

# Automatic Network Protocol Analysis

Gilbert Wondracek, **Paolo Milani Comparetti**, Christopher Kruegel  
and Engin Kirda

pmilani@ sssup.it  
gilbert@ seclab.tuwien.ac.at  
chris@ cs.ucsb.edu  
engin.kirda@ eurecom.fr

---

# Reverse Engineering Network Protocols

---

*Secure Systems Lab  
Technical University Vienna*

- Find out what application-layer “language” is spoken by a server implementation
  - Message formats
  - Protocol state machine
- Slow manual process
- Do it automatically!

# Reverse Engineering Network Protocols: Security Applications

*Secure Systems Lab  
Technical University Vienna*

---

- Black-box fuzzing
- Deep packet inspection
- Intrusion detection
- Reveal differences in server implementations
  - server fingerprinting
  - testing/auditing

# Reverse Engineering Network Protocols: Sources of Information

---

*Secure Systems Lab  
Technical University Vienna*

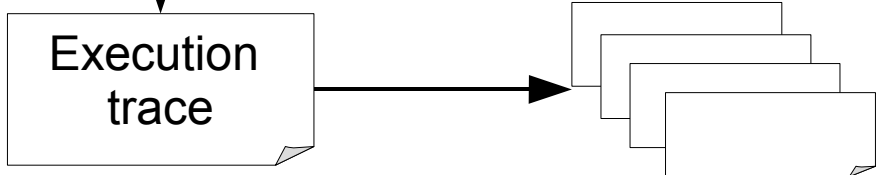
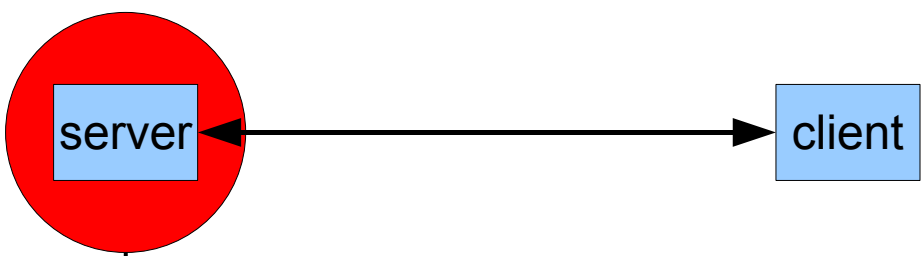
- **Network traces**
  - limited information (no semantics)
- **Server binaries**
  - static analysis
  - dynamic analysis

# Our approach

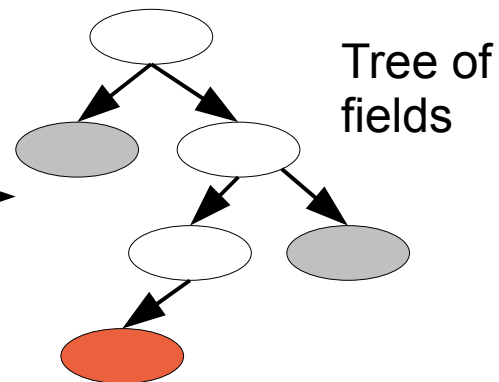
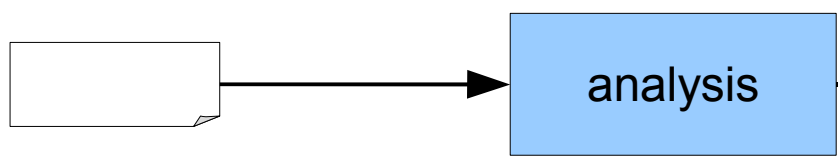
*Secure Systems Lab  
Technical University Vienna*

- Mostly dynamic analysis (+ static analysis)
- Use dynamic taint analysis to observe the data flow
- Observe how the program processes (parses) input messages
- Analyze individual messages
- Generalize to a message format for messages of a given type (i.e. HTTP get, NFS lookup..)
- Classification of messages into types is currently done manually

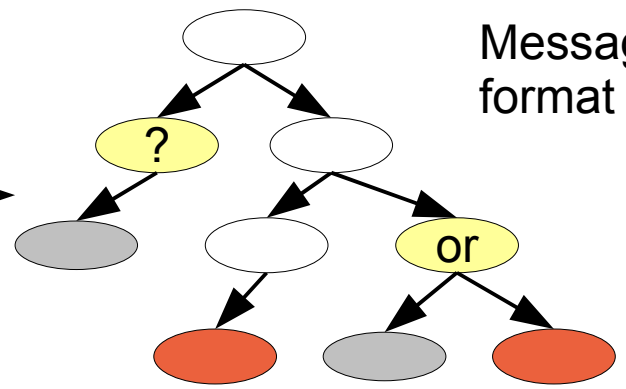
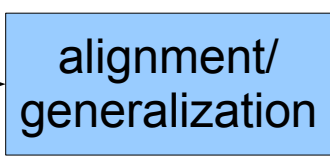
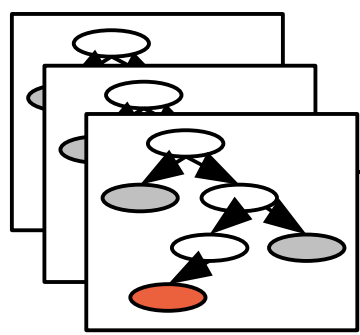
Dynamic  
taint analysis  
environment



Execution traces  
for individual messages



Tree of  
fields



Message  
format

# Dynamic Taint Analysis

Secure Systems Lab  
Technical University Vienna

- Run unmodified binary in a monitored environment (based on qemu, valgrind, ptrace..)
- Assign a unique label to each byte of network input
- Propagate the labels in shadow memory
  - for each instruction, assign labels of input to output destinations
  - also track address dependencies (example: lookup table-based *toupper()* function)

## Label Input:

G	E	T		/		H	T	T	P	/	1	.	0	\r	\n	\r	\n
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17

## Propagate Labels:

EAX



c	G	E
	0	1



BL

G
0

```
push %esi
push %ebx
mov (%eax),%bl
sub $0x1,%ecx
```

## Tainted data affects program flow:

Is (something derived from) byte 0 equal to '\n'?

```
cmp $0x0a,%bl
je 93
```

# Message Format Analysis

*Secure Systems Lab  
Technical University Vienna*

---

- **Structure-forming semantics**
  - enough information to parse a message out of a network data flow
  - variation between messages
- **Additional semantics**
  - keywords, file names, session ids,...

# Structure-Forming Semantics

*Secure Systems Lab  
Technical University Vienna*

---

- Length fields
  - and corresponding target fields, padding
- Delimiter fields
  - and corresponding scope fields
- Hierarchical structure

# Detecting Length Fields (1/2)

Secure Systems Lab  
Technical University Vienna

---

- Length fields are used to control a loop over input data
- Leverage static analysis to detect loops
- Look for loops where an exit condition tests **the same taint labels** on every iteration
- Need at least 2 iterations

# Detecting Length Fields (2/2)

*Secure Systems Lab  
Technical University Vienna*

- The tricky part is detecting the target field!
- Look at labels touched inside length loop
- Remove labels touched in all iterations
- May need to merge multiple loops (example: memcpy uses 4-byte mov instructions, but may need to move 1-3 bytes individually)
- Some bytes may be unused

# Detecting Delimiters

- Delimiter is one or more bytes that separate a field or message
  - Observation: all bytes in the scope of the delimiter are compared against a part of the delimiter
- Delimiter field detection
  - Create a list of taint labels used for comparisons for each byte value, merge consecutive labels into intervals
- Intervals indicate delimiter scope,
  - nesting gives us a hierarchical structure
  - recursive analysis to “break up” message

0 4 8 12 16 20 24

GET /index.html HTTP/1.1\r\n

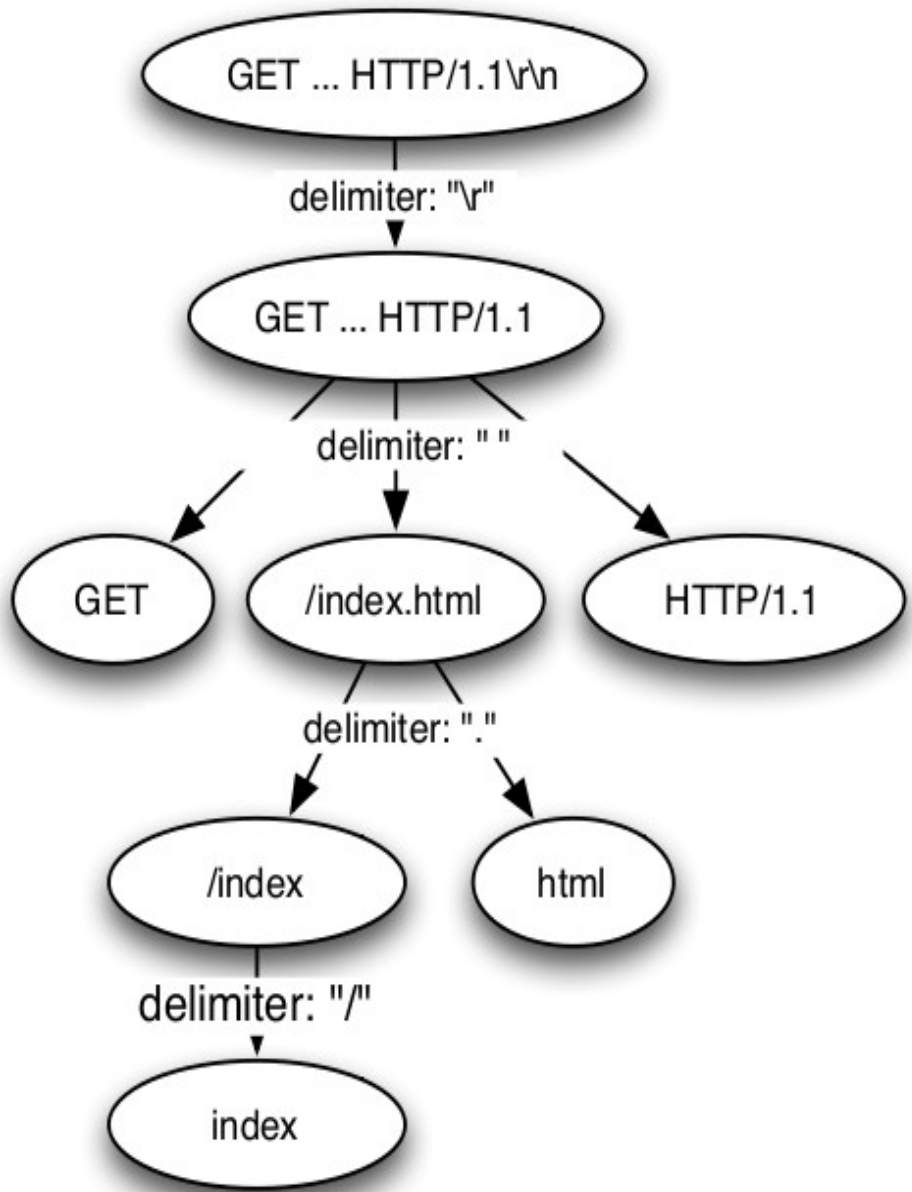
scan for "\r"

scan for " "

scan for "."

scan for "/"

	Initial Intervals
"\r"	[0,25]
" "	[0,23]
"."	[4,15]
"/"	[4,9]



# Additional Semantics

*Secure Systems Lab  
Technical University Vienna*

- Protocol keywords
- File names
- Echoed fields (session id, cookie,..)
- Pointers (to somewhere else in packet)
- Unused fields

# Detecting Keywords

- A keyword is a sequence of (1 or 2 byte) characters which is tested against a constant value
  - adjacent characters being successfully compared to non tainted values are merged into a string
  - take delimiters into account
- Ideally, we would want to check it is being tested against values which are hard coded in binary
  - trace taint from entire binary
- Currently, we just check the string (of at least 3 bytes) is present in the binary

# Generalization (1/3)

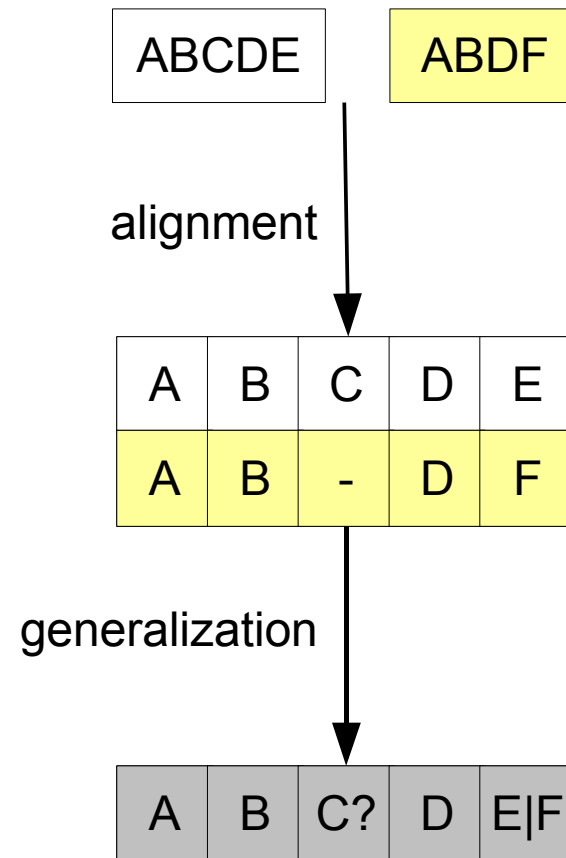
---

*Secure Systems Lab  
Technical University Vienna*

- Message alignment
- Based on Needlman-Wunsch
- Extended to a hierarchy of fields

# Generalization (2/3)

- Needleman-Wunsch
- Dynamic programming algorithm for string alignment
- Computes alignment which minimizes edit distances
- Also provides edit path between the strings
- Scoring function (for match, mismatch, gap)



# Generalization (3/3)

Secure Systems Lab  
Technical University Vienna

---

- Hierarchical Needleman-Wunsch
- Operate on a tree of fields, not on a string of bytes
- To align two inner nodes (complex fields) recursively call NW on the sequence of child nodes
- To align two leaf nodes, take into account field semantics
  - a length field only matches another length field
  - a keyword only matches same exact keyword
  - ...
- Simple scoring function: +1 for match, -1 for mismatch or gap

# Generalization: More Semantics

Secure Systems Lab  
Technical University Vienna

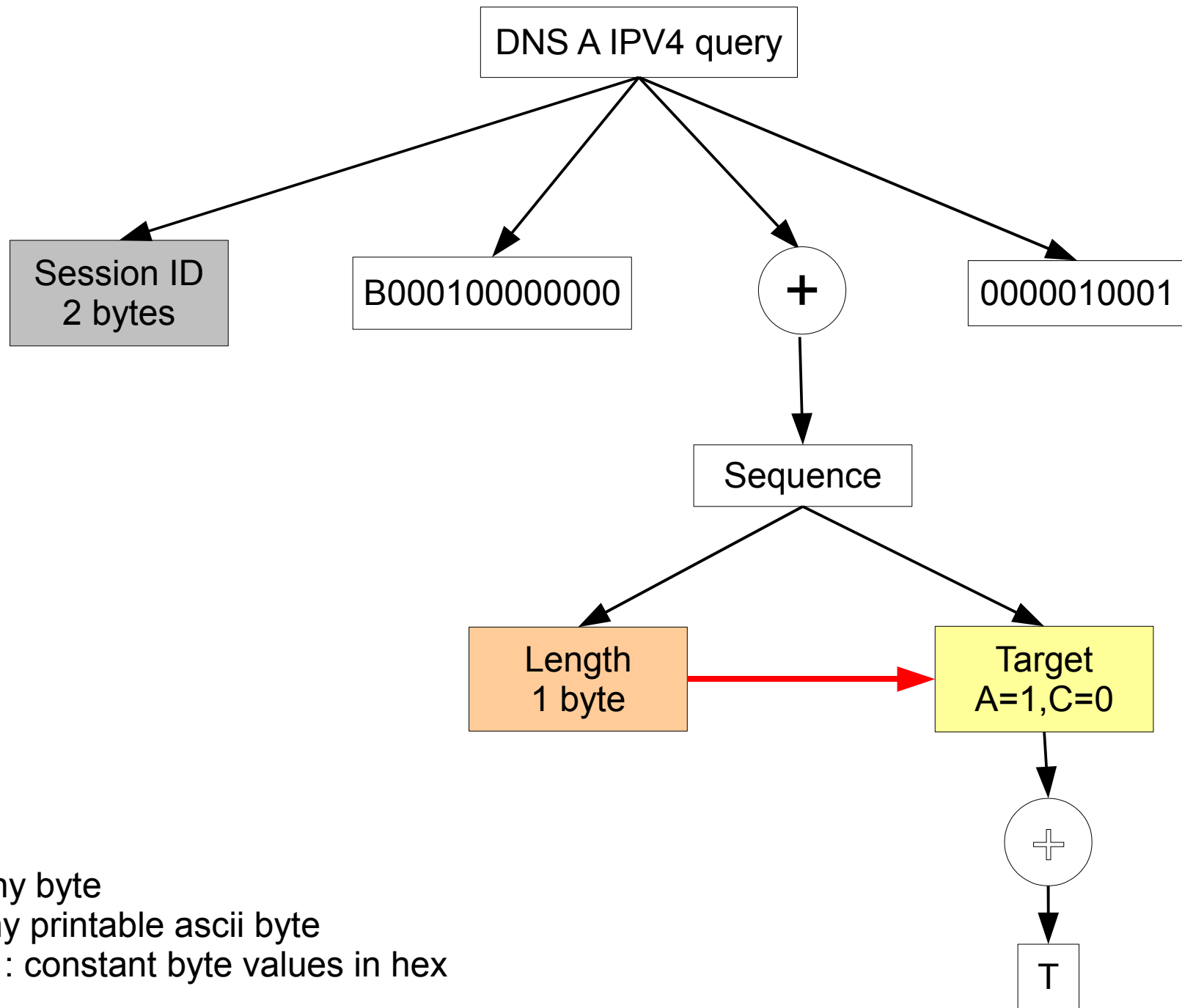
- Sets of keywords (i.e. *keep-alive* OR *close..*)
- Length field semantics
  - encoding: endianness
  - compute target field length  $T$  from length  $L$ :  $T=A*L+C$
- Pointer field semantics
  - encoding: endianness
  - offset: relative or absolute
  - offset value is  $A*L+C$
- Repetitions
  - generalize  $a?a?$  to  $a^*$

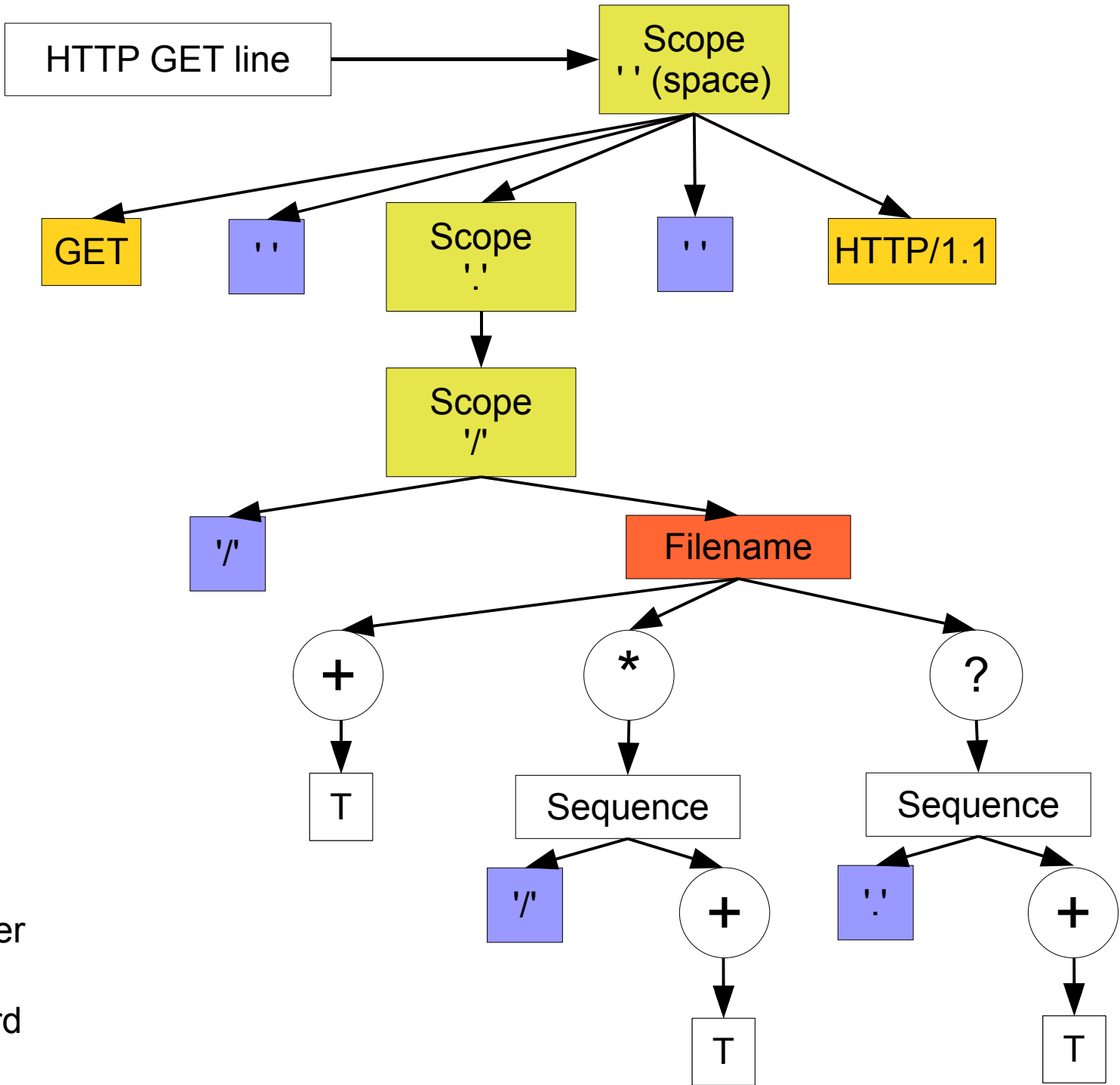
# Evaluation

*Secure Systems Lab  
Technical University Vienna*

---

- 7 servers (apache,lighttpd,iacd,sendmail,bind,nfsd,samba)
- 6 protocols (http, irc, smtp, dns, nfs, smb)
- 14 message types (
  - http get
  - irc nick, user
  - smtp mail, helo, quit,
  - dns IPv4 A query
  - rpc/nfs lookup, getattr, create, write
  - smb/cifs negotiate protocol request, session setup andX request, tree connect andX request





Delimiter  
 Keyword

# Parsing

*Secure Systems Lab  
Technical University Vienna*

---

- The message format allows us to produce a parser
- Successfully parses real-world messages of same type
  - all structural information was successfully recovered
- Rejects negative examples
  - different message types from same protocol
  - hand-crafted negative examples

Test Case	Length	Target	Padding	Pointer	Delimiter	Keyword	File	Repetition	Total
apache	0	0	0	0	4/5	6/6	1/1	1/2	12/14 (86%)
lighttpd	0	0	0	0	4/5	7/7	1/1	1/2	13/15 (87%)
ircnick	0	0	0	0	1/1	1/1	0	0	2/2 (100%)
ircuser	0	0	0	0	2/2	1/1	0	0	3/3 (100%)
smtphelo	0	0	0	0	1/2	1/1	0	0	2/3 (67%)
smtpquit	0	0	0	0	1/1	1/1	0	0	2/2 (100%)
smtpmail	0	0	0	0	3/5	3/3	0	0	6/8 (75%)
dnsquery	1/1	1/1	0	0	0	0	0	1/1	3/3 (100%)
nfslookup	4/5	4/4	2/2	0	0	0	1/1	0	11/11 (92%)
nfsgetattr	3/4	3/3	1/1	0	0	0	0	0	7/8 (88%)
nfscreate	4/5	4/4	2/2	0	0	0	0	0	10/11 (91%)
nfswrite	4/6	4/4	2/2	0	0	0	0	0	10/12 (83%)
smbnegotiate	2/2	2/2	1/1	0	1/1	10/10	0	0/1	16/17 (94%)
smbtree	2/3	2/2	0	1/1	2/2	3/3	0	0	10/11 (91%)
smbsession	8/9	8/8	0	7/7	2/2	2/2	0	0	27/28 (96%)

**Table 2. Field detection results: correctly identified fields / total fields in message format.**

# Related Work

*Secure Systems Lab  
Technical University Vienna*

- Network traces
  - M. Beddoe. The Protocol Informatics Project. Toorcon 2004
  - C. Leita, K. Mermoud, M. Dacier. ScriptGen: An Automated Script Generation Tool for Honeyd. ACSAC 2005
  - W. Cui, V. Paxson, N. Weaver, R. Katz. Protocol-Independent Adaptive Replay of Application Dialog. NDSS 2006
  - W.Cui, J.Kannan,H.J.Wang: Discoverer: Automatic Protocol Reverse Engineering from Network Traces
- Static and dynamic analysis
  - J. Newsome, D. Brumley, J. Franklin, and D. Song. Replayer: Automatic Protocol Replay by Binary Analysis. ACM CCS 2006.
- Dynamic taint analysis
  - J. Caballero and D. Song. Polyglot: Automatic Extraction of Protocol Format using Dynamic Binary Analysis. ACM CCS 2007
  - Z. Lin, X. Jiang, D. Xu, and X. Zhang. Automatic Protocol Format Reverse Engineering through Context-Aware Monitored Execution. NDSS 2008.

# Conclusions

*Secure Systems Lab  
Technical University Vienna*

- Reverse engineer application layer network protocols
- Recover a message format
- Validate format by parsing real world messages
- Tested on common servers and protocols

# Questions?

---

*Secure Systems Lab  
Technical University Vienna*