

Homework 2

MICS-204, Summer 2024

Karl-Johan Westhoff
email kjwesthoff@berkeley.edu

UC Berkeley School of Information
MICS Course 204 Summer 2024 Section 3 (Jennia Hizver)

CWE/SANS declared the top 25 most dangerous software weaknesses, 4 selected weaknesses

CWE-787 Out-of-bounds Write

Bjarne Stroustrup¹ once said: "C makes it easy to shoot yourself in the foot; C++ makes it harder, but when you do it blows your whole leg off" [1]. Some programmers regard C/C++ as "high level" (People who code assembler and FORTRAN), others regards it as very "low level" (people who use python and js). Anyway, with languages where you get to directly access memory, there is a danger of putting data where it was not intended. To mitigate this, code must be written carefully so the bits end up in the right place. For example:

- Check length before doing something to assure it is within what you have allocated room for
- use `strncpy()` instead of `strcpy()` (the first has a parameter for length of copied string so you can check it..)
- On the OS, "canaries" (places in memory which can be checked for overwrites) or deploy Address Space Layout Randomization (ASLR) which will reduce the risk of having malicious actors hit something that executes.

CWE-78 Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')

"Never trust user inputs" Whenever something is used as inputs during execution, it must be ensured that creative formatting of the input does not get to run commands on the os. Possible ways are:

- Sanitize input for special characters that may be interpreted as commands on the OS
- Use abstraction, write pre written commands which are then selected based on user inputs - when possible (like using an ORM model for accessing databases)
- ReDoS attacks, where the user inputs are formed to crash the system matching strings using regular expressions, can be mitigated like above and additionally by limiting resources to the process, if excess resource consumption then it is probably malicious.

CWE-22 Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')

Is a variant of CWE 78 above. I remember being able to access 'other interesting departments' folders in Windows at work by using extra `../` (relative path traversal). Mitigation is sanitation of inputs by removing consecutive `../`s and `..` 's and combinations thereof, and proper read-write-execute assignment to folders and files.

CWE-807 Reliance on Untrusted Inputs in a Security Decision

Meaning that the inputs may have been manipulated by someone but are otherwise correct (MAC-spoofing, MITM attacks etc.) Mitigation here is to do something extra on a separate channel, for example 2 factor authentication.

¹ Created C++

“Make Least Privilege a Right (Not a Privilege)”

2.1 What are the five principles of least privilege (POLP) requirements?

1. Compartmentalize: Split applications into smaller protection domains
2. Least Rights: Assign exactly the right privileges to each compartment to complete a task, no more.
3. Interfaces: Engineer communication channels between the compartments
4. Isolate traffic: Ensure that, save for intended communication, the compartments remain isolated from one another
5. Auditable: Make it easy for anyone to audit the intended separation.

2.2 What are the drawbacks of the chroot/jail approach?

The process resembles stuffing an elephant into a taxicab.[2] p1. 2.1

- It breaks don't repeat yourself (DRY) principles, as configurations need to be copied to the jailed areas and maintained
- File and resource sharing is inefficient, it works like a VM and what can talk to what needs to be controlled
- jailing/chroot cannot be used for a lot of executables simultaneously (see above listed) and if bulk executing many similar things (email attachments mentioned) they will all be able to hack each other- sort of ruining the point of jailing in the first place..

2.3 What is one of the main difficulties with ad-hoc privilege separation?

Finding available process and user id's on the host system, maintain the lifecycle of these and ensure that they execute as expected (atomicity).


2.4 How is a capability-based program similar to or different from allowlist?

Assuming a pile of functions, we are to build systems with the right privileges from that pile:

- Capability based system assigns privilege by subtracting the functions that are not allowed
 - Benefits: The system is complete and functional - minus the un-allowed, works well with imposing specific limitations
 - Drawbacks: It's coarse grained, does not work well in complex cases with more than a few functions
- Allowlist based systems assigns only the functions that are allowed
 - Benefits: Granular privileges can be allocated, custom build for each case, and you are sure that no extra privileges are granted
 - Drawbacks: Required build of complete systems for each case including all 'auxiliary' functions

“Memory corruption mitigation via hardening and testing”

3.1 Briefly describe the four exploit avenues mentioned in the paper.

- Code Corruption: We (in this case) assume that compiled code is safe (code integrity ) , however for interpreted languages, code is compiled at runtime (just in time, JIT). Here it is possible that code is changed and code integrity cannot be ensured
- Control-flow Hijack: Manipulate a pointer to go somewhere else and maybe to execute shellcode pawning the machine, pointers typically manipulated by utilizing buffer overflow conditions.
- Data Corruption: Much like Control-Flow, but instead of pointing to a place with malicious code, some data already there is manipulated to do what the attacker wants, an example is: encrypting the whole hard drive in a ransomware attack.

- Information Disclosure: Leaking information, and example is out-of-bounds read, where whatever follows the allowed in memory is also outputted.

3.2 Why code integrity cannot be fully enforced by browsers using Just-In-Time compilation?

Page 13 [3]: "code integrity cannot be fully enforced because there is a time window during which the generated code is on a writable page."

3.3 Describe each step in the control-flow hijack exploit. For each step, discuss the mechanisms which could either detect or prevent the step.

Freely based in figure 2.1 in [3]

1. Get hold of a pointer (Out of bounds)
2. Use pointer to write
3. Modify pointer -
4. To go to an address with malicious code
5. And return the value to somewhere that executes
6. Target pawned

3.4 Why data corruption attacks are called non-control-data attacks? Give one example of a non-control-data attack involving user identity data.

I guess because the attack manipulates data already in place, An example is corrupting file read/write/executable for privileges escalation.

3.5 What are the most widely deployed protection mechanisms against memory corruption attacks?

Memory manipulation is mitigated by:

- Canaries, where a pointer has its begin and end addresses stored with it as metadata - and therefore can be checked. This requires resources and can only be done for a limited number of pointers.
- A scheme to randomize where executable code is located in memory, Address space layout randomization (ASLR) does not completely remove the possibility of malicious code hitting the target in memory, but it reduces the risk.

3.6 Beside security, what is the most important requirement for protection mechanisms as stated by the paper? How does this requirement affect gaining wide adoption in production environments?

Beside security, the most important requirement is speed [3], p.19, section 2.3.2. Speed is extremely important. If there is a speed gain in a technology it will be adopted, contrarily if it slows things down then overwhelming reasons need to be present for it to be introduced.

3.7 What is the novel protection mechanism proposed by the author to prevent control-flow hijack

Code Pointer Integrity (CPI) where important pointers and functions are identified (static analysis) and monitored (instrumentation) during execution.

References

- [1] *Bjarne Stroustrup website*. <https://www.stroustrup.com/quotes.html>. Accessed: 2024-7-10.
- [2] *Krohn et.al. Make Least Privilege a Right (Not a Privilege)*. <https://bcourses.berkeley.edu/courses/1534708/files/88731104?wrap=1>. Accessed: 2024-7-12.
- [3] *László Szekeres phd dissertation*. <https://bcourses.berkeley.edu/courses/1534708/files/88731075?wrap=1>. Accessed: 2024-7-12.