Towards Enhanced Dynamic Analysis and Automated Testing

or "How to Be Productively Lazy"

Vissarion Moutafis

15-12-2025

whoami

Security Engineer; PhD Researcher @ TU Delft; ex-sdi1800119 ;-)

vissarionmouta@tudelft.nl

https://www.linkedin.com/in/vissarion-moutafis-843947192/

BSc @ DIT; MSc @ TU Delft; Ex Sec-Engineer @ CENSUS; PhD Candidate @ TU Delft;

Interested in Systems {Design, Development, Security}, Application Security & IoT.

Agenda

00

Fuzzing Intro

01

Protocol Fuzzing

02

Current Work in Progress

03

System Calls in Fuzzing

04

Conclusion and Q&A

00

Fuzzing

What? How? Why?

Program Testing

Static Analysis

- No execution
- CFG Analysis
- Symbolic Reasoning
- No runtime info

Dynamic analysis

- Execution-based
- Runtime Interaction
- Needs initial seed
- Low hanging bugs

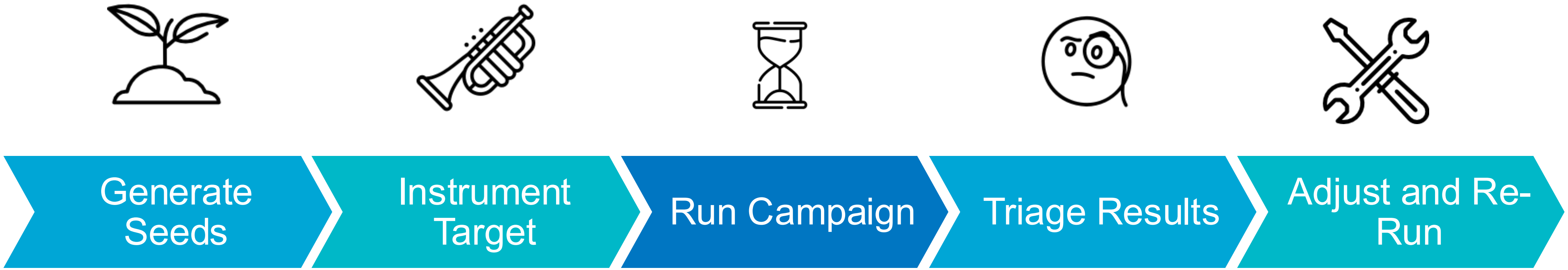
Fuzzing is the practice of automating testcase generation for maximizing code coverage (which ultimately results in bug discovery)

System Under Test or SUT is the application that will be tested against our tool (and will eventually break)

An interaction between the fuzzer and the SUT interprets one “testing” iteration.

Fuzzing is a form of dynamic analysis.

Process | 5 steps



Generate Seeds

Instrument Target

Run Campaign

Triage Results

Adjust and Re-Run

Seeds
Starting Seeds are the first part of generating diverse high-quality testcases

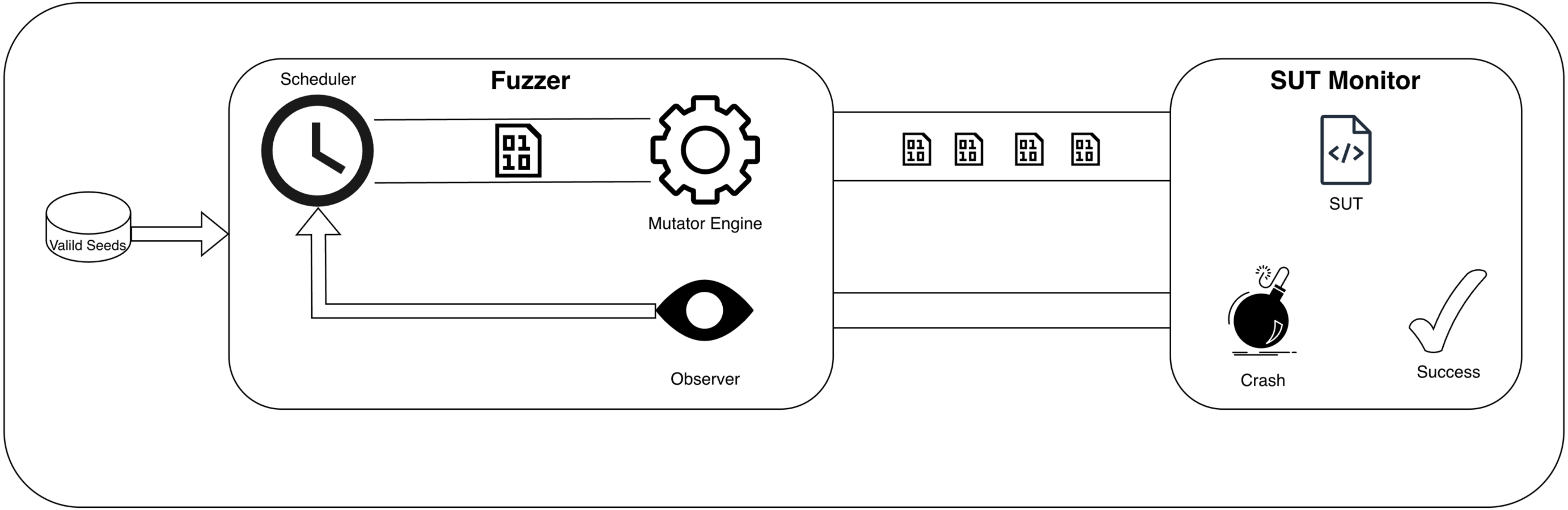
Harness and Branch Monitor
Harness routine to send input; Compile-time instrumentation

24-hr Campaigns
The fuzzer will generate random testcases, score them and repeat

Manual Work
Yay!

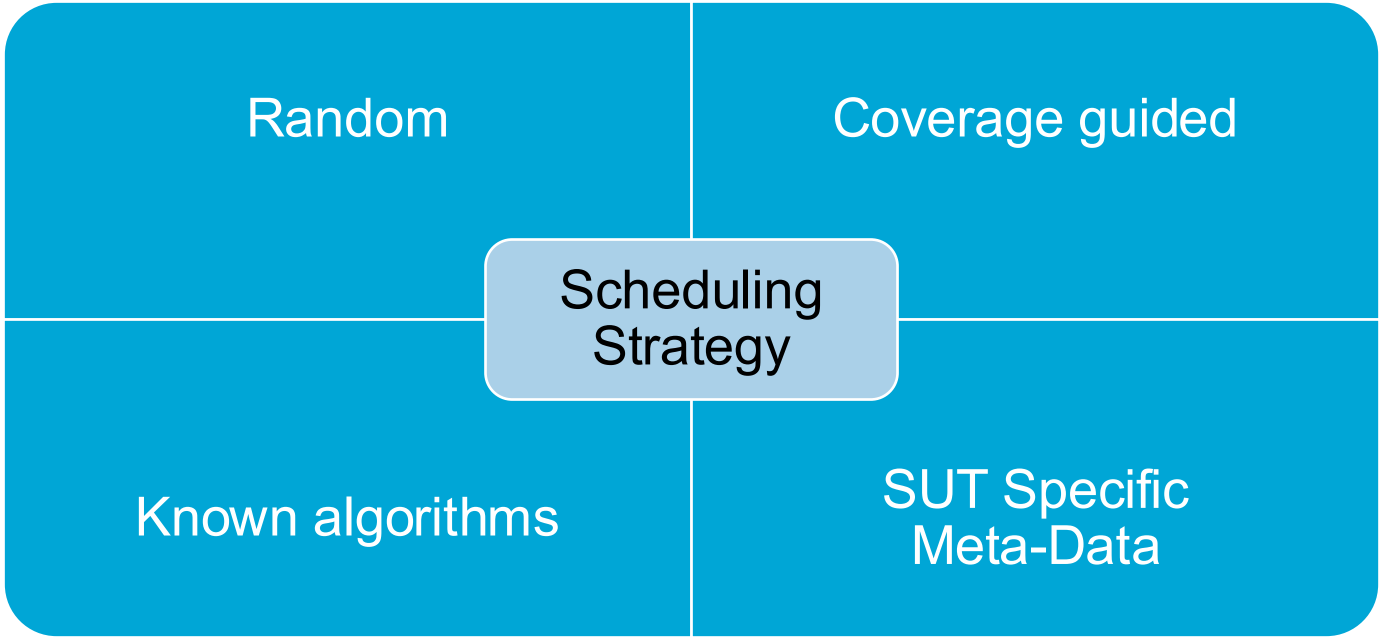
So what?
Did we cover important paths? Any bugs? Adjust for increasing testcase quality.

High Level Design



The Scheduler

Main Job: Pick Next seed to mutate



Scheduling Strategy

Random

Coverage guided

Known algorithms

SUT Specific Meta-Data

Mutations

Bit/Byte Flips

- Fast
- Universal
- Random

Arithmetic

- Fast
- Might flip numeric conditions
- Bounds Errors

Insert/Delete

- Bound Checking robustness
- Random or pattern based

Overwrite

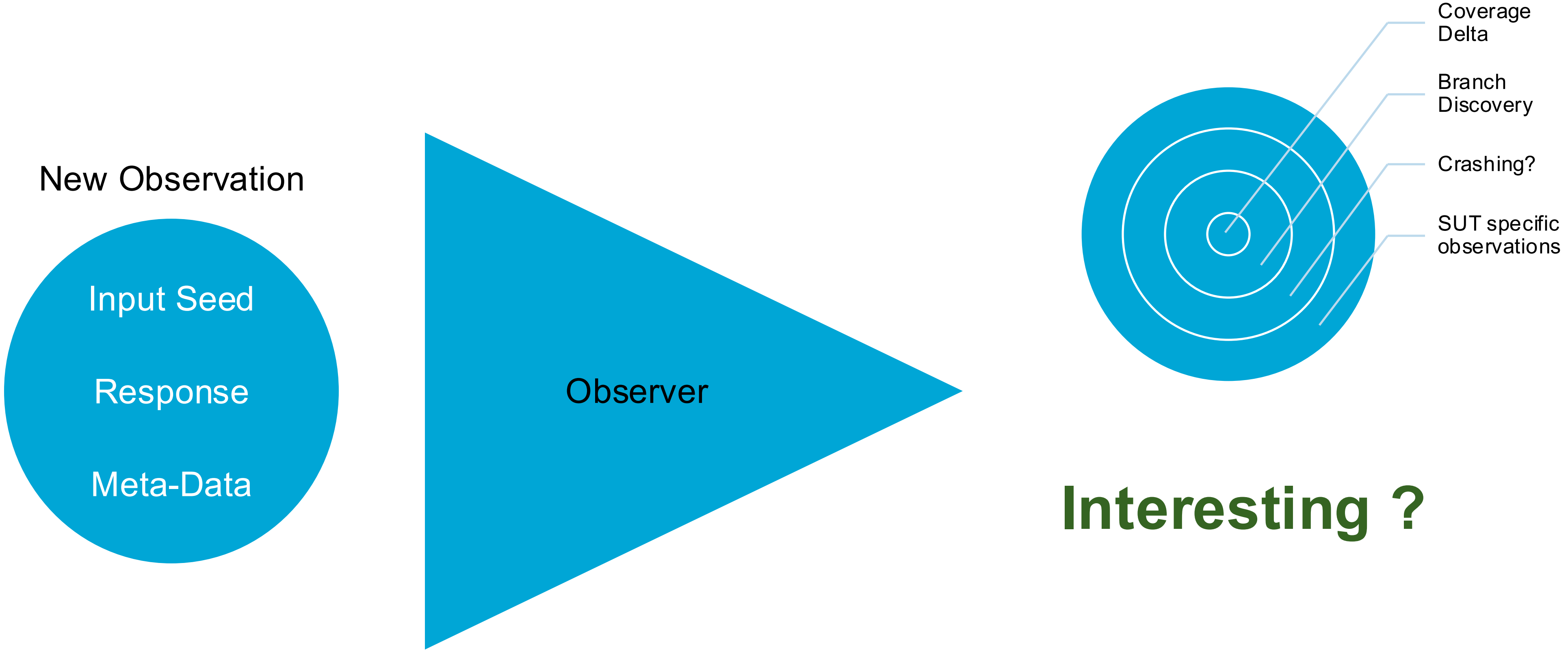
- Change parts of the execution path
- More Intrusive

Special Values

- 0x00, 0xFF, etc.
- Aiming for edge cases

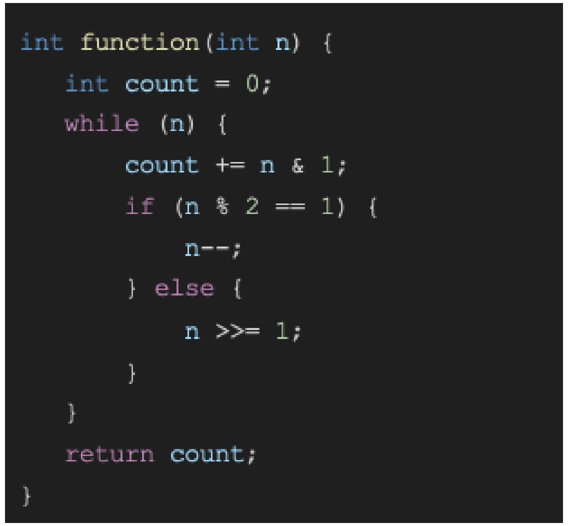
The Observer

Main Goal: Parse Observations, Score Mutation, Update Internal Observations

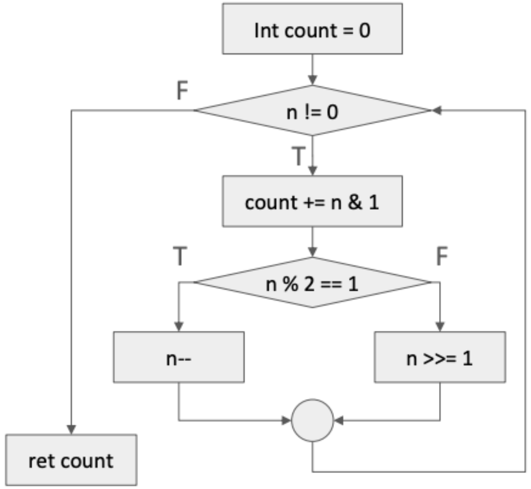


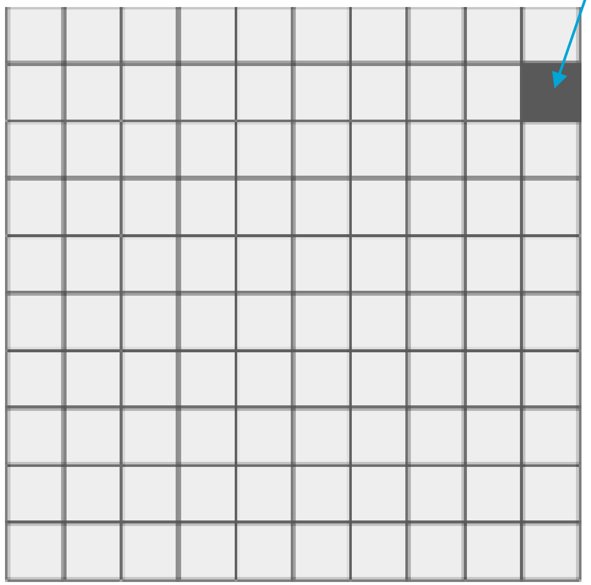
Interesting ?

CFGs and Coverage



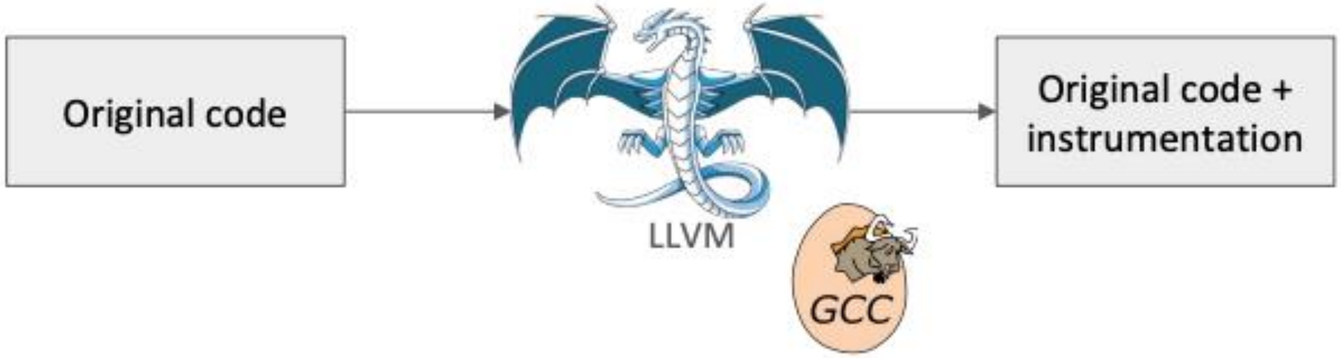
```
int function(int n) {
   int count = 0;
   while (n) {
       count += n & 1;
       if (n % 2 == 1) {
           n--;
       } else {
           n >>= 1;
       }
   }
   return count;
}
```





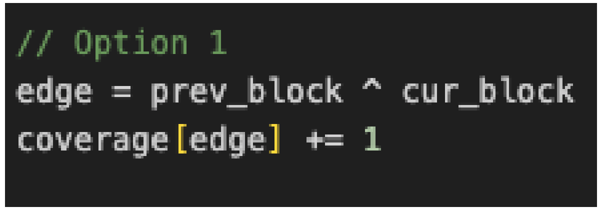
Every edge of the CFG map is an item in the coverage map

How to implement coverage

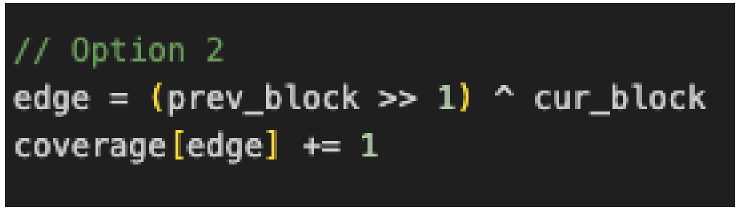


After instrumenting the coverage we can use a **simple uint[] map** to interpret coverage. This achieves a sufficient way to keep track of a current execution path, then compare with all our observations and based on the difference evaluate the input mutation.

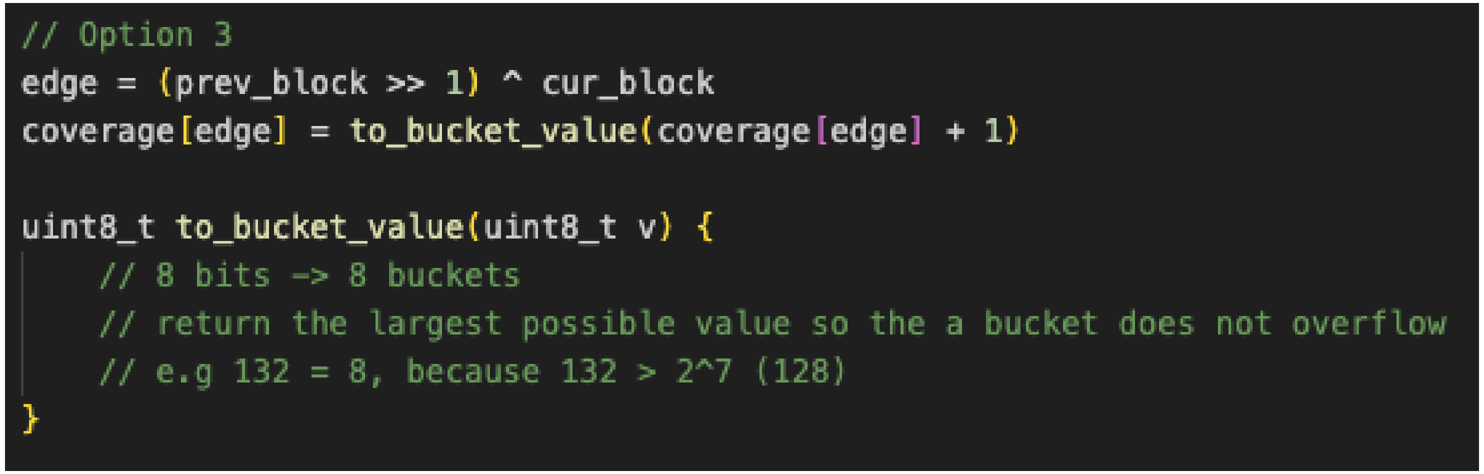
Quiz Time: Edge-based coverage interpretation



```
// Option 1
edge = prev_block ^ cur_block
coverage[edge] += 1
```



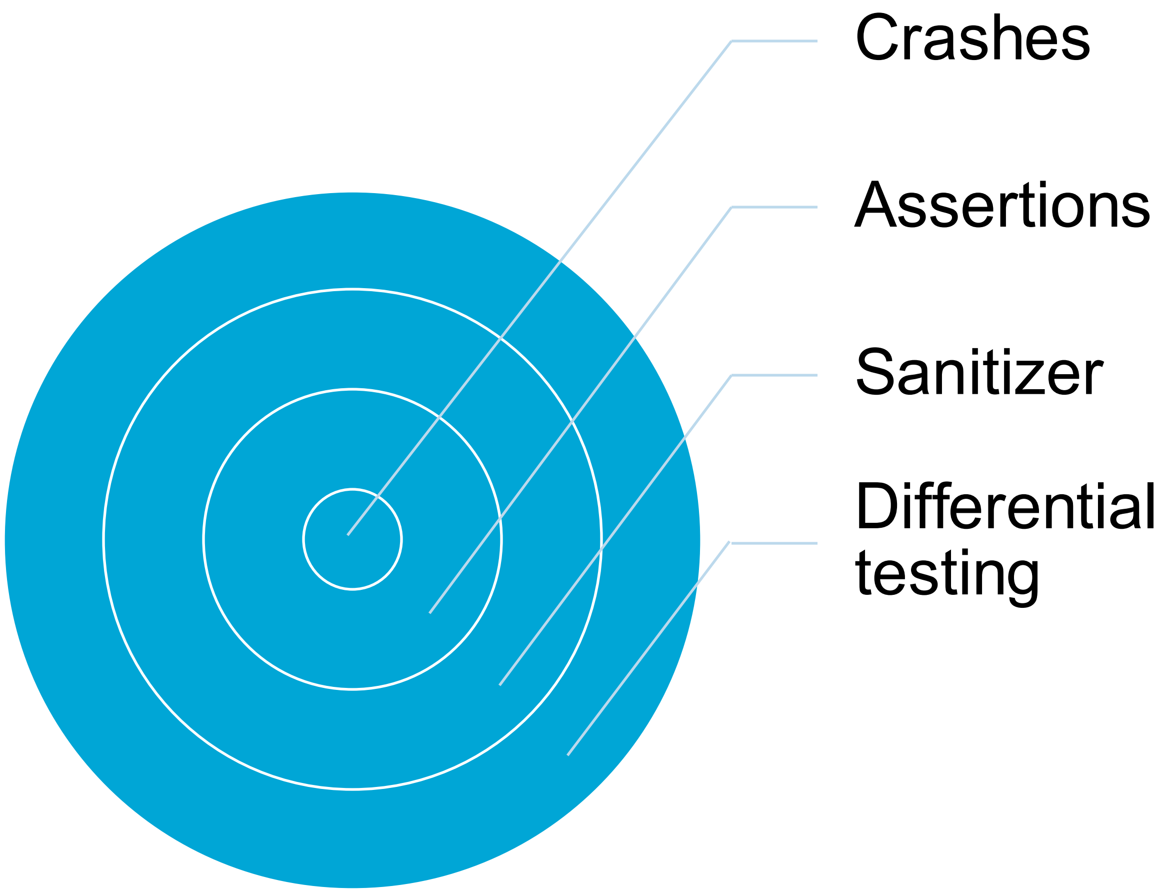
```
// Option 2
edge = (prev_block >> 1) ^ cur_block
coverage[edge] += 1
```



```
// Option 3
edge = (prev_block >> 1) ^ cur_block
coverage[edge] = to_bucket_value(coverage[edge] + 1)

uint8_t to_bucket_value(uint8_t v) {
    // 8 bits -> 8 buckets
    // return the largest possible value so the a bucket does not overflow
    // e.g 132 = 8, because 132 > 2^7 (128)
}
```

How we monitor the SUT?



Crashes

Assertions

Sanitizer

Differential testing

Quiz: How would you implement the SUT Manager/Monitor?

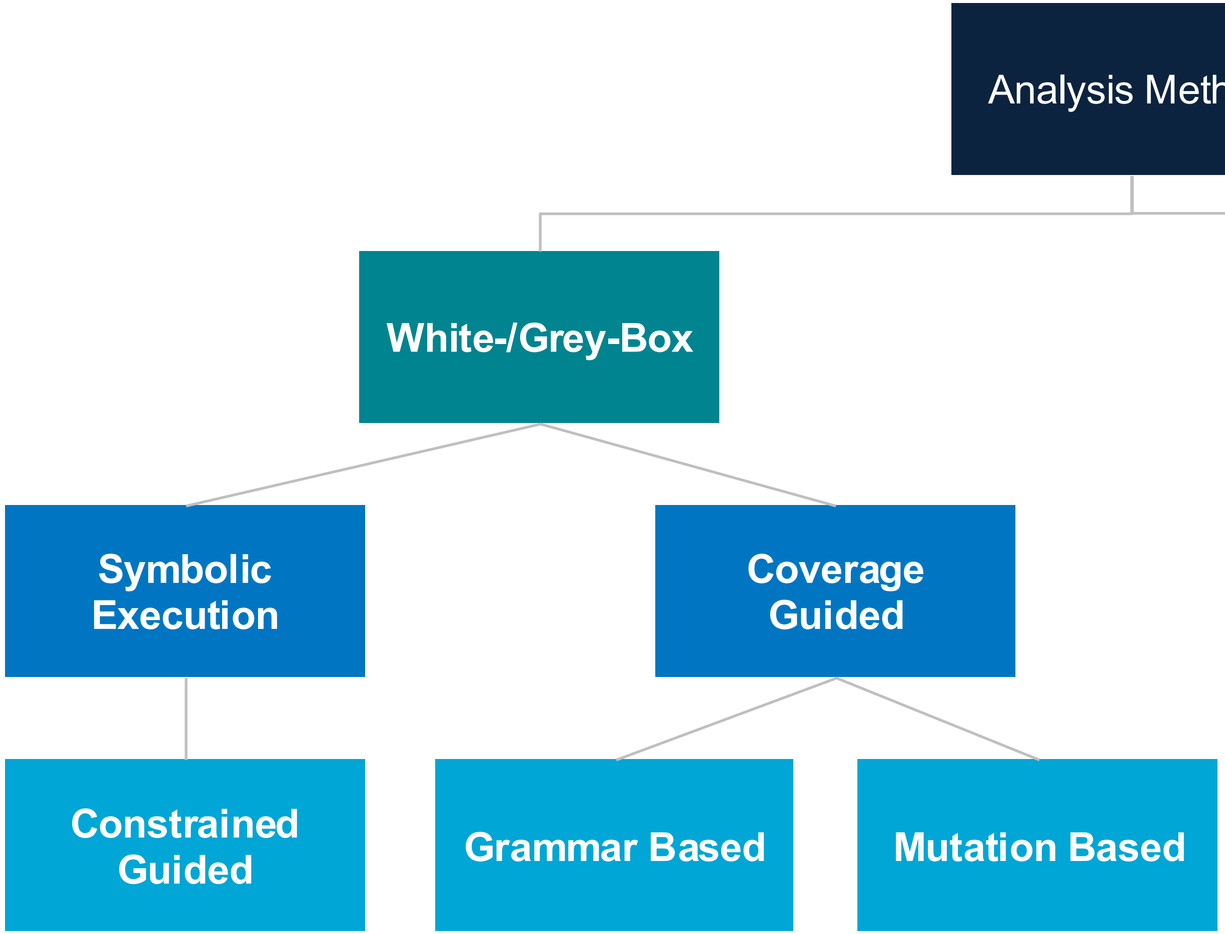
Common SUT Manager implementations

- **Forkserver**: Spawn a parent, init communication channels, on RUN command, fork and exec
- **Persistent mode**: Run the target in loop within a single process, reset between iterations
- **Snapshot based**: Snap/Restore process/system information before running a testcase
- **Parallel/Distributed**: Multiple SUTs run at the same time, increased throughput, might hit OS bottlenecks
- **Hybrids**

The forksever architecture is the most common for popular commercial fuzzers (e.g. AFL++ family).

The key for quality guided fuzzing is the quality of observations. But then..

Recap and then again…



Analysis Method

White-/Grey-Box

Black Box

Symbolic Execution

Coverage Guided

Constrained Guided

Grammar Based

Mutation Based

Q: What can we use for coverage inference when we only have a stripped binary ?

A: Any observable behavior !!

- Return codes (Busch et al (2023))
- Server Status (Pham et al. (2020))
- Output behavior (e.g. written files, stdout, etc.) (LABRADOR: Response Guided Directed Fuzzing for Black-box IoT Devices, S&P ’24)
- Syscalls Traces (Xiao et al.)
- Execution time
- .. and anything else that can indicate change in execution path

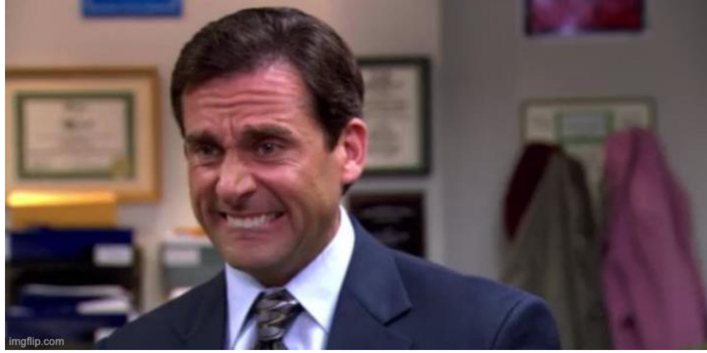
What about stateful SUTs?

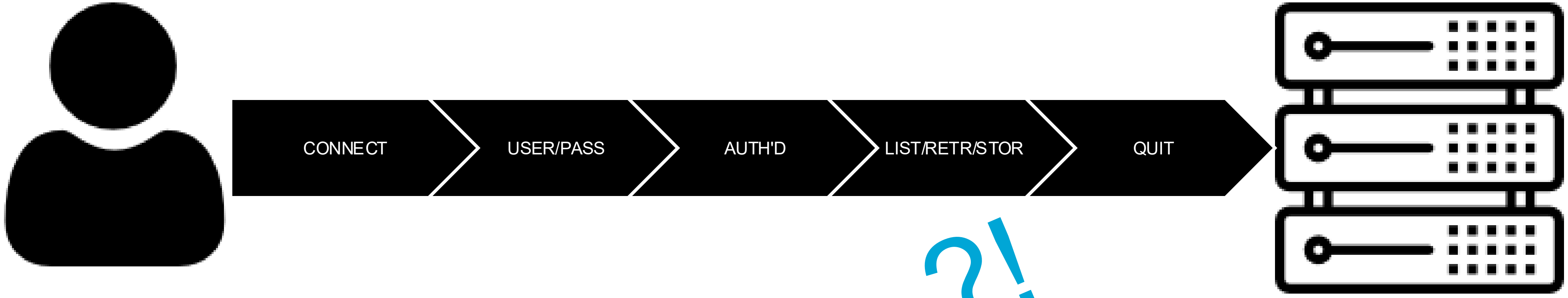
01

Network Protocol Fuzzing

Stateful Black Boxes

Example: FTP Server Fuzzing





Solution ?!

State Awareness

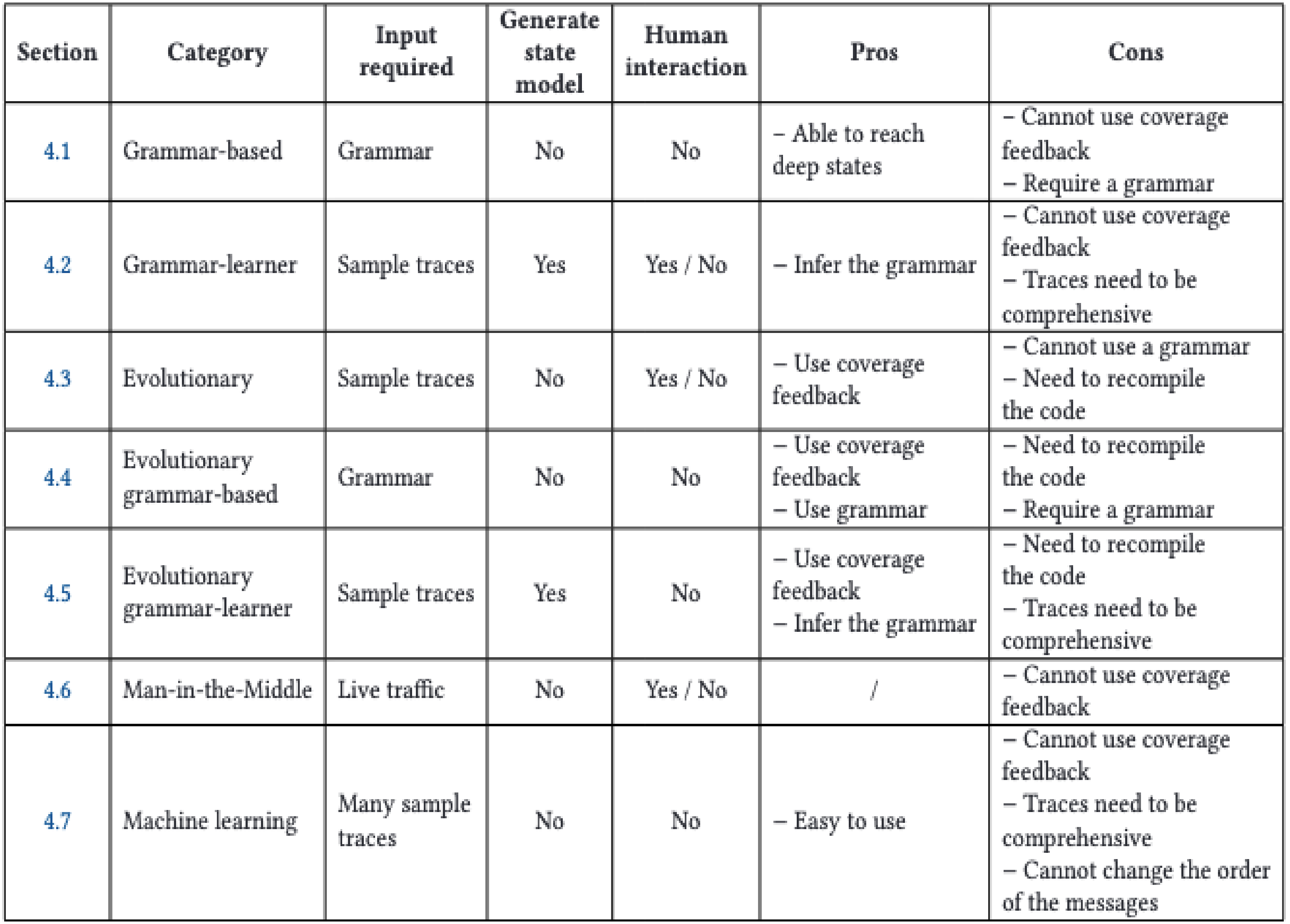
USER
CONTEXT

SESSION
META-DATA

CRYPTOGRAPHIC
MATERIAL

Guide execution based on the *hidden state machine* of the target implementation.

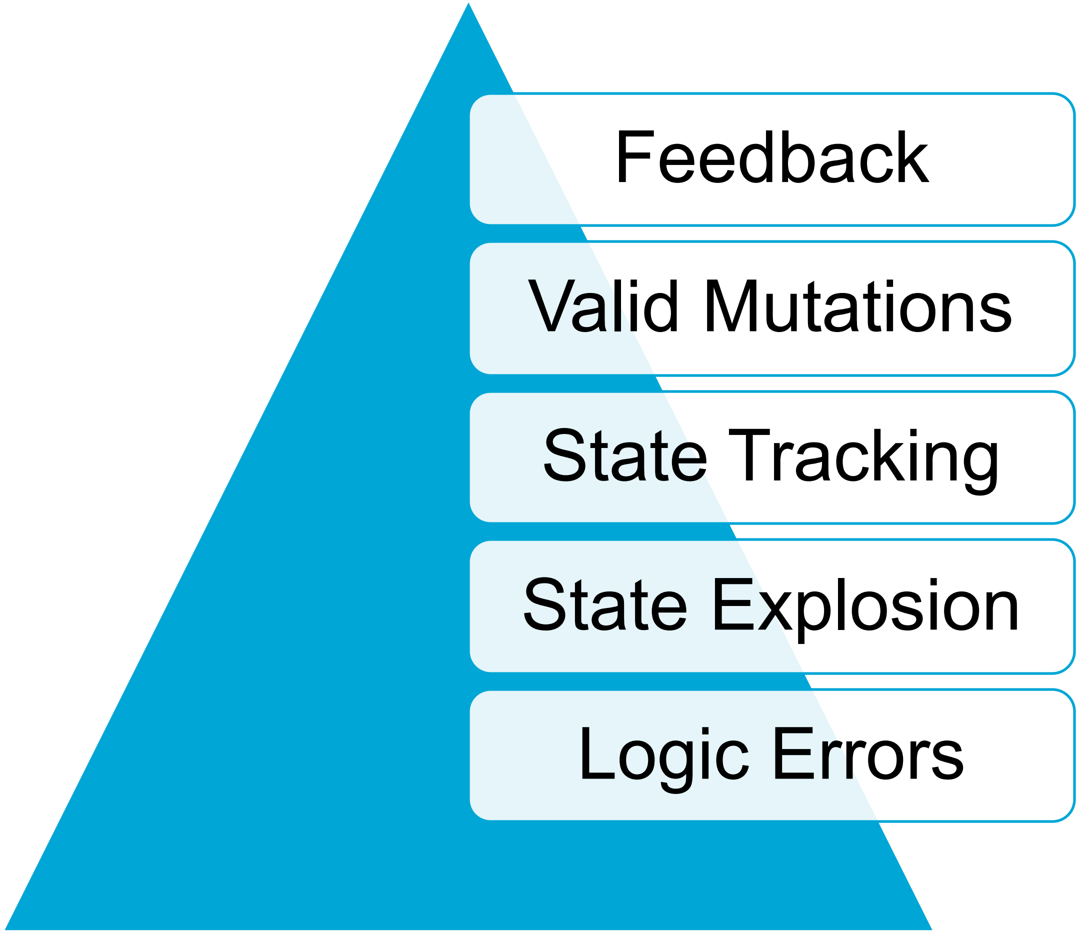
Existent Approaches



Section	Category	Input required	Generate state model	Human interaction	Pros	Cons
4.1	Grammar-based	Grammar	No	No	– Able to reach deep states	– Cannot use coverage feedback – Require a grammar
4.2	Grammar-learner	Sample traces	Yes	Yes / No	– Infer the grammar	– Cannot use coverage feedback – Traces need to be comprehensive
4.3	Evolutionary	Sample traces	No	Yes / No	– Use coverage feedback	– Cannot use a grammar – Need to recompile the code
4.4	Evolutionary grammar-based	Grammar	No	No	– Use coverage feedback – Use grammar	– Need to recompile the code – Require a grammar
4.5	Evolutionary grammar-learner	Sample traces	Yes	No	– Use coverage feedback – Infer the grammar	– Need to recompile the code – Traces need to be comprehensive
4.6	Man-in-the-Middle	Live traffic	No	Yes / No	/	– Cannot use coverage feedback
4.7	Machine learning	Many sample traces	No	No	– Easy to use	– Cannot use coverage feedback – Traces need to be comprehensive – Cannot change the order of the messages

Daniele, C., Andarzian, S. B., & Poll, E. (2024). Fuzzers for Stateful Systems: Survey and Research Directions. *ACM Computing Surveys*, *56*(9), 1–23. https://doi.org/10.1145/3648468

Black-Box Stateful Fuzzing Challenges - Recap



Feedback

Valid Mutations

State Tracking

State Explosion

Logic Errors

15-20’ Break!?

02

Star Wars: The Fuzz Wars

Automating Space Protocol Testing

Space Mission Architecture – SDL Protocol

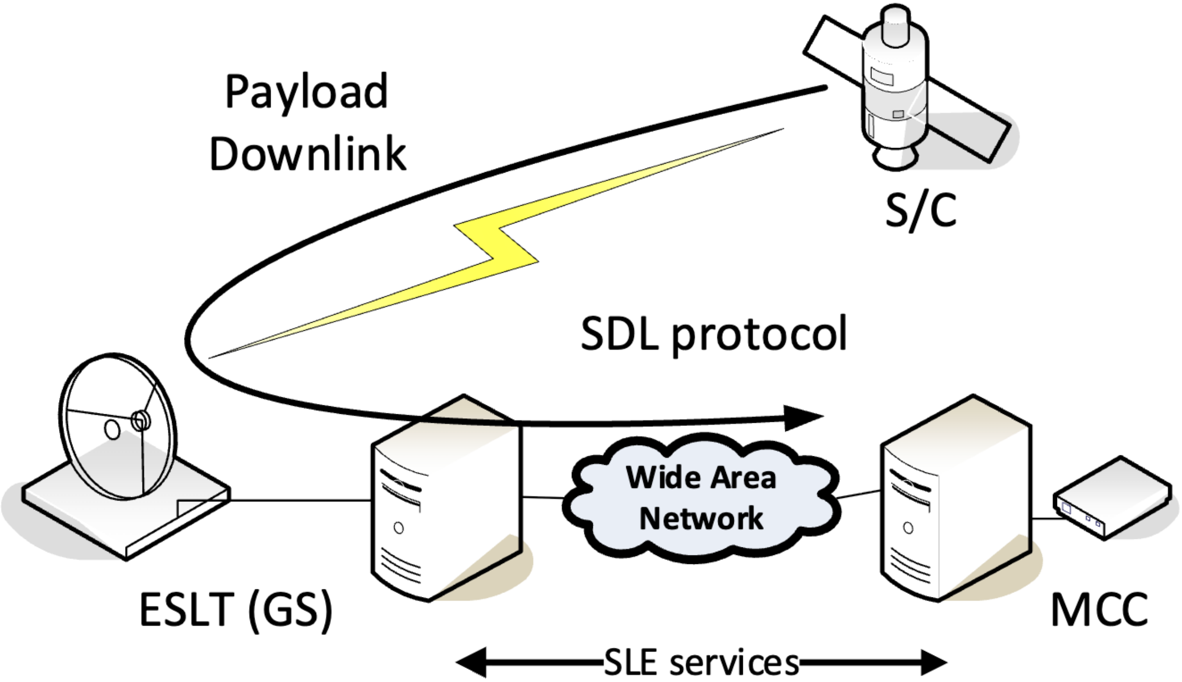


Figure 2-2: Mission Network Topology A

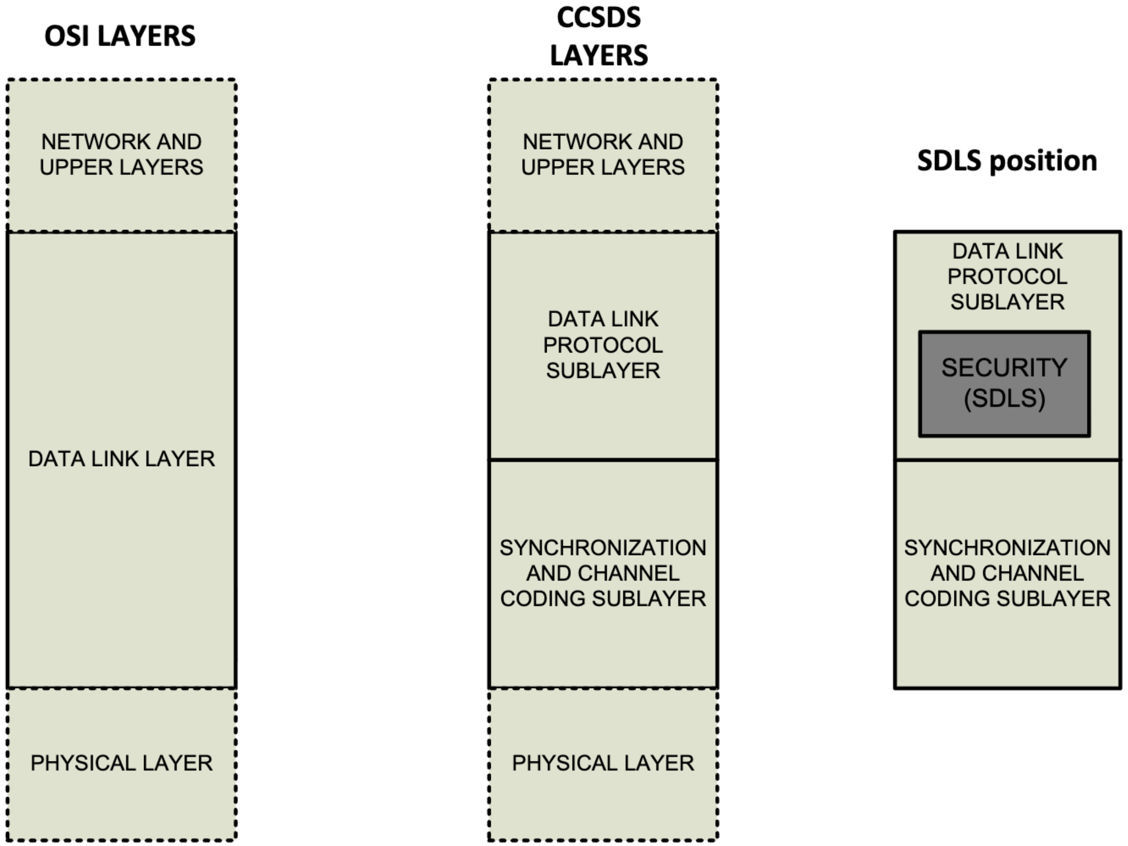
CCSDS 350.5-G-2 (2024), https://ccsds.org/publications/greenbooks/entry/3257/

Space Data Link Protocols (TC/TM/AOS)

- **TC (Telecommand):** Carries control commands from ground to spacecraft, often triggering state changes and safety-critical actions onboard.
- **TM (Telemetry):** Transports monitoring and status data from spacecraft to ground, reflecting internal system and mission state.
- **AOS (Advanced Orbiting Systems):** Provides a high-throughput, virtual-channel based data link protocol for complex space missions and onboard networks.

- They do not define security measures, meaning that traffic is unencrypted and easy to tamper with…
- How to solve that? Make a Security Wrapper

Space Data Link Security Protocol (1/4) - Placement

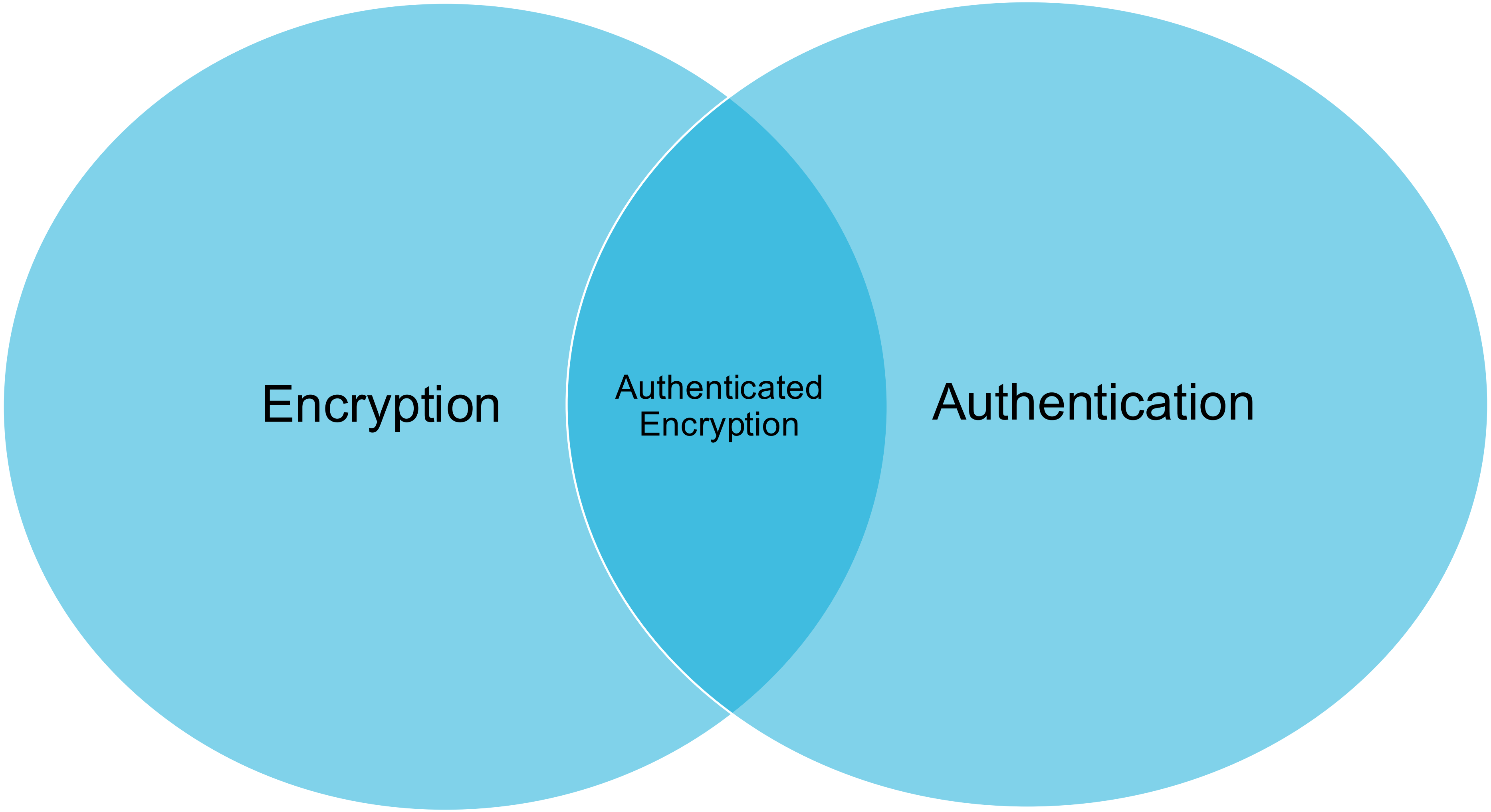


CCSDS 350.5-G-2 , (2024), https://ccsds.org/publications/greenbooks/entry/3257/

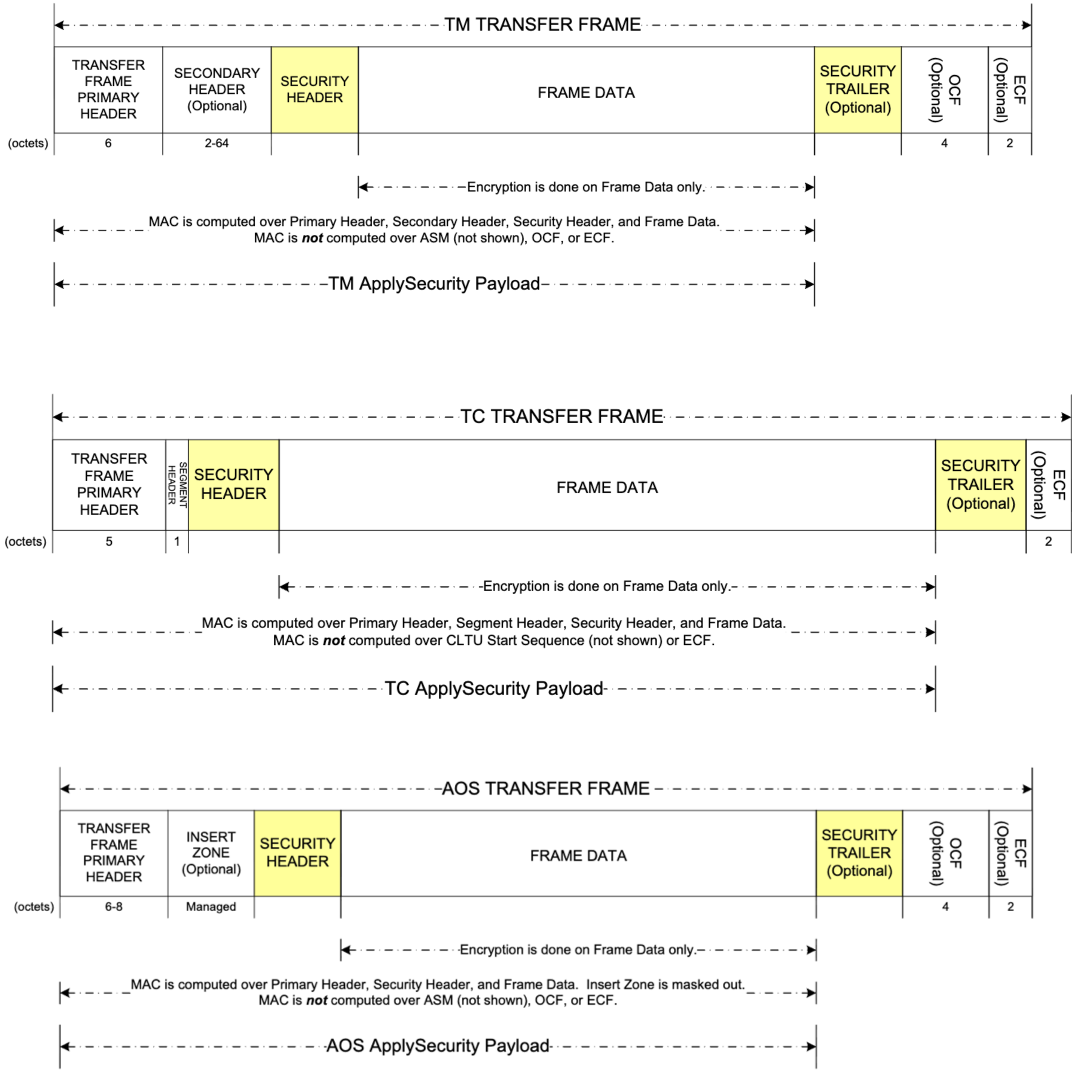
Space Data Link Security Protocol (2/4) - Ops

- 2 Important interfaces
  - Process Security
  - Apply Security
- Managed parameters per channel
- Crypto parameters and Storages are defined statically
- Session keys can be changes during mission
- 3 modes of operation

Space Data Link Security Protocol (3/4) - Ops



Space Data Link Security Protocol (4/4)



TM TRANSFER FRAME

	TRANSFER FRAME PRIMARY HEADER	SECONDARY HEADER (Optional)	SECURITY HEADER	FRAME DATA	SECURITY TRAILER (Optional)	OCF (Optional)	ECF (Optional)
(octets)	6	2-64				4	2

Encryption is done on Frame Data only.

MAC is computed over Primary Header, Secondary Header, Security Header, and Frame Data.
MAC is ***not*** computed over ASM (not shown), OCF, or ECF.

TM ApplySecurity Payload

TC TRANSFER FRAME

	TRANSFER FRAME PRIMARY HEADER	SEGMENT HEADER	SECURITY HEADER	FRAME DATA	SECURITY TRAILER (Optional)	ECF (Optional)
(octets)	5	1				2

Encryption is done on Frame Data only.

MAC is computed over Primary Header, Segment Header, Security Header, and Frame Data.
MAC is ***not*** computed over CLTU Start Sequence (not shown) or ECF.

TC ApplySecurity Payload

AOS TRANSFER FRAME

	TRANSFER FRAME PRIMARY HEADER	INSERT ZONE (Optional)	SECURITY HEADER	FRAME DATA	SECURITY TRAILER (Optional)	OCF (Optional)	ECF (Optional)
(octets)	6-8	Managed				4	2

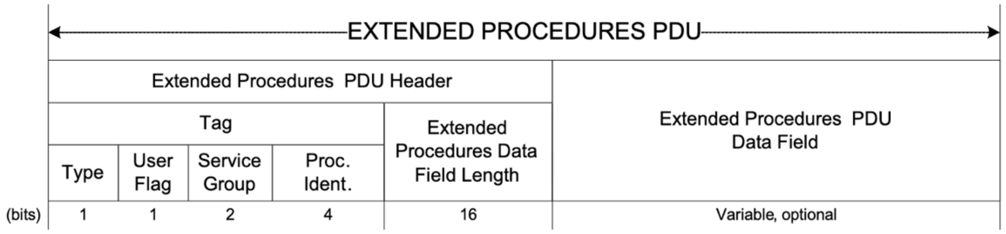
Encryption is done on Frame Data only.

MAC is computed over Primary Header, Security Header, and Frame Data. Insert Zone is masked out.
MAC is ***not*** computed over ASM (not shown), OCF, or ECF.

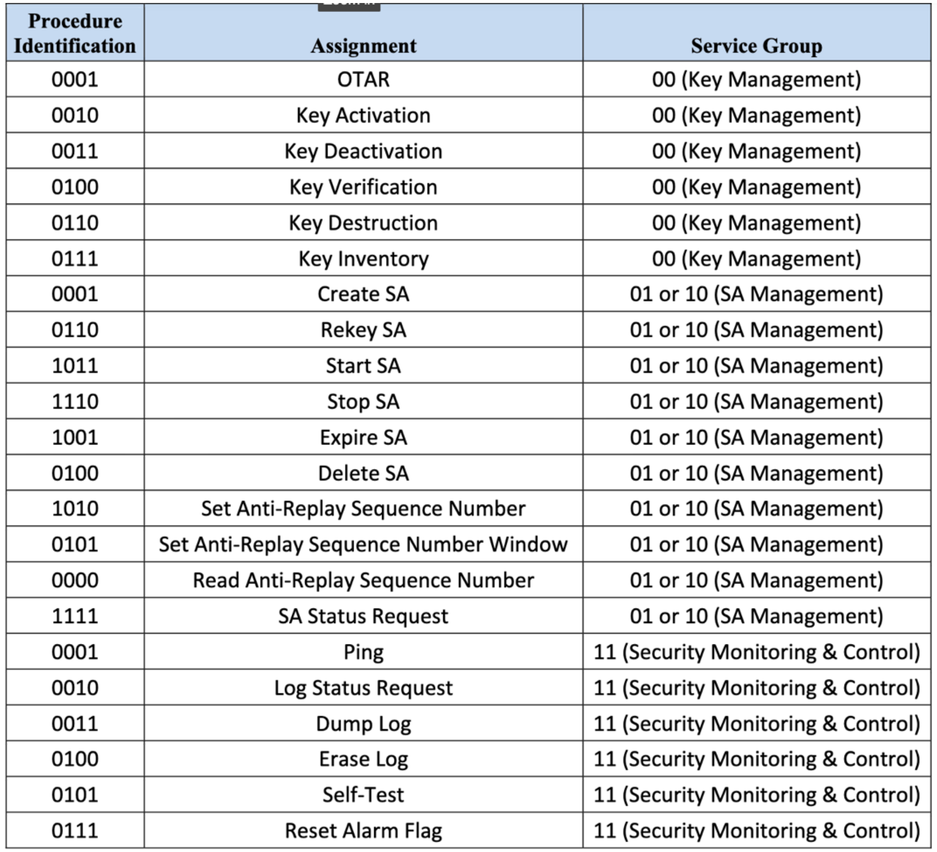
AOS ApplySecurity Payload

CCSDS 355.0-B-2, (2022), https://ccsds.org/publications/bluebooks/entry/3258/

SDLS EP (1/3) – Commands and Tags



	EXTENDED PROCEDURES PDU					
	Extended Procedures PDU Header					Extended Procedures PDU Data Field
	Tag				Extended Procedures Data Field Length	
	Type	User Flag	Service Group	Proc. Ident.		
(bits)	1	1	2	4	16	Variable, optional

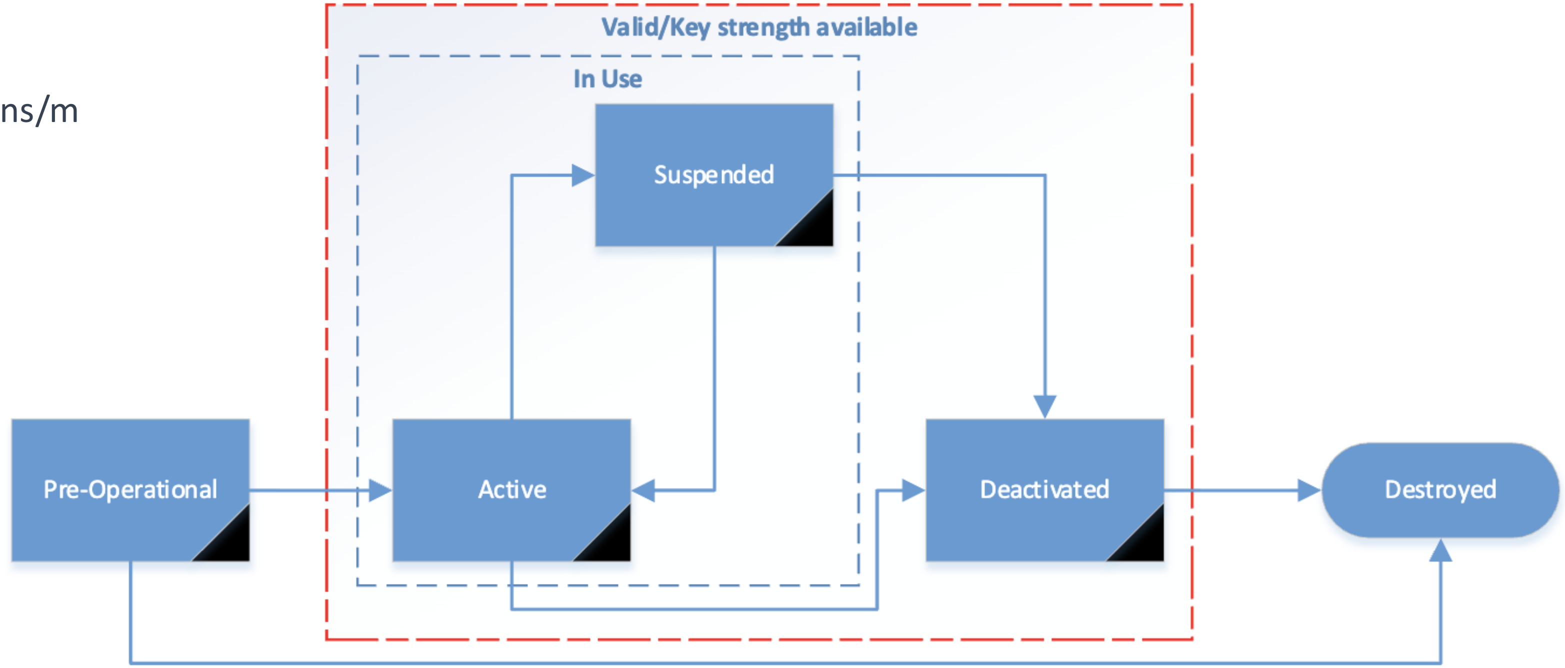


Procedure Identification	Assignment	Service Group
0001	OTAR	00 (Key Management)
0010	Key Activation	00 (Key Management)
0011	Key Deactivation	00 (Key Management)
0100	Key Verification	00 (Key Management)
0110	Key Destruction	00 (Key Management)
0111	Key Inventory	00 (Key Management)
0001	Create SA	01 or 10 (SA Management)
0110	Rekey SA	01 or 10 (SA Management)
1011	Start SA	01 or 10 (SA Management)
1110	Stop SA	01 or 10 (SA Management)
1001	Expire SA	01 or 10 (SA Management)
0100	Delete SA	01 or 10 (SA Management)
1010	Set Anti-Replay Sequence Number	01 or 10 (SA Management)
0101	Set Anti-Replay Sequence Number Window	01 or 10 (SA Management)
0000	Read Anti-Replay Sequence Number	01 or 10 (SA Management)
1111	SA Status Request	01 or 10 (SA Management)
0001	Ping	11 (Security Monitoring & Control)
0010	Log Status Request	11 (Security Monitoring & Control)
0011	Dump Log	11 (Security Monitoring & Control)
0100	Erase Log	11 (Security Monitoring & Control)
0101	Self-Test	11 (Security Monitoring & Control)
0111	Reset Alarm Flag	11 (Security Monitoring & Control)

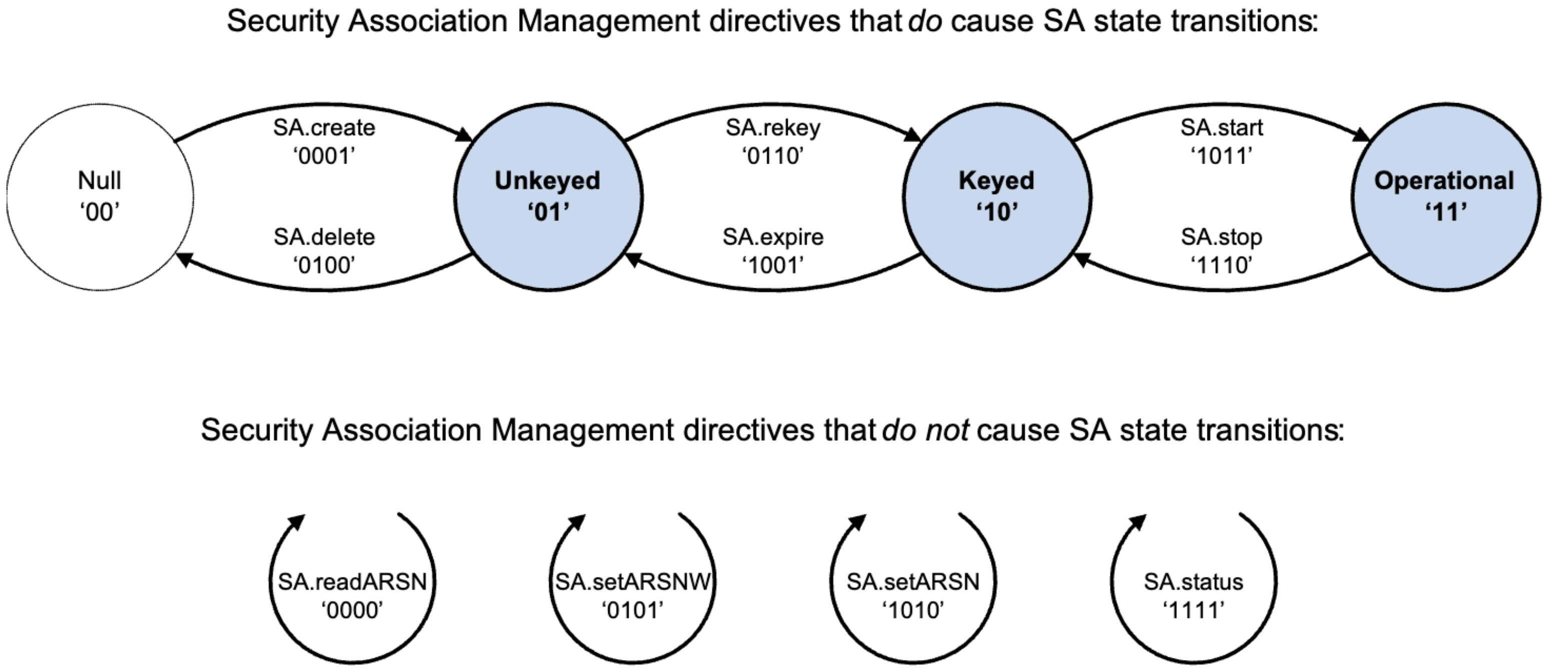
CCSDS 355.1-B-1, (2020), https://ccsds.org/publications/bluebooks/entry/3259/

SDLS EP (2/3) – Key Management

CCSDS 354.0-M-1, (2023), https://ccsds.org/publications/magentabooks/entry/3142/



SDLS EP (2/3) – SA Management



CCSDS 355.0-B-2, (2022), https://ccsds.org/publications/bluebooks/entry/3258/

Problem 1: Grammar is very specific
Problem 2: What can we observe?
Problem 3: Conformance issues ?
Problem 4: State representation ?

Testing Mode: Black Box access to crypto APIs that can send messages to the spacecraft; Crypto Material already configured.

Mutations: Bit Masks

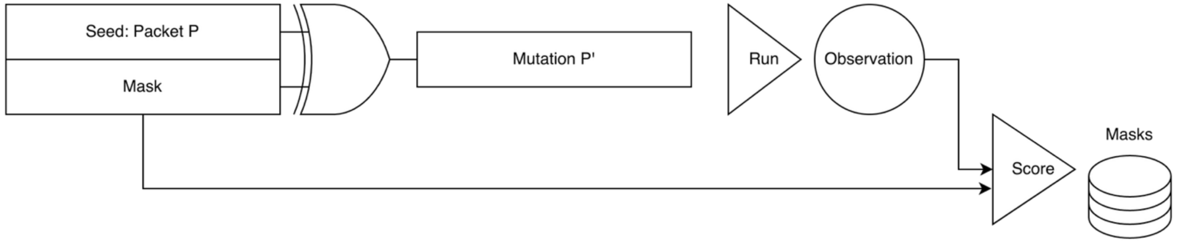
1. M = GetMask(M.Length)
2. P’ = P ^ M (Problems?)
3. Observation(P’) = {SUT(P’), P, M, State}
4. Score the mask based on the observation (Ideas ?)
5. Update and adjust mask based on score
6. Goto 1.

Step 4: What do we care about when scoring a “mutation” not the whole RUN

Step 5: Either store the whole mask, or update interest weights of flipping the n’th bit/byte.

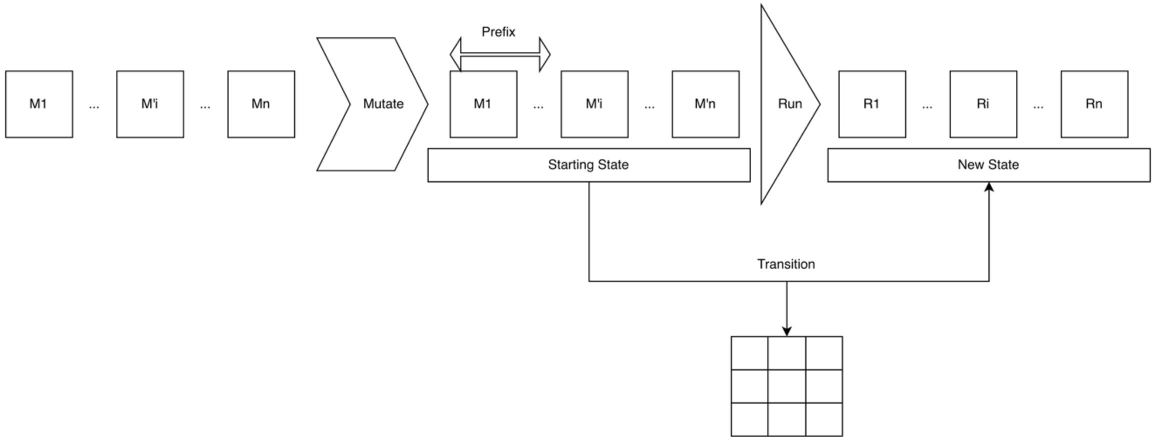
- What is the issue with the first?

- How can we improve the second from



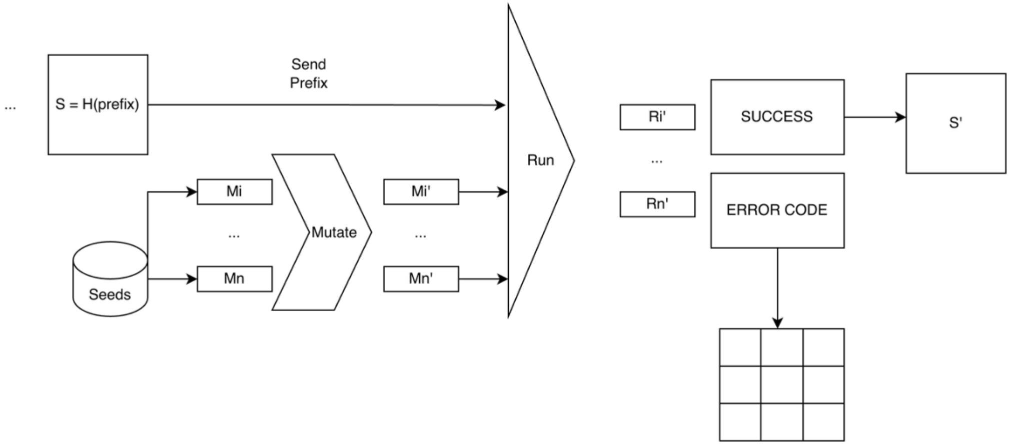
Two different State Exploration strategies

Random Walk



1. Use prefix to reach a state
2. Sequence and Message Specific Mutations
3. State = {m, where response(m) is SUCCESS}
4. Global Coverage = State transition

BFS with a twist



1. Use prefix to reach a state
2. Until possible status codes die out; do
    1. Mutate message M
    2. If response Ri’ is success, possible new state
    3. Otherwise, update coverage
3. State = Prefix to reach this point
4. Global Coverage = State transitions
5. (Local) Per State Coverage = possible error codes

Can we do better than tracking a binary response ?

Protocol Domain awareness

- We chose to fuzz the library as software, so we can use the API return codes
- In real missions, it depends of whether we will get such responses back from the spacecraft
- What would be a nice idea to monitor in such case?
  - State of storages (Key and SA storages)
  - There are status yield commands that could enable that
  - Cons: Non generic, + per iteration overhead, extra final message required

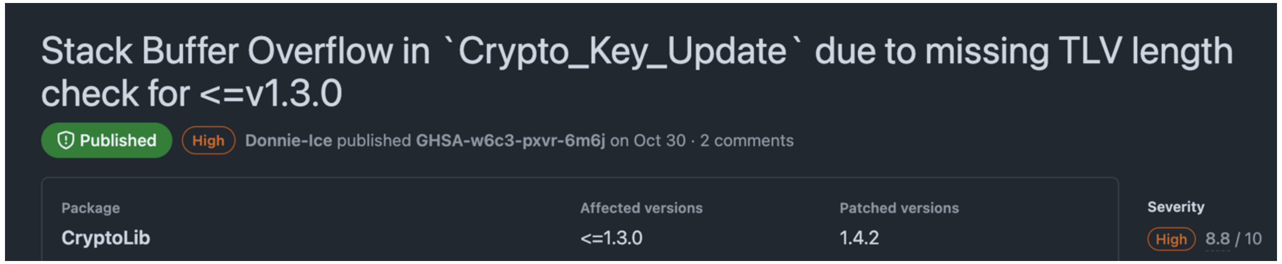
Conformance Testing – Model Based Fuzzing

- Incremental implementation of the SDLS and SDLS EP standard
- Use a safe memory language to avoid undetected issues of C
- Online response mismatch detection
- Integrate mismatches to scoring to avoid triggering previous bugs
- Extra Pros:
    - Easier to fuzz un-encrypted packets, less early stage faults
    - Partial development is fast and does not require the whole protocol to be implemented

Preliminary Results

- ✓ Found most of the previously reported bugs automatically
- ✓ 1 High Impact CVE on NASA's open source implementation of SDLS + EP
- ✓ Around 38% total library coverage

We are still in beta phase so more concrete data analysis and comparison with state of the art is needed.



Stack Buffer Overflow in `Crypto_Key_Update` due to missing TLV length check for <=v1.3.0

Published · High · Donnie-Ice published GHSA-w6c3-pxvr-6m6j on Oct 30 · 2 comments

Package	Affected versions	Patched versions	Severity
CryptoLib	<=1.3.0	1.4.2	High 8.8 / 10

https://github.com/nasa/CryptoLib/security/advisories/GHSA-w6c3-pxvr-6m6j

03

Syscall Interposition

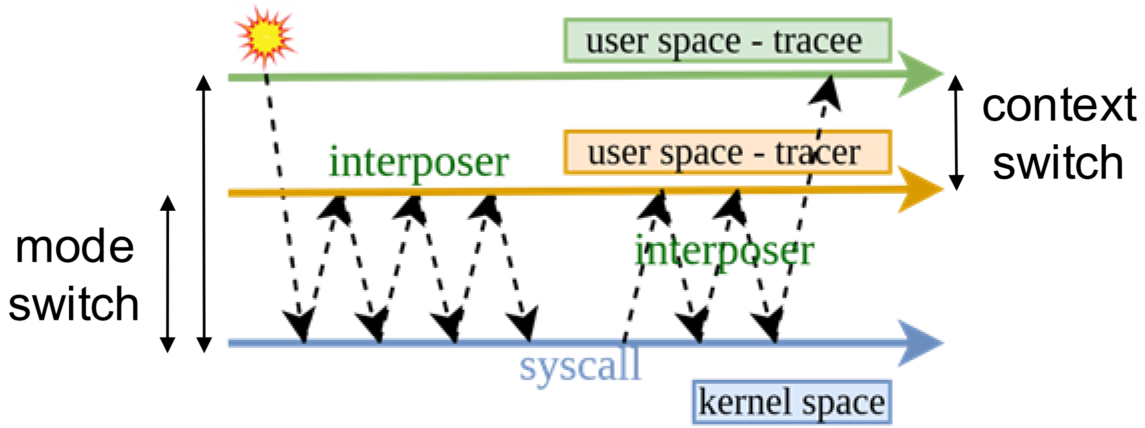
A side-channel to infer execution paths

Syscall interposition is a technique that **observes, modifies, blocks, or emulates system calls** made by a process as it interacts with the operating system.

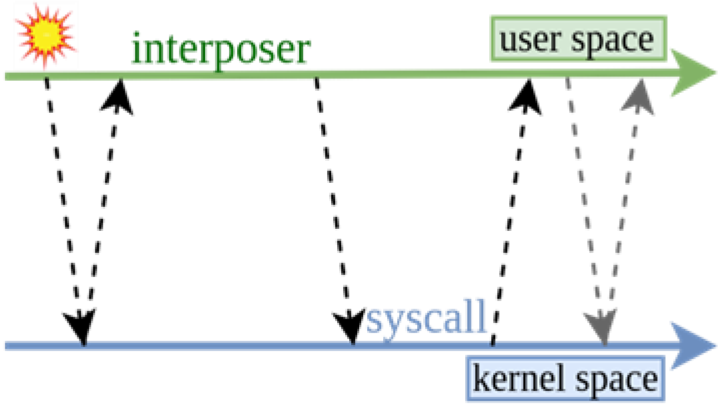
Syscall interposition does not require source code.

Syscall Interposition Tools

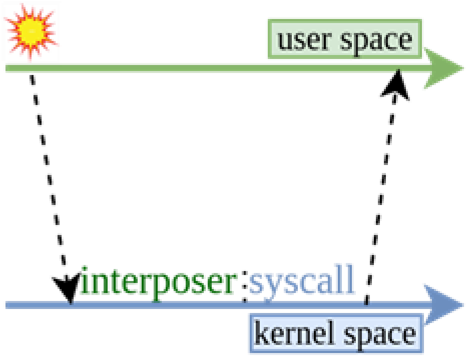
ptrace



Kernel-to-user trap



Kernel-level interposition



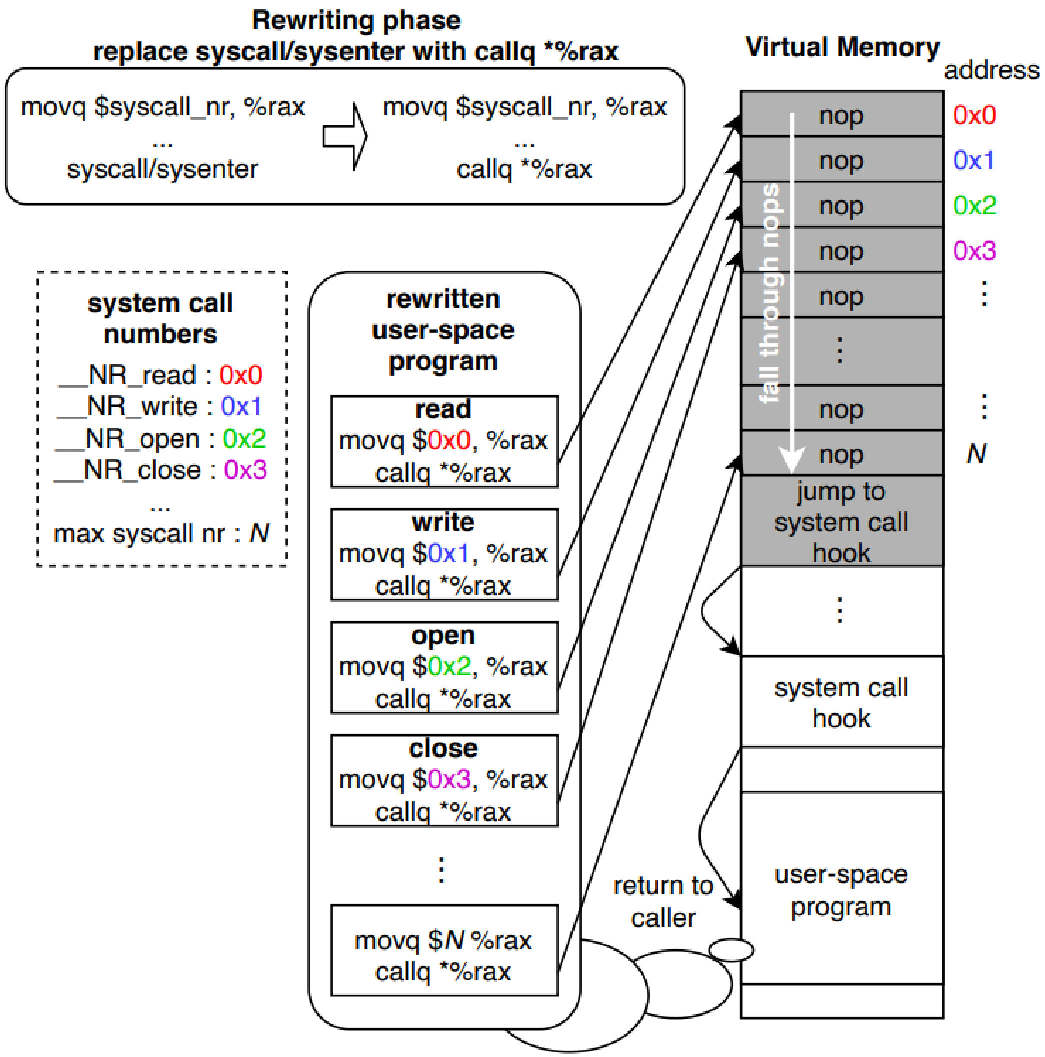
We can use Binary Rewritting to put the interposer in the Userspace?!

Binary Rewritting

1. Identify `syscall` instructions*
   - Coverage vs Correctness
   - Code vs data
   - Unaligned instructions → heuristics
   - Obfuscation
   - Dynamically loaded/generated code
2. Rewrite `syscall` instructions
   - `jmp/call` > 2 bytes
   - Assumptions about surrounding code

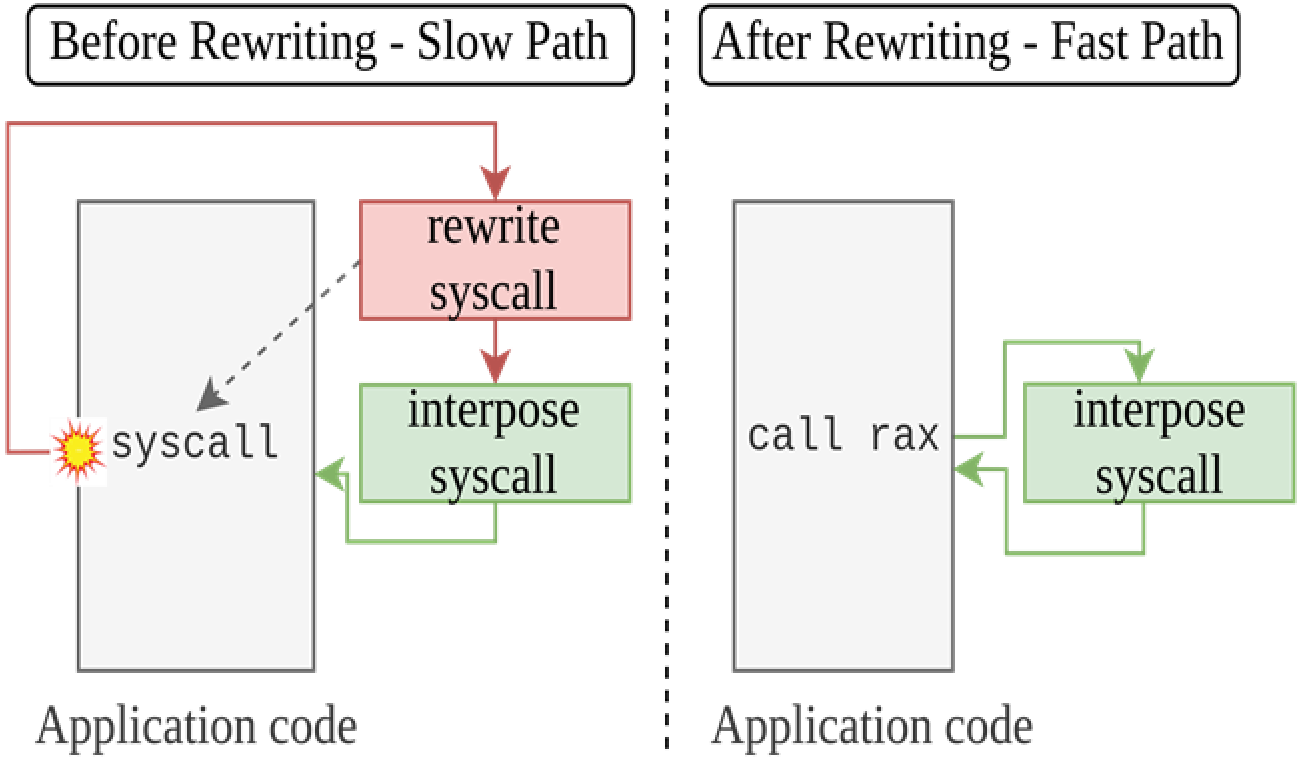
Frameworks: zpoline

- USENIX ATC 2023
- syscall/sysenter → call rax
- NULL page → NOP-sled
- Interposer hook → after NOPs
- What about JIT?



Frameworks: lazypoline

1. Use the kernel (SUD) at first: identify `syscall` instructions on their first use
2. Stop using the kernel: rewrite `syscall` instructions on the fly
3. Problems? (Might break correct programs) and what else?



A. Jacobs, M. Gülmez, A. Andries, S. Volckaert , **and A. Voulimeneas**.
“System Call Interposition Without Compromise”.
In DSN 2024.

Expressive
Exhaustive
Efficient

Lazypol

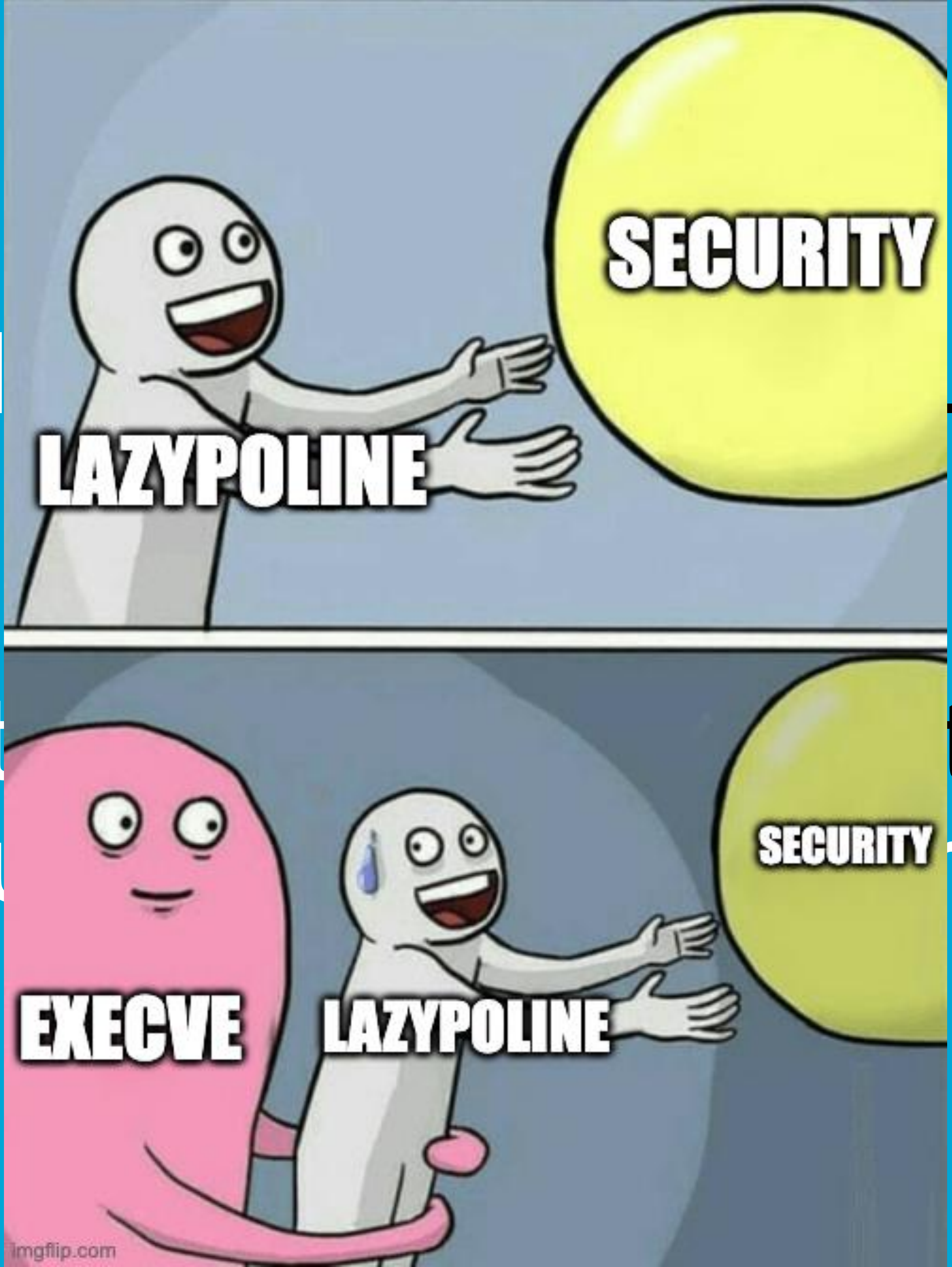
LOAD to

Can you t

tic syscall

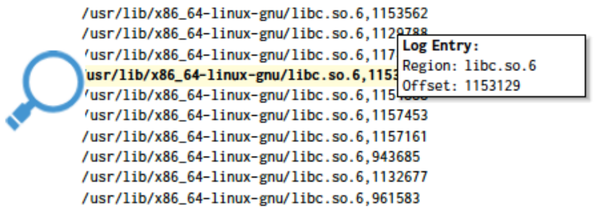
that cou

rposer?



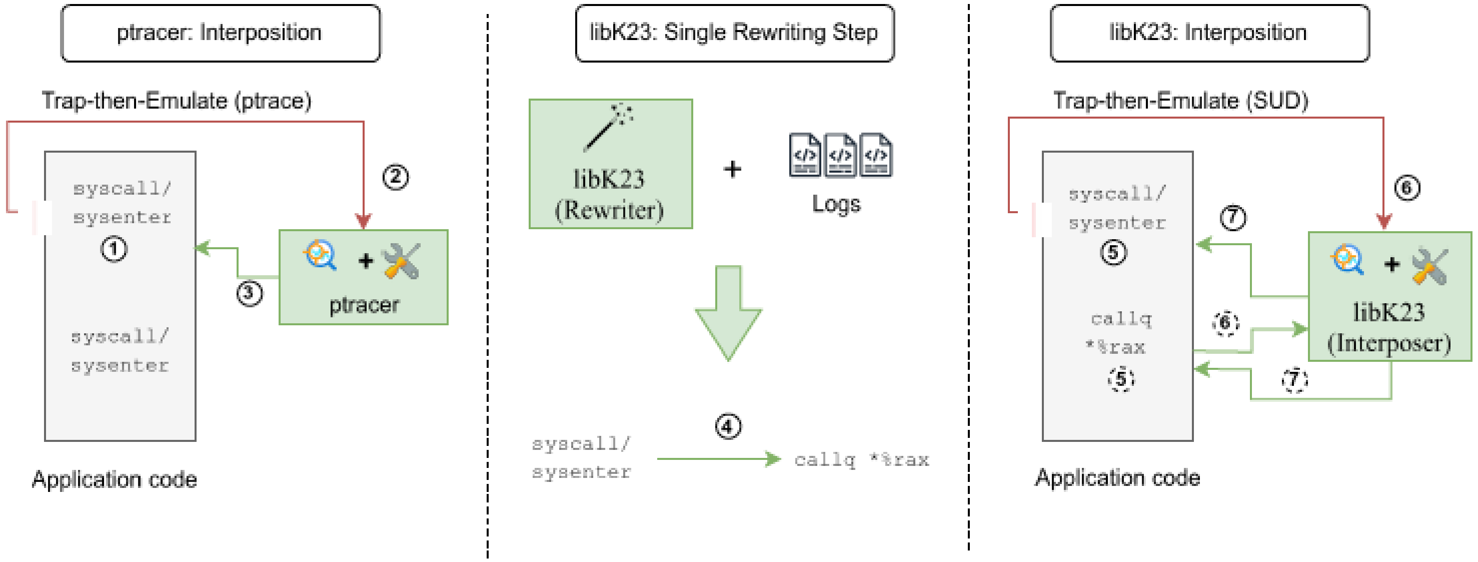
Frameworks: K23 (Our lab’s work)

1. **Offline Phase**: Gather offset of syscall instruction in libraries through the slow path (SUD sig-handler)
2. **Loading phase:** ptrace to catch load-time syscalls
3. **Online Phase – Rewrite** based on the logs
4. **Online Phase – Interpose** with both slow and fast path, BUT do not rewrite (no longer breaking correct programs)



```
/usr/lib/x86_64-linux-gnu/libc.so.6,1153562
/usr/lib/x86_64-linux-gnu/libc.so.6,1129788
/usr/lib/x86_64-linux-gnu/libc.so.6,117
/usr/lib/x86_64-linux-gnu/libc.so.6,1153
/usr/lib/x86_64-linux-gnu/libc.so.6,115
/usr/lib/x86_64-linux-gnu/libc.so.6,1157453
/usr/lib/x86_64-linux-gnu/libc.so.6,1157161
/usr/lib/x86_64-linux-gnu/libc.so.6,943685
/usr/lib/x86_64-linux-gnu/libc.so.6,1132677
/usr/lib/x86_64-linux-gnu/libc.so.6,961583
```

```
Log Entry:
Region: libc.so.6
Offset: 1153129
```



J. M. Gómez, **V. Moutafis**, A. Dionysiou, F. Kuipers, G. Smaragdakis, B. Coppens, **and A. Voulimeneas.** **“Clair Obscur: The Light and Shadow of System Call Interposition – From Pitfalls to Solutions with K23”.** (To Appear) In Middleware 2025.

Thanks!
Questions?

Vissarion Moutafis

15-12-2025

REFERENCES

- Busch, M., Machiry, A., Spensky, C., Vigna, G., Kruegel, C., & Payer, M. (2023). TEEzz: Fuzzing Trusted Applications on COTS Android Devices. *2023 IEEE Symposium on Security and Privacy (SP)*, 1204–1219. https://doi.org/10.1109/SP46215.2023.10179302
- Daniele, C., Bethe, T., Maugeri, M., Continella, A., & Poll, E. (n.d.). *LibAFLstar: Fast and State-Aware Protocol Fuzzing*.
- Daniele, C., Andarzian, S. B., & Poll, E. (2024). Fuzzers for Stateful Systems: Survey and Research Directions. *ACM Computing Surveys*, *56*(9), 1–23. https://doi.org/10.1145/3648468
- Pham, V.-T., Bohme, M., & Roychoudhury, A. (2020). AFLNET: A Greybox Fuzzer for Network Protocols. *2020 IEEE 13th International Conference on Software Testing, Validation and Verification (ICST)*, 460–465. https://doi.org/10.1109/ICST46399.2020.00062
- Fioraldi, A., Maier, D., Eißfeldt, H., & Heuse, M. (n.d.). *AFL++: Combining Incremental Steps of Fuzzing Research*.
- Hu, Z., & Pan, Z. (2021). A Systematic Review of Network Protocol Fuzzing Techniques. *2021 IEEE 4th Advanced Information Management, Communicates, Electronic and Automation Control Conference (IMCEC)*, *4*, 1000–1005. https://doi.org/10.1109/IMCEC51613.2021.9482063
- Zhang, Z., Zhang, H., Zhao, J., & Yin, Y. (2023). A Survey on the Development of Network Protocol Fuzzing Techniques. *Electronics*, *12*(13), 2904. https://doi.org/10.3390/electronics12132904
- Gascon, H., Wressnegger, C., Yamaguchi, F., Arp, D., & Rieck, K. (2015). Pulsar: Stateful Black-Box Fuzzing of Proprietary Network Protocols. In B. Thuraisingham, X. Wang, & V. Yegneswaran (Eds.), *Security and Privacy in Communication Networks* (Vol. 164, pp. 330–347). Springer International Publishing. https://doi.org/10.1007/978-3-319-28865-9_18
- Xiao, J., Jiang, P., Zhao, Z., Huang, R., Liu, J., & Li, D. (n.d.). *Robust, Efficient, and Widely Available Greybox Fuzzing for COTS Binaries with System Call Pattern Feedback*.
- The fuzzing book, https://www.fuzzingbook.org/