PA193 - Secure coding principles and practices

LAB: Dynamic analysis, fuzzing

Łukasz Chmielewski *chmiel@fi.muni.cz* **(based on the seminar by P. Svenda)**
(email me with your questions/feedback)
Centre for Research on Cryptography and Security, Masaryk University

CROCS
Centre for Research on Cryptography and Security

Note: password for the various activities

- If you are asked for password (Miro boards…) use ‘fimunicz’

SOLO ACTIVITY: RUN TOOLS, ANALYZE

Static and Dynamic analysis combined

- Download problematic code buggy.cpp from IS
- Perform operation and observe output
  - note tool name which found a particular bug
- Compilation only
  - Compile with MSVC /W4
  - Compile with **`g++ -Wall -Wextra -g`**
- First compile and run
  - **`MSVC /RTC /GS (on by default)`**
  - g++ **`-fstack-protector-all`**
- If you have time: **`BufferOverflow.cpp`**

Windows vs. Linux

- For Windows tools: use Visual Studio, cppcheck…
  - Use SAL (see next slide)
- For Linux tools:
  - Use your own machine or
  - Use Aisa:
    - `ssh xxx@aisa.fi.muni.cz`
    - Use cppcheck
    - Compile with `g++ -g buggy.cpp`
    - `scp .\buggy.cpp xxx@aisa.fi.muni.cz:.`
- Run dynamic analysis (own computer or Aisa)
  `valgrind --tool=memcheck --leak-check=full ./yourprogram`
- Run stack dynamic analysis on own comp. or Aisa (see one of the next slides)

Windows: Visual Studio & PREfast & SAL

- Use:
  - `_Out_writes_bytes_all_`
- In particular:

```
int memcheckFailDemo(
   _Out_writes_bytes_all_(arrayStackLen) int* arrayStack,
   unsigned int arrayStackLen,
   _Out_writes_bytes_all_(arrayHeapLen) int* arrayHeap,
   unsigned int arrayHeapLen);
```

- In particular:
  - https://www.codeproject.com/Reference/879527/SAL-Function-Parameters-Annotations

Linux:

- Can you run: **`valgrind --tool=exp-sgcheck`** `./yourprogram`
  - It is not supported anymore! Available till version 3.15.0. It is hard to install now, but I will demo it.
- From 3.16 release notes:

```
The exprimental Stack and Global Array Checking tool has been removed.
It only ever worked on x86 and amd64, and even on those it had a
high false positive rate and was slow.  An alternative for detecting
stack and global array overruns is using the AddressSanitizer (ASAN)
facility of the GCC and Clang compilers, which require you to rebuild
your code with -fsanitize=address.
```

- Run dynamic analysis (own computer or Aisa) as follows:
  - Compile: **`g++ -fsanitize=address buggy.cpp`**
  - Run: `./yourprogram` (usually `./a.out`)
- Demo (see the screen with valgrind 3.15 on my VM):
  - **`valgrind --tool=exp-sgcheck`** `./a.out`

Questions: 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*
- Why Valgrind-memcheck missed some memory leaks detected by Cppcheck?
  - What you need to change so memcheck will find it?
  - How is this relevant to test coverage?

Group activity: Discuss false positives/ negatives

- Groups of 3 students (breakout rooms), Discuss and reason within the group
- Answer questions from the previous slide, write into mindmap
- Link for Miro board:
  - https://miro.com/app/board/uXjVNkRaA_4=/
  - (one of you can Share the screen with Miro board for easier group discussion)

FUZZING

Pre-prepare

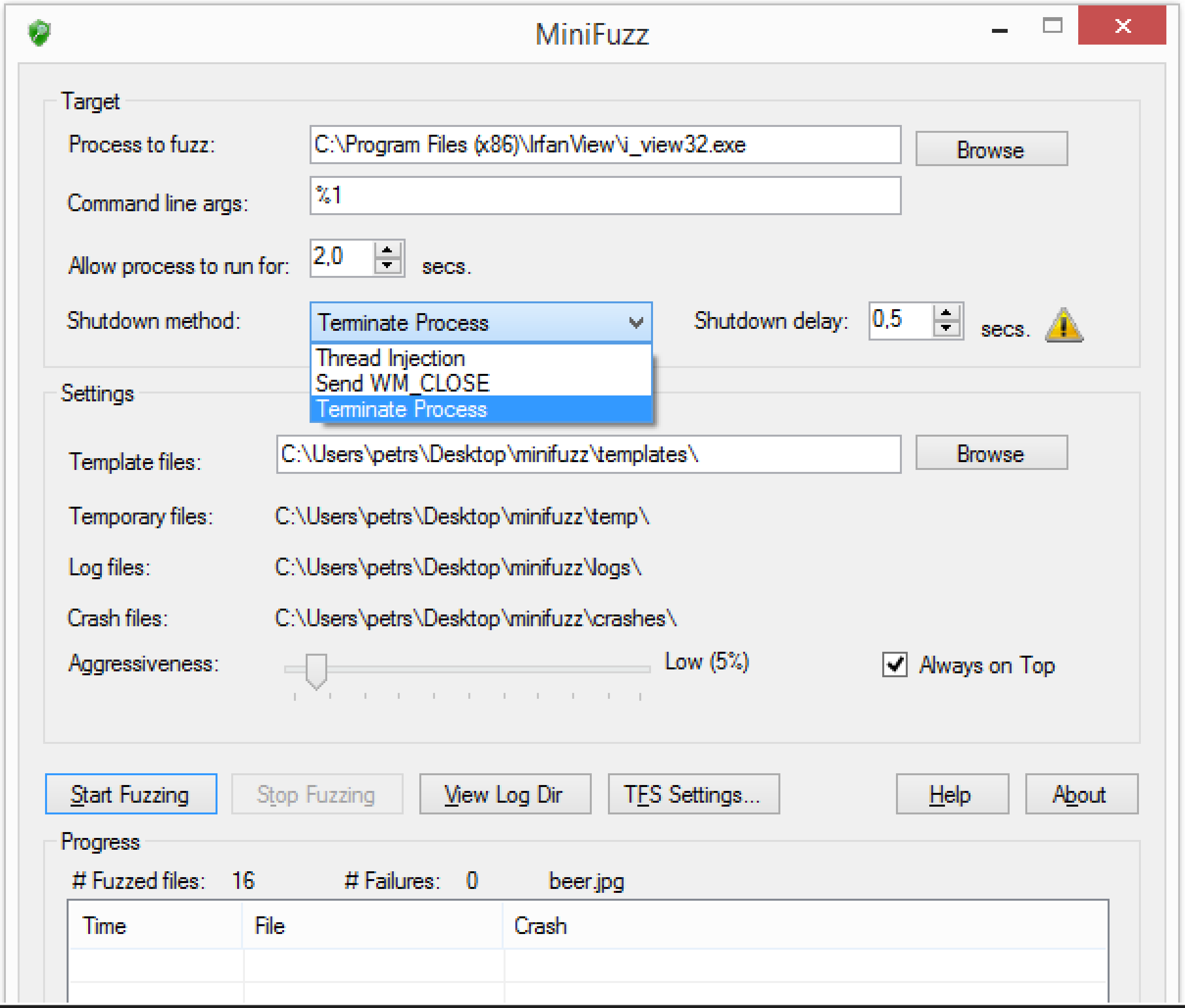
- Download zip with all binaries and data from IS

- If you want to try to do optional task then let me know!
- Optional: if you to use for extra tasks 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
  - See the IS documents
- 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...

Microsoft's SDL MiniFuzz File Fuzzer



MiniFuzz

Target

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

Command line args: %1

Allow process to run for: 2,0 secs.

Shutdown method: Terminate Process

Thread Injection
Send WM_CLOSE
Terminate Process

Shutdown delay: 0,5 secs.

Settings

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

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

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

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

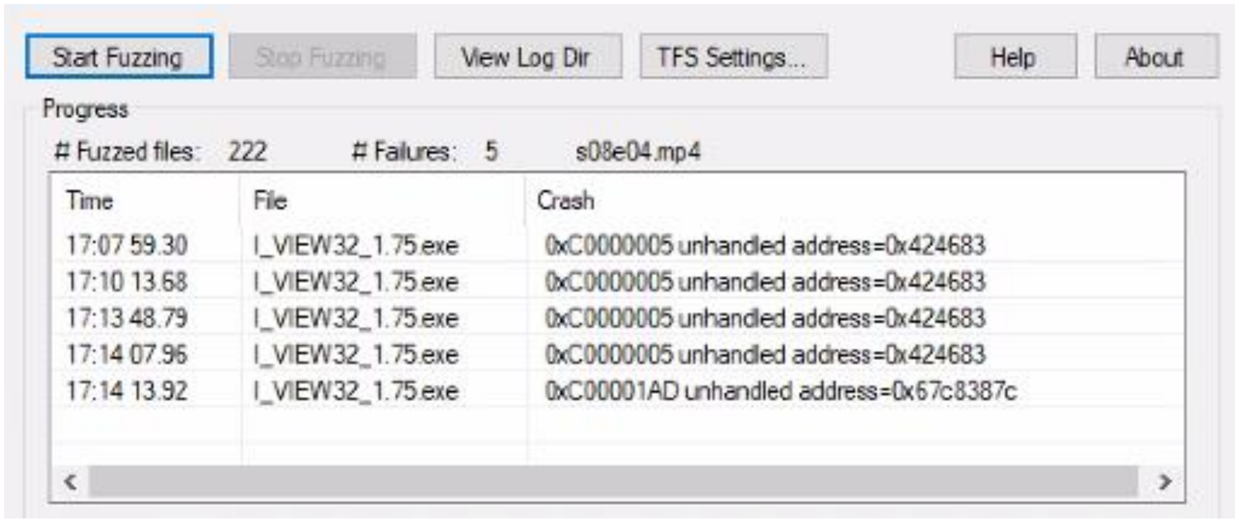
Aggressiveness: Low (5%)

[✓] Always on Top

Progress

Fuzzed files: 16 # Failures: 0 beer.jpg

Time	File	Crash

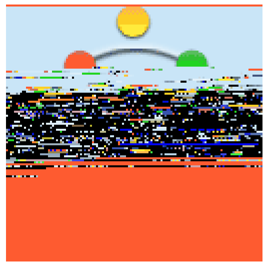


Progress

Fuzzed files: 222 # Failures: 5 s08e04.mp4

Time	File	Crash
17:07 59.30	I_VIEW32_1.75.exe	0xC0000005 unhandled address=0x424683
17:10 13.68	I_VIEW32_1.75.exe	0xC0000005 unhandled address=0x424683
17:13 48.79	I_VIEW32_1.75.exe	0xC0000005 unhandled address=0x424683
17:14 07.96	I_VIEW32_1.75.exe	0xC0000005 unhandled address=0x424683
17:14 13.92	I_VIEW32_1.75.exe	0xC00001AD unhandled address=0x67c8387c

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. Copy 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, Terminate Process)
4. Run and observe crashes (log, crashing images)

Play with SDL MiniFuzz – bonus tasks

- Where can you find images that caused crash?
- Bonus: Can you increase the speed of testing?
- Bonus: What is the impact of aggressiveness?
- How can you test your application?
- How can you test VLC with 1.9GB movie?

- Note: MS SDL requires 100k runs without failure

Radamsa fuzzer

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

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

- On Windows: use radamsa-0.4_win.exe from IS
- On Linux: Download from https://github.com/aoh/radamsa/releases

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 your 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. Fuzz some program that parses XML files – we do not do it in this exercise. Just generate the files.
- Example:
  - use data\books.xml as template (you can also use **`--seed x`**):
    `radamsa -o fuzz_%n.xml -n 10 ..\data\books.xml`
  - generate 10 fuzzed variants and inspect the result in text editor

Radamsa as fuzzing client – test server

- **`radamsa -o ip:80 -n inf 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 the server and store it into **`fuzz_%n.http-req`** files
- Testing your server
  1. Capture a 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 the behavior of your server under Radamsa requests
- Test against localhost at port 8000:
  - Create a localhost server with: **`python -m http.server 8000`**
  - First test with http://localhost:8000/
  - use data\astrolight.http-req as a starting point, we will modify it together
- Important: always test only your servers or with the owner's consent!!!
- In principle, you can use Wireshark to observe fuzzing:
  - https://www.wireshark.org/

Radamsa as fuzzing server – test client

- `radamsa -o :8888 -n inf samples/*.http-resp`
  - Starts as a server on port 8888, runs infinitely (-n inf)
  - Ip can be empty if localhost
  - Takes template inputs from `*.http-resp` files
  - Return fuzzed input to connecting client
- Testing your 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 a server (see above)
  3. Run your client (repeatedly, browser or curl, but remember about option `--http0.9`) and monitor its behaviour

Questions for Radamsa

- In what is SDL MiniFuzz better than Radamsa?
- Why is Radamsa better in fuzzing text files?
- How can you combine Radamsa and MiniFuzz?

- Can you fuzz vulnserver.exe?
  - 127.0.0.1:9999
- How to test server/client in stateful protocol?

ASSIGNMENT 2

Assignment 2: Code buggy (as hell)

- Create your own C/C++ compile-able program
  - 1kB size at maximum (STRICT REQUREMENT!)
  - Including main function, must compile under both gcc/g++ & MSVC
- Insert as many (>>10) different vulnerabilities
  - buffer overflow, string format problems, memory corruptions (stack / heap) as you can
  - Only principally different bugs will be counted
  - Document bugs inserted/found in a separate report
  - Some vulnerabilities should be triggered the attacker providing input, so fuzzing can be successful too!
- Run various static and dynamic checkers on your program
  - Compiler (+flags), CppCheck, PREFast, **`valgrind`**, suitable fuzzer…
- Points: static part: 4, dynamic part: 4, the rest (reporting, etc.): 2
  - Optional: Implement fuzzing using AFL++. Provide a description how you used it. Bonus: 2 extra points.
  - For inspiration: https://aflplus.plus/docs/tutorials/

Assignment 2: Code buggy (as hell)

- Produce short (4xA4) text with description of your solution
  - Create report from results obtained by running the analysis tools
  - Create table with all problems inserted and if detected by given tool
    - Rows == Problems, Columns == tool result
  - Highlight false positives and false negatives (and discuss why)
- What to submit
  - Source code of buggy application
  - Results from analysis tools
  - Report with the description of your solution
- When and where to submit
  - Submit **before 13.3.2023 23:59** into IS HW vault
  - Soft deadline: -3 points for every started 24 hours

CHECK-OUT

Checkout

- Which of the seminar parts you enjoyed most?
- Write three items you liked (ideally inserted as single word each)
- Write to sli.do when displayed

THANK YOU FOR COMING, SEE YOU NEXT WEEK

OPTIONAL, OWN WORK

Clang

- Try clang static analyser on buggy.cpp
- https://clang-analyzer.llvm.org/scan-build.html
- Do you find more issues than with the other tools?

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'

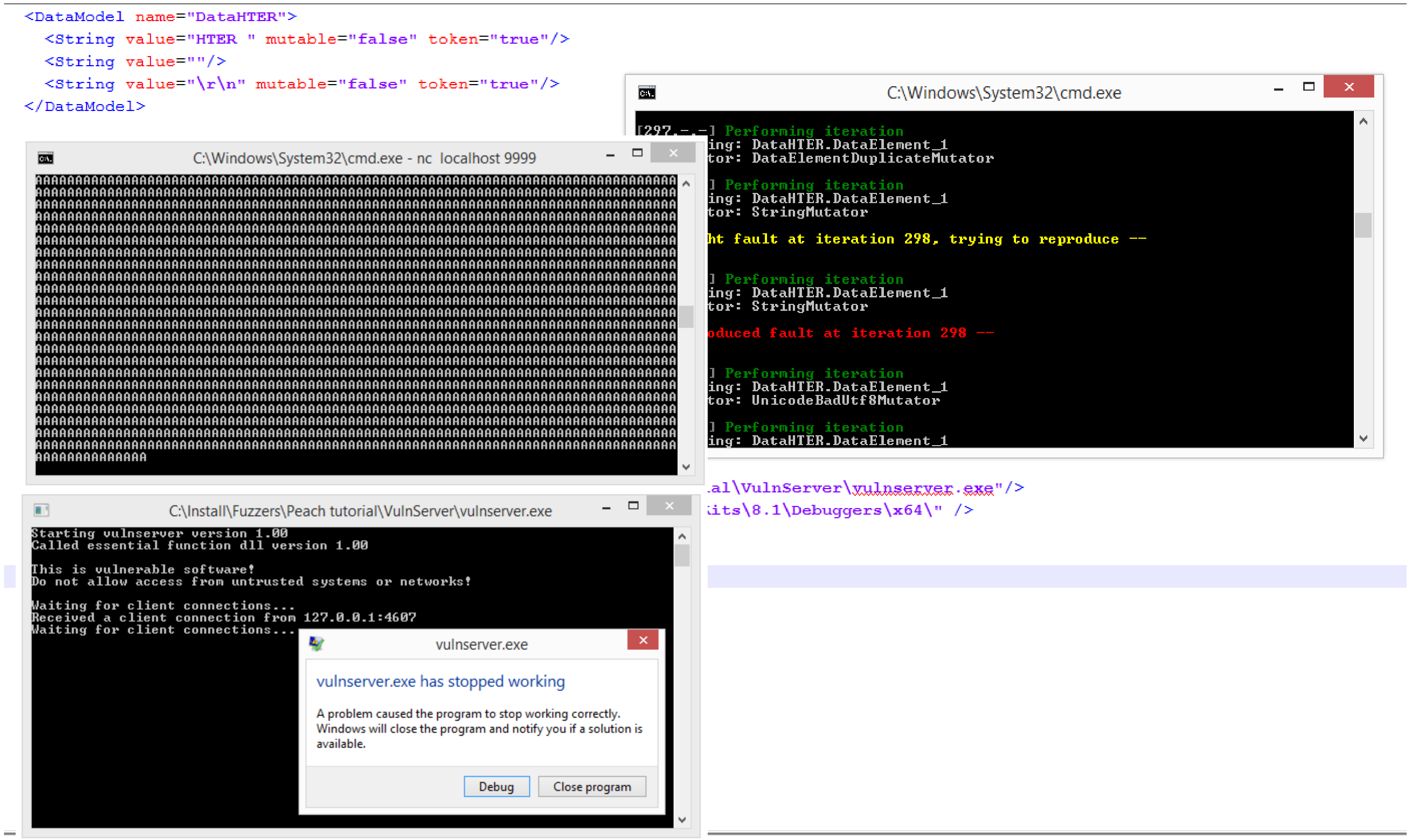
1. Read any string
2. Send fuzzed input
3. Read any string

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>
```

C:\Windows\System32\cmd.exe

```
[297,-,-] Performing iteration
ing: DataHTER.DataElement_1
tor: DataElementDuplicateMutator

] Performing iteration
ing: DataHTER.DataElement_1
tor: StringMutator

ht fault at iteration 298, trying to reproduce --

] Performing iteration
ing: DataHTER.DataElement_1
tor: StringMutator

oduced fault at iteration 298 --

] Performing iteration
ing: DataHTER.DataElement_1
tor: UnicodeBadUtf8Mutator

] Performing iteration
ing: DataHTER.DataElement_1
```

C:\Windows\System32\cmd.exe - nc localhost 9999

```
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAA
```

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

C:\Install\Fuzzers\Peach tutorial\VulnServer\vulnserver.exe

```
Starting vulnserver version 1.00
Called essential function dll version 1.00

This is vulnerable software!
Do not allow access from untrusted systems or networks!

Waiting for client connections...
Received a client connection from 127.0.0.1:4607
Waiting for client connections...
```

vulnserver.exe

vulnserver.exe has stopped working

A problem caused the program to stop working correctly. Windows will close the program and notify you if a solution is available.

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?

```
# powershell.exe -ExecutionPolicy Bypass ./ff_radamsa.ps1 beer.jpg irfan.exe 10
$fileTemplate = $args[0]
$fileTemplateResolved = Resolve-Path $args[0]
$targetApp = Resolve-Path $args[1]
$totalRuns = $args[2]
$radamsa= Resolve-Path "radamsa.exe"
$count=1
while ($count -le $totalRuns) {
   $fuzzFileName = "fuzz-" + $count + "_" + $fileTemplate
   $fuzzFileWindbRes = $fuzzFileName + ".wdbg.log"
   # run Radamsa to generate single fuzzed file
   & $radamsa -o $fuzzFileName $fileTemplate
   Write-Host "New file $fuzzFileName generated"
   # run target application with fuzzed file as argument under WinDbg monitoring
   & windbg -logo $fuzzFileWindbRes $targetApp $fuzzFileName
   # wait some time
   Start-Sleep -s 2
   # terminate target program inside windbg
   $a = Get-WmiObject win32_process -Filter "name = 'windbg.exe'"
   $a | % {Invoke-WmiMethod -Name terminate -InputObject $_ | out-null}
   # TODO: parse output log files *.wdbg.log
   $count++
}
```

Own work – fuzz student-selected app

- Find any application on internet and fuzz it
  - Make sure you can execute it on your machine
  - Various image and movie players are good targets
  - Download some old(er) release – more bugs possibly
- Try to fuzz it to crash (MiniFuzz, Radamsa)
- Inspect results and discuss