Software Security Assessment through Specification Mutations and Fault Injection

- Vulnerabilities caused by robustness problems exist in software
- Software robustness can be assessed using interface fault injection technique
- WAP gateway assessment was conducted as an example and numerous security problems were found
- Aim is to support early elimination of trivial vulnerabilities

PROTOS Project

- This work is done in project PROTOS - "Security Testing of Protocol Implementations" running 1999-2001
- VTT (Technical Research Centre of Finland) Electronics
- OUSPG (University of Oulu, Secure Programming Group)
- Funded by TEKES (National Technology Agency) and partner companies
- Contact
  rauli.kaksonen@vtt.fi
- Project Internet home page
  http://www.ee.oulu.fi/research/ouspg/protos

Setting

- Software which tolerates unexpected input is *robust*
- **Robustness problems are security problems** as well, these flaws (e.g. buffer overflows) can be exploited to compromise a system from outside
- Contemporary software is infested with robustness problems causing security holes (see e.g. *BugTraq*)

Protocol Security Assessment

- Protocol implementations are logical targets for security analysis
- Messages are often transmitted over the Internet or other insecure networks, which exposes them to malicious modification
- Cryptographic protections are not effective against attacker who can negotiate a legal session
- In interface *fault injection* software is purposefully fed with exceptional and/or erroneous input through interfaces
- **Interface fault injection can simulate attacks through network connections**

The Assessment Approach

- Non-traditional testing using interface fault injection
- *Black-box* approach, no source code is required
- Test are designed by mutating message syntax, message content and message exchanges
- **Output is not checked for specification conformance, this greatly reduces the needed effort**
- The semantic meaning of messages and exchanges is preserved, as far as possible (contrast to *random testing*)
- The hypothesis is that carefully constructed input is more likely to find errors

Test Design

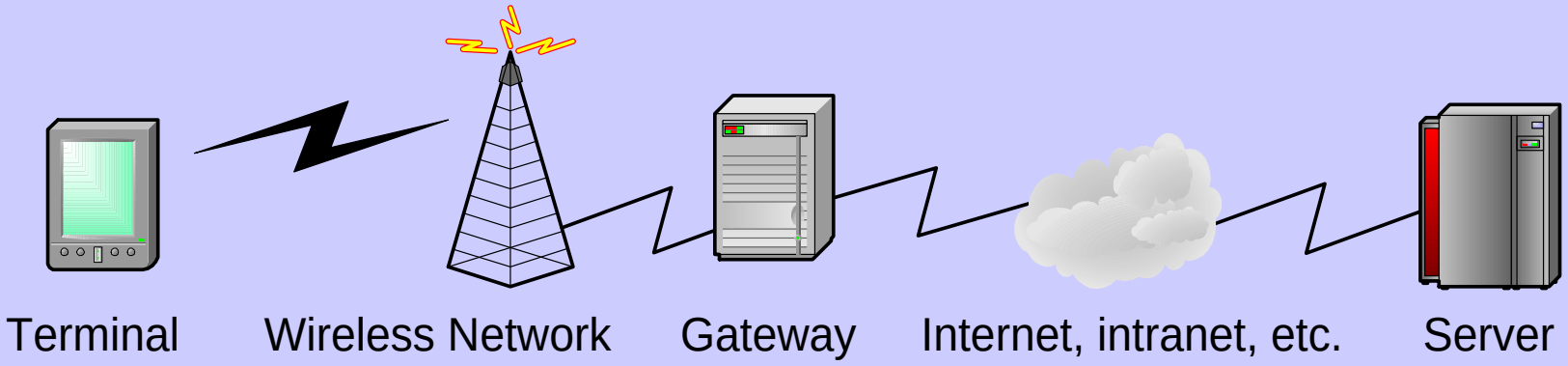
1. Acquire or write machine processable protocol specification (an extended dialect of BNF)
2. Simplify the specification by removing elements irrelevant to the test purpose, i.e. limit the test input space
3. Add rules for maintaining semantic validity, if possible
4. Set protocol base elements to have typical values
5. Add *exceptional elements* to protocol specification as alternatives to existing elements, i.e. mutate specification
6. Design *test cases* by selecting suitable combinations of elements

Mini-Simulations

- Test cases are executed using a prototype *mini-simulation* environment
- Mini-simulation is a BNF-formed executable specification augmented with Java-implemented rules
- Mini-simulation provides only the minimum functionality needed for sending the test input
- The environment provides flexibility for mutating the specification and selecting test cases

WAP-WSP-Request Test Suite

- WAP is a family of protocols for delivering services and Internet content to wireless terminals
- A *WAP gateway* mediates traffic between terminals and content providers
- *WAP-WSP-Request test suite* assesses the ability of a WAP gateway to handle maliciously formatted WSP messages



WAP Testing Motivation

- Security of WAP gateways is essential since even encrypted traffic will be exposed as plain text inside a gateway
- Using a workstation or a laptop with a modem and a phone an intruder can send malicious messages
- **WAP is an important milestone for getting Internet to the phones and acts as an example for things to come**
  - The overall security of a WAP system was not assessed
  - However, a single vulnerable point is sufficient to totally compromise a system

WAP-WSP-Request Test Design

- The starting point of test design was the WSP-request part of the WAP specification, which was mutated to add exceptional elements
- A simple WSP request has the form

```
0x01 0x40 0x1a  "http://127.0.0.1/index.wml"
```

- The specification was mutated using 36 different groups of mutations, the total number of selected test cases was 4236
- For example, the protocol field "`http`" was replaced with different longish strings for finding potential buffer overflows

```
0x03 0x40 0x1a "aaaa://127.0.0.1/index.wml"
0x04 0x40 0x1e "aaaaaaaa://127.0.0.1/index.wml"
0x05 0x40 0x26 "aaaaaaaaaaaaaaaa://127.0.0.1/index.wml"
```

Test Results

- The test suite was executed against seven different WAP gateways from different vendors
- Total number of 4326 test cases in 36 groups
- **Test runs against all seven gateways contained failed test cases** indicating *potential vulnerability*

Gw	*Failed cases/groups*	Gw	*Failed cases/groups*
1	569/10	5	664/8
2	141/18	6	622/14
3	10/2	7	148/20
4	385/16		

Test Analysis

- Four gateways were *verified to be vulnerable* beyond denial-of-service using a buffer-overflow based exploit
- Total compromise of the gateway services based on any of these four implementations was demonstrated
- Test results were sent to the vendors
- Reactions varied, but all were positive (as far we know, the individual flaws we found are now fixed)
- Some vendors indicated that they will take actions to prevent vulnerabilities of this kind in the future
- **As an indicator of the security of future technology this gives a warning for us all**

Public Test Suite

- The test suite was made publicly available after a *grace period*
  http://www.ee.oulu.fi/research/ouspg/protos
- The exploits against the vulnerabilities and the names of the tested products are excluded
- The test cases are in binary form without explaining their structure
- The aim is to make the material available for all vendors and their customers and to promote public discussion

Discussion

- The effectiveness of this simple method is surprising
  - Addition to WAP, we have tested implementations of various other protocols, the end results are mostly similar
- Problems similar to ones we found are constantly reported in the Internet by casual evaluators using ad-hoc methods
- A systematic approach should be used to assess software components before they are taken into serious use
- This could enhance the overall Internet security:
  - Many vulnerabilities are found and fixed early
  - Clients assess software robustness before deployment
  - **Software is implemented to have higher quality in the first place**

Conclusions

- The presented work was motivated by large number of robustness and security problems in contemporary software
- Systematically injecting exceptional input into software components reveals robustness problems
- No source code is required and testing effort is less than in traditional testing
- As an example, 7/7 tested WAP gateways were found to have robustness problems, four were demonstrated to be vulnerable.
  - **As an indicator of the things to come this gives a warning**
- Use of robustness evaluation would promote production of more secure software

Thank You!

- Any questions?
- Contact
  rauli.kaksonen@vtt.fi
- PROTOS project Internet home page
  http://www.ee.oulu.fi/research/ouspg/protos