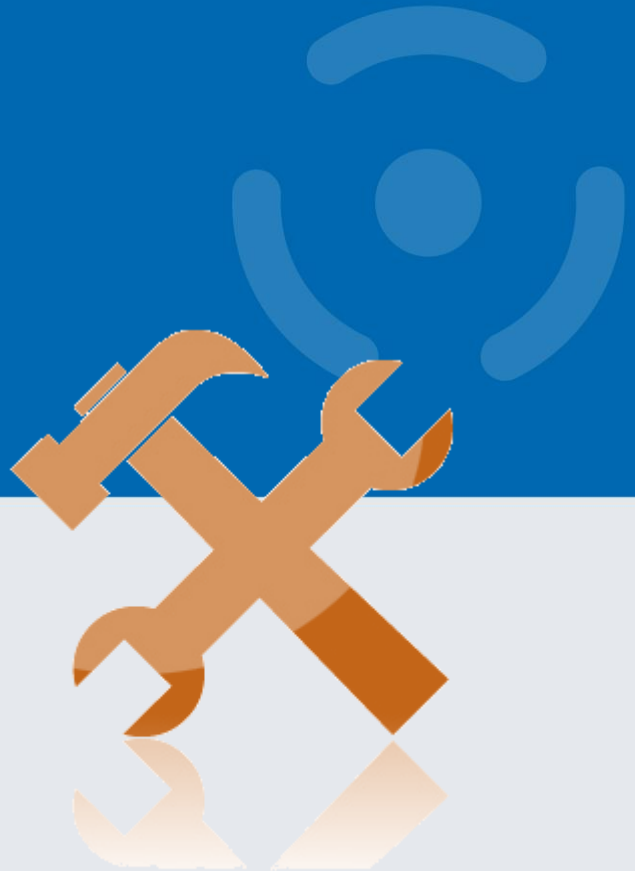
PA193 - Secure coding principles and practices



Dynamic analysis, fuzzing

Petr Švenda svenda@fi.muni.cz

CROCS
Centre for Research on Cryptography and Security

LAB

Static and Dynamic analysis combined

- Download problematic code buggy.cpp from IS
- Perform operation and observe output
- Comment out problems found and note tool name
  - only that found by the particular tool
- Compilation only
  - Compile with MSVC /W4
  - Compile with g++ -Wall -Wextra -g
- Compile and run
  - **`MSVC /RTC /GS (on by default)`**
  - g++ **`-fstack-protector-all`**

Windows vs. Linux

- For Windows tools use Visual Studio
- For Linux tools
  - ssh aisa.fi.muni.cz
  - Compile with g++
  - Use -g flag
  - Run valgrind’s memcheck and exp-sgcheck

Static and Dynamic analysis combined (2)

- Run static analysis
  - Run Cppcheck
  - Run PREfast
- Run dynamic analysis

```
valgrind --tool=memcheck --leak-check=full ./yourprogram
valgrind --tool=exp-sgcheck ./yourprogram
```

Decide for every tool

- What type of issues were detected?
- What are the limitations of tool?
- *Stack* vs. *heap* vs. *static* memory issues detected
- *Local* vs. *global* (function) issues detected
- *Static analysis* vs. *dynamic analysis*

FUZZING

Pre-prepare

- Download zip with all binaries and data from IS

- Optional: if you need WinDbg, use:
  - Standalone Debugging Tools for Windows (WinDbg) is enough
  - https://msdn.microsoft.com/en-us/windows/hardware/hh852365

Microsoft's SDL MiniFuzz File Fuzzer

- Application input files fuzzer
  - http://www.microsoft.com/en-us/download/details.aspx?id=21769
- Templates for valid input files (multiple)
- Modify valid input file (randomly, % aggressiveness)
- Run application with partially modified inputs
- Log resulting crash (if happen)
  - exception, CPU registers...
- Video overview
  - http://msdn.microsoft.com/en-us/security/gg675011.aspx

Microsoft's SDL MiniFuzz File Fuzzer

MiniFuzz

Target

Process to fuzz: C:\Program Files (x86)\IrfanView\i_view32.exe Browse

Command line args: %1

Allow process to run for: 2,0 secs.

Shutdown method: Terminate Process Shutdown delay: 0,5 secs.

Thread Injection
Send WM_CLOSE
Terminate Process

Settings

Template files: C:\Users\petrs\Desktop\minifuzz\templates\ Browse

Temporary files: C:\Users\petrs\Desktop\minifuzz\temp\

Log files: C:\Users\petrs\Desktop\minifuzz\logs\

Crash files: C:\Users\petrs\Desktop\minifuzz\crashes\

Aggressiveness: Low (5%) Always on Top

Start Fuzzing | Stop Fuzzing | View Log Dir | TFS Settings... | Help | About

Progress

Fuzzed files: 16 # Failures: 0 beer.jpg

Time	File	Crash

Play with SDL MiniFuzz

- Goal: crash IrfanView v1.75 (1996)
  - Image file goes as first argument

1. Select target executable (bin\I_VIEW32_1.75.exe)
2. Gather at least one input file into template folder
   - Template files directory, copy data\Icon_ManBig_128.GIF from zip file
3. Set proper shutdown method (experiment)
4. Run and observe crashes (log, crashing images)

Play with SDL MiniFuzz – bonus tasks

- Where can you find crashing images?
- Bonus: Can you increase the speed of testing?
- Bonus: What is the impact of aggressiveness?
- How can you test your application?

- Note: MS SDL requires 100k runs without failure

Radamsa fuzzer

- *"…easy-to-set-up general purpose shotgun test to expose the easiest cracks…"*
  - https://code.google.com/p/ouspg/wiki/Radamsa
- Just provide input files, all other settings automatic
  - **cat** file **|** radamsa **>** **file.fuzzed**

```
>echo "1 + (2 + (3 + 4))" | radamsa --seed 12 -n 4
 1 + (2 + (2 + (3 + 4?)
 1 + (2 + (3 +?4))
 18446744073709551615 + 4)))
 1 + (2 + (3 + 170141183460469231731687303715884105727))
```

Radamsa as file fuzzer (XML example)

- **`radamsa -o fuzz_%n.xml -n 10 *.xml`**
  - Takes file template from *`.xml` file(s)
  - Generates given number (10) of fuzzed files (-n 10)
- Testing you application
  1. Collect valid input file(s) for target app into *.xml file(s)
  2. Run Radamsa to create large number of fuzzed files
  3. Run your application with fuzzed input file and monitor
     - Custom code for monitoring (e.g., crash detected by success in acquire of named mutex)
     - WinDbg for monitoring, parse output log file
- Example:
  - use data\books.xml as template
  - generate 10 fuzzed variants and inspect result in text editor

Radamsa as fuzzing client – test server

- **`radamsa ip:80 -n 10 -o fuzz_%n.http-req samples/*.http-req`**
  - Connects as client to server at ip:80, runs infinitelly (-n inf)
  - Takes template inputs from **`*.http-req`** file(s)
  - Send fuzzed input to server and store it into **`fuzz_%n.http-req`** files
- Testing you server
  1. Capture valid request for your client to server (e.g., GET request) and store into *.http-req file(s)
  2. Run (repeatedly) Radamsa as TCP client
  3. Monitor behaviour of your server under Radamsa requests
- Test against astrolight.cz (use data\astrolight.http-req)
- Important: always tests only your servers or with owner consent!!!

Radamsa as fuzzing server – test client

- `radamsa -o fuzz_%n.http-resp :8888 -n inf samples/*.http-resp`
  - Starts as server on port 8888, runs infinitelly (-n inf)
  - Takes template inputs from `*.http-resp` files
  - Return fuzzed input to connecting client
- Testing you client
  1. Capture valid responses from your server (e.g., HTML page) and store into *.http-resp file(s)
     - Use data\string.http-resp as template
  2. Run Radamsa as server (see above)
  3. Run your client (repeatedly, browser) and monitor its behaviour

Questions for Radamsa

- In what is SDL MiniFuzz better then Radamsa?
- Why is Radamsa better in fuzzing text files?
- Can you fuzz vulnserver.exe?
  - 127.0.0.1:9999
- How to test server/client in stateful protocol?

OPTIONAL – PEACH FRAMEWORK

Vulnerable server (vulnServer.exe)

- Only for Windows
  - for Linux, consider OWASP Mutillidae
- Vulnerable server inside VulnServer.zip
- Run it – waits for connection
- Connect via telnet (putty)
  - host=localhost port=9999
- Type HELP

- Server is vulnerable, we will try to crash it by fuzzing

Peach – fuzzing vulnerable network server

1. Prepare Peach Pit file (example hter_pit.xml)
   - data model, state model, agent…
2. Run Peach Agent (first terminal)
   - `peach -a tcp`
3. Run Peach fuzzing (second terminal)
   - `Peach hter_pit.xml TestHTER`
   - Wait for detected crash (fault)
4. Inspect directory with crash logs
   - *Logs\hter_pit.xml_TestHTER_???\Faults\EXPLOITABLE_???*
5. Debug crash using fuzzed data from crash log
   - E.g., *1.Initial.Action.bin, 2.Initial.Action_1.bin…*

```
<DataModel name="DataHTER">
 <String value="HTER " mutable="false" token="true"/>
 <String value=""/>
 <String value="\r\n" mutable="false" token="true"/>
</DataModel>

<StateModel name="StateHTER" initialState="Initial">
 <State name="Initial">
  <Action type="input" ><DataModel ref="DataResponse"/></Action>
  <Action type="output"><DataModel ref="DataHTER"/></Action>
  <Action type="input" ><DataModel ref="DataResponse"/></Action>
 </State>
</StateModel>

<DataModel name="DataResponse">
 <String value=""/>
</DataModel>

<Agent name="RemoteAgent" location="tcp://127.0.0.1:9001">
 <!-- Run and attach windbg to a vulnerable server. -->
 <Monitor class="WindowsDebugger">
    <Param name="CommandLine" value="vulnserver.exe"/>
    <Param name="WinDbgPath" value="c:\Program Files (x86)\Windows Kits\8.1\Debuggers\x64\" />
 </Monitor>
</Agent>

<Test name="TestHTER">
 <Agent ref="RemoteAgent"/>
 <StateModel ref="StateHTER"/>
 <Publisher class="TcpClient">
  <Param name="Host" value="127.0.0.1"/>
  <Param name="Port" value="9999"/>
 </Publisher>

 <Logger class="File">
  <Param name="Path" value="Logs"/>
 </Logger>
```

Model of input data
'HTER *anything* \r\n'

1. Read any string
2. Send fuzzed input
3. Read any string

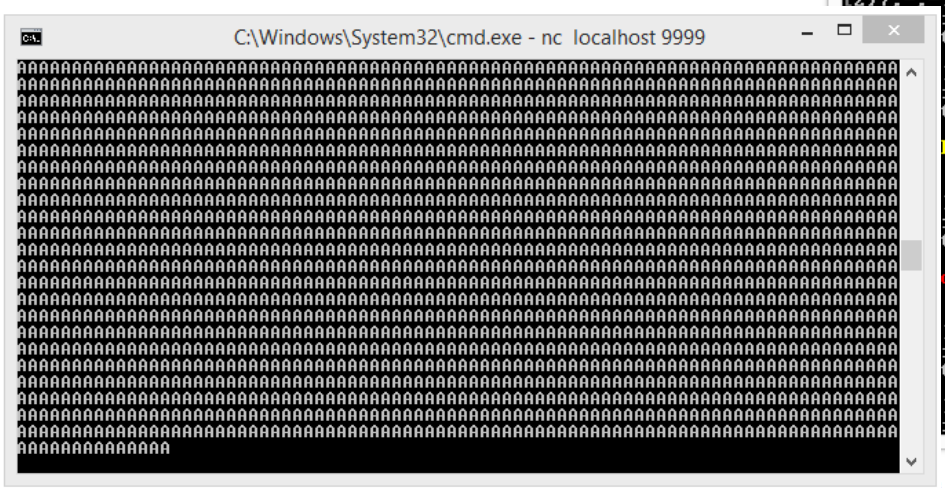
Agent responsible for starting target application with debugger connected

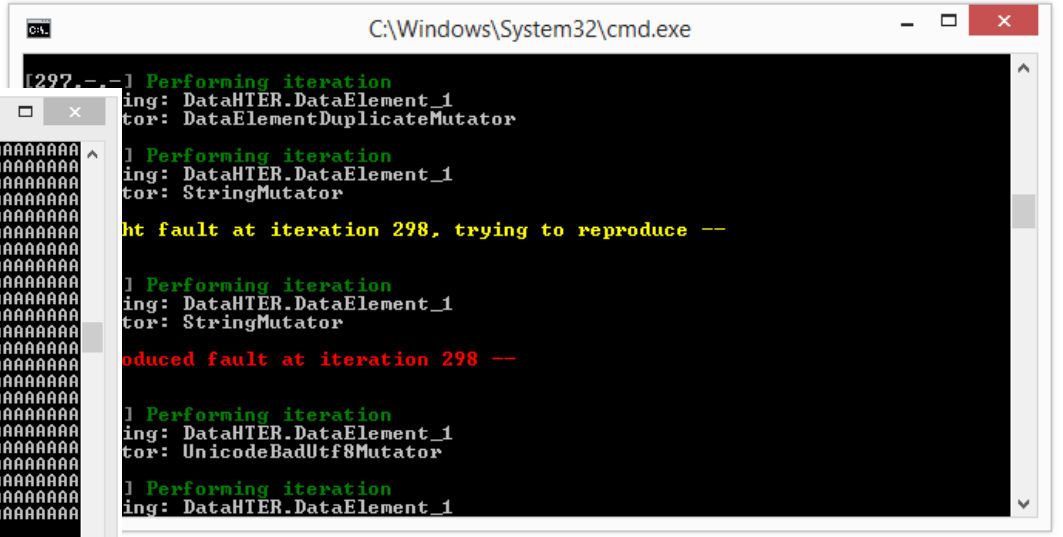
Test scenario with specified settings

How to communicate with target application

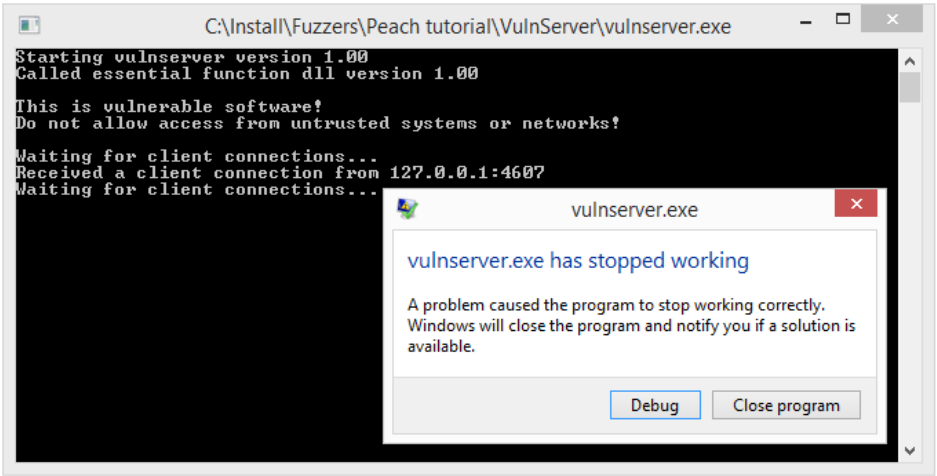
How to store results

```
<DataModel name="DataHTER">
  <String value="HTER " mutable="false" token="true"/>
  <String value=""/>
  <String value="\r\n" mutable="false" token="true"/>
</DataModel>
```






```
al\VulnServer\vulnserver.exe"/>
its\8.1\Debuggers\x64\" />
```



Example from http://rockfishsec.blogspot.ch/2014/01/fuzzing-vulnserver-with-peach-3.html

Questions for Peach

- Is Peach able to fuzz stateful protocols?
- Is Peach able to specify custom data format?
- Does Peach monitor only application crash?

Homework

- Create your own C/C++ compile-able program
  - 1kB size at maximum
  - including main function, must compile under both gcc & MSVC
- Insert as many different vulnerabilities like buffer overflow, string format problems, memory corruptions (stack / heap) as you can
  - principally different bugs will be counted
  - document bugs inserted/found in separate report
- Run various static and dynamic checkers on your program
  - Compiler, CppCheck, PREFast, suitable fuzzer
  - create report from these tools, showing real problems detected, false positives, false negatives
- Deadline for submission is 5.11. 23:59 (groups 1-3), 10.11. 23:59 (group 4)