NTLM authentication

Webmitm (SPIKE version, not dsniff Version)

- Transparent proxy (originally from dsniff)
- Used to generate http_request files
- Can do SSL
- Rewrites Host: headers
- Cool with “Connection: keep-alive”

Makewebfuzz.pl

- Creates webfuzz.c files from http_request files
- Superceeded by SPIKE Console wizardry and generic .spk scripts, but still useful

Webfuzz

- Sends the valid request, but incrementally goes through each variable in the request and checks it for common vulnerabilities

A Standard Request

GET /login.asp?Username=**Dave**&Password=**Justine**

Host: bobsbagoffish.com

Content-Length: 16

Server=**whitebait**

A webfuzz request

GET /login.asp?Username=**../../etc/hosts%00**&Password=**Justine**

Host: bobsbagoffish.com

Content-Length: 16

Server=whitebait

Closed_source_webserver_fuzz

- Uses same set of fuzz strings to locate common web server overflows, format string bugs, etc
- Also useful for rigorous manual testing of one CGI

Automating the process of finding SQL injection bugs

- odbcwebfuzz.sh
  - Make a directory of captured http_requests using webmitm
  - Compile each of these into a webfuzz using makewebfuzz.pl
  - Run each of these against the server
  - Grep through results for interesting errors (such as ODBC)
  - You just saved 20K! :>

When Automation Fails

- This is an exponential problem!
  - Unlike commercial alternatives to SPIKE every part of SPIKE is open
  - SPIKE can be extended with any other GPL code
  - I accept patches

Examples where automation fails

- User Registration that requires a sequence of pages to be hit
  - use SPIKE to automate hitting the first two and then fuzz every variable on a third page
- More complex web applications that use characters other than '&' to split up variables
- Page sequences that require some parsed input from a previous page to be included in a submitted request

The SPIKE Console

- wxPython
  - cross platform
  - pretty
- Wizards enable quick utilization of SPIKE's capabilities
- Currently beta, but useful
- Under heavy development

The Future of SPIKE

- SPIKE Console Improvements
- Additional SPIKE protocol demos and updates

Conclusion

- For most standard web applications SPIKE can quickly help you find SQL injection, overflow, and format string bugs
- SPIKE can be quickly customized for your specific needs
- Use SPIKE to reverse engineer and fuzz binary protocols in less time than you otherwise could
- Download for FREE today!
  - http://www.immunitysec.com/spike.html
- Comments to dave@immunitysec.com