INPUT ATTACK TREES

Death of a thousand leaves

BlackHat Japan 2006
Heikki Kortti, Codenomicon

Heikki Kortti, Codenomicon, Black Hat Japan, Oct 5-6 2006.
How are vulnerabilities discovered?

- Sheer luck
- Source code inspection
- Reverse engineering
- Observing program behaviour
- Trying malicious inputs
INPUT ATTACK TREES  Why luck does not work

- Great if you have it, takes too long for everyone else
- Does not detect bugs *by the dozen*
- Does not provide details on where the fault is and how likely it is to occur elsewhere
INPUT ATTACK TREES

Inspection is slow and error-prone

- Software is complex
- Manual inspection takes ages
- Machine-assisted inspection finds false positives
- Source code may not be available
Reverse engineering reveals only part of the picture

- Great if you’re Halvar Flake, but too hard for everyone else
- Simply a “smoked-glass” view
Observation is tedious

- Trying to assess security by observing how the software functions is like trying to fix a cuckoo clock with an axe
- Observation is very slow
- Creating suitable stimuli to which the software should react is also very slow
INPUT ATTACK TREES  Don’t be smart, be evil

- Trying out malicious inputs works well
- Complete coverage = infinite time
- Time can be reduced by generating inputs automatically (fuzzing)
- Fuzzers can be
  - simple (non-structured) or
  - intelligent (structured)
INPUT ATTACK TREES  

Ways to do input testing

- By hand
- Create a program that tries the inputs for you
- Create a program that creates programs that try the inputs for you
The problem with simple fuzzing

- Fuzzing with random data is not enough for complex protocols like TLS or BGP
- A structural model of the protocol is needed
- “Intelligent fuzzing” or “robustness testing”
At a minimum, a fuzzing model has to capture the following:

- Field-level structures (8-bit integer, date field)
- Packet-level structures (header+payload)
- Context structures (packet sequences)
- Dynamic behaviour (runtime calculations, crypto functions, nonces, timestamps, lengths)
**INPUT ATTACK TREES**  Aiding the design of fuzzing models

- Use existing models
  - ABNF
  - ASN.1
  - XML

- Roll your own

- What makes a good model for designing effective attacks?
INPUT ATTACK TREES  Attack trees in general

- Well-known and proven methodology for observing and assessing the security of any system
- Provides a good overall view of the security of a system
- Risks and possible attacks can be viewed at a glance
Gain access to data in system

Break physical security
- Steal disk
- Access system console

Break virtual security
- View data with Tempest
- Exploit vulnerability
- Obtain password
- Crack trusted computer

Exploit vulnerability
- Obtain password

Obtain password
- Crack trusted computer

Steal disk
- Access system console
INPUT ATTACK TREES  Applying attack trees for input testing

• Input tree: all possible inputs for the defined interface
• Attack tree: a catalogue of all possible attacks against a system
• Input attack tree: an input tree augmented with attacks against all input branches
• Charts all of the possible attacks against an interface
INPUT ATTACK TREES  Benefits for attackers

- Easy to see which attacks have been tried
- Easy to see which attacks have not been tried
- Easy to see the vulnerable areas
- Easy to feed into a fuzzer generator for creating tests automatically
INPUT ATTACK TREES  Benefits for defenders

- What attacks do we need to protect against?
- What areas of the interface definition are too complex to get right?
- Have we tested all of our input handling code?
INPUT ATTACK TREES  Benefits for designers

• Easy to see which features are likely to be misimplemented

• Easy to see what branches will be attacked and when

• Similar branches can be compared with existing vulnerabilities

• Great feedback for the next release of the interface definition
INPUT ATTACK TREES  

Attack subtrees

- Create an attack once, reuse infinitely
- Attacks against a particular datatype can be used again and again
- Examples: date fields, integers, ASCII and UTF-8 strings, C-style format strings, URIs, IPv4 and IPv6 addresses, regexes
INPUT ATTACK TREES  Attaching attack subtrees

- Try negative length
- Try zero length
- Try large length

Length field
INPUT ATTACK TREES

Length field attack subtree (detailed)

16-bit MSB length field

0x00 0x00

0x00 0x01

0x00 0x0f

0x00 0x10

0x00 0x20

0x00 0x40

...
INPUT ATTACK TREES  Using the input attack tree

• Looking at the attack tree already provides good insights
• If tests are created automatically, you may want to create more test cases around the more problematic areas
• More test cases = more problems found
• More test cases = more time for testing
INPUT ATTACK TREES  Ways to focus the attacks

- Assign weights to branches
- Complex areas should be heavier
- An automated test generator can steer itself based on branch or leaf weights
- A light branch may merit only a few basic tests
- A very heavy branch needs really thorough coverage
INPUT ATTACK TREES  Testing an implementation

- Model the messages and message sequences
- Create the potential malicious inputs for all datatypes, structures and messages
- Attach attack subtrees to the main trunk
- Steer test case selection through weights
- Execute tests and observe the results
- The design and creation of a fuzzing framework and fuzzer have been omitted as trivial
INPUT ATTACK TREES

Example: DNS input tree

DNS

Test client

Test server

Zone transfer
INPUT ATTACK TREES  Testing a DNS server

- Test server
  - Use UDP
  - Use TCP
Testing a server over UDP

1. Use UDP
2. Issue query
3. Wait for response
4. Don’t wait for response
Expanding an input tree to an input attack tree

- Issue query
  - Valid query
  - Invalid query
Sample input attack tree for simple DNS queries

• Issue invalid query
  • Break lower level (UDP/TCP, IP, Ethernet)
  • Break DNS TCP length field (TCP only)
  • Break header
  • Break question section
  • Break answer section
  • Break authority section
  • Break additional section
  • Send response as query
  • Combine
• Surprisingly useful for some protocols
• Think complex, layered abominations like SOAP, RPC, Corba, HTTP
• Practical example: breaking an UDP datagram breaks a SIP implementation
INPUT ATTACK TREES  Break DNS TCP length

- All length fields are delicious
- Use zero length
- Use too large lengths
- Use too small lengths
- Use negative lengths
INPUT ATTACK TREES  Break header

- Bit-walk through all of the fields in the DNS header
- Interesting branches:
  - Try wrong and non-existing query types
  - Flip the AA, TC, RD, RA fields
  - Break the question/answer/authority/additional counts
INPUT ATTACK TREES  Break question section

- Break QNAME
- Break QTYPE
- Break QCLASS
- Add more than one question
- Omit question section
- Underflow
• Add one
  • Break all possible RRs and their substructures

• Add more than one
• Underflow
• Answer section can contain any number of RRs (resource records)
• RRs have complex substructures
• A server must parse the answer section also in queries
• Weigh down the branch with answer tests
INPUT ATTACK TREES  Weighing down the answer tests

Add answer section

Break RRs

Break owner  Break type  Break class  Break TTL  Break RDATA
INPUT ATTACK TREES

RDATA substructures are even more interesting

- SOA
- A
- PTR
- CNAME
- Break RDATA
- MX
- NS
- ...

Heikki Kortti, Codenomicon, Black Hat Japan, Oct 5-6 2006.
Out-of-context messages in a sequence can sometimes be very effective
INPUT ATTACK TREES  Combine any previous attacks

- Break question count and question section
- Break length and use underflow
- Use wrong length and illegal characters
- Break answer section and send answer as query
- Break length and use invalid offsets
INPUT ATTACK TREES

Read the specification like the devil reads the Bible

- Make labels longer than 63 octets
- Make label sequences longer than 255 octets
- Use a zero-length label in a non-root position
- Try asking for AXFR over UDP
- Use illegal characters
- Try non-specified types
- Try reserved values
- Try invalid combinations (source address = destination address, etc.)
INPUT ATTACK TREES

CONCLUSIONS
By viewing an interface as an input tree, we can easily see what are its weak points.

Attacks can be attached to the tree easily.

Tests can be created by traversing the tree.

Tests can target complex areas more heavily.

Can be used for attack and defense.

Can be applied to testing any interface.
INPUT ATTACK TREES

Inspiration

Schneier, Bruce: Attack Trees

Moore, Andrew; Ellison, Robert; Linger, Richard: Attack Modeling for Information and Survivability
http://www.cert.org/archive/pdf/01tn001.pdf

Convery, Sean; Cook, David; Franz, Matthew: An Attack Tree for the Border Gateway Protocol (obsoleted Internet draft)

Hares, Susan: BGP Attack Trees: Real World Examples
INPUT ATTACK TREES
ANY QUESTIONS?

THANK YOU!

Heikki Kortti
hkortti@codenomicon.com