Report: BADASS (Executable Stack Issues) and Why It's Not Possible in OP-TEE

1. Paper Summary

Background

- Historically, attackers exploited **buffer overflows** by injecting shellcode into the stack and executing it.
- The defense mechanism **W⊕X (Write XOR Execute)** prevents this: memory pages can be **writable or executable, but not both**.
- Modern OSes (Linux, Windows, BSDs) and CPUs enforce W⊕X using the **NX (No-Execute)** bit in page tables.

The Problem (BADASS)

The authors identify a recurring issue they call **BADASS**:

If a program includes an assembly file that does **not** contain the directive:

.section .note.GNU-stack,"",@progbits

then the final binary may end up with an **executable stack**, breaking W⊕X.

- Even experienced security researchers (working on IRMs, binary rewriters, CFI tools) introduced BADASS by mistake.
- **11 out of 21** investigated security-hardening tools (e.g., **MCFI, RockJIT, πCFI, ERIM, PathArmor**) produced binaries with executable stacks.

- This happens because the enforcement of W⊕X in Linux relies on a **chain of trust** across:
    - Compiler (GCC/Clang)
    - Assembler
    - Linker
    - Loader
    - Linux kernel

If any step misses the `.note.GNU-stack` section, the chain breaks and the process stack becomes executable.

Consequences

- Attackers can directly place **shellcode on the stack** and execute it.
- This **undoes decades of mitigations** and makes exploitation easier.
- The issue isn't just developer carelessness — it's caused by the **subtle design** of the GNU toolchain and ELF semantics.

2. Why BADASS Is Not Possible in OP-TEE

Key Points About OP-TEE

Different Execution Model

- OP-TEE runs in the **secure world (TrustZone)**.
- Code executes inside the **TEE core** and **Trusted Applications (TAs)**.
- Unlike Linux processes, OP-TEE does **not rely on ELF loaders or GNU toolchain behavior** for stack permissions.

Memory Management

- The OP-TEE OS kernel controls its own memory mappings using **page tables** configured in the secure world.
- Stacks for both TEE core and TAs are explicitly mapped as **non-executable**.
- This is enforced by **MMU settings with NX bits**, not ELF metadata like `.note.GNU-stack`.

Compilation & Linking Differences

- In Linux: missing `.note.GNU-stack` → linker sets `PT_GNU_STACK` with `PF_X` → stack executable.
- In OP-TEE:
  - Build system uses strict Clang/GCC flags.
  - Final enforcement is done by the **TEE OS**, not ELF section flags.
  - TAs are **custom TA binaries**, not standard ELF executables.
  - The OP-TEE loader **ignores `.note.GNU-stack`** entirely.

No Legacy Compatibility Concerns

- Linux BADASS exists partly due to **legacy support** (nested functions, trampolines, old CPUs).
- OP-TEE has **no legacy baggage** — all TAs are compiled in a tightly controlled environment.
- Thus, the subtle `.note.GNU-stack` behavior never applies.

Security Policy

- OP-TEE enforces **strict separation of code and data regions**.
- Even if a developer forgot `.note.GNU-stack`, OP-TEE's runtime memory mapping ensures **stacks remain non-executable**.

3. The BADASS Problem on Linux

Root Cause

In Linux, if any assembly file misses:

.section .note.GNU-stack,"",@progbits

the **compiler + assembler + linker chain** may produce a binary with an **executable stack**.

This happens because Linux relies on `.note.GNU-stack` and `PT_GNU_STACK` ELF segment flags to enforce W⊕X.

Example Demonstration

- Without `a.s`: stack is `rw-` (non-executable).

```
kali@kali-virtual-machine:~/mtp1/buffer$ gcc code.c -o code
kali@kali-virtual-machine:~/mtp1/buffer$ readelf -W -l code | grep 
GNU_STACK -A1 || true
  GNU_STACK      0x000000 0x0000000000000000 0x0000000000000000 0x0
00000 0x000000 RW  0x10
  GNU_RELRO      0x002db8 0x0000000000003db8 0x0000000000003db8 0x0
00248 0x000248 R   0x1
```

- With `a.s`: stack becomes `rwx` (executable).

```
kali@kali-virtual-machine:~/mtp1/buffer$ gcc code.c a.s -o codebad 
kali@kali-virtual-machine:~/mtp1/buffer$ readelf -W -l codebad | gr
ep GNU_STACK -A1 || true
  GNU_STACK      0x000000 0x0000000000000000 0x0000000000000000 0x0
00000 0x000000 RWE 0x10
  GNU_RELRO      0x002db8 0x0000000000003db8 0x0000000000003db8 0x0
00248 0x000248 R   0x1
kali@kali-virtual-machine:~/mtp1/buffer$ 
```

Security Tools Also Affected

- Even **hardened tools** (IRMs, CFI frameworks, binary rewriters) made this mistake.
- 11 out of 21 tested tools (e.g., **MCFI, RockJIT, πCFI, ERIM, RetroWrite**) introduced executable stacks.

Why It Happens Frequently

W⊕X enforcement in Linux is a **chain of trust**:

1. Compiler inserts `.note.GNU-stack`.
2. Assembler translates it.
3. Linker sets `PT_GNU_STACK`.
4. Loader + kernel apply stack permissions.

If any step misses the directive, the chain breaks → stack becomes executable.

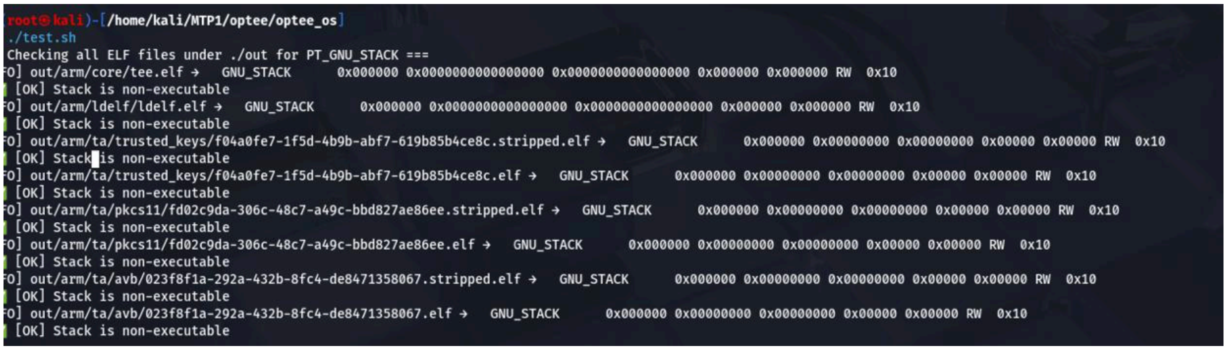
4. Why BADASS Is Not Possible in OP-TEE

Different Execution Environment

- OP-TEE runs in the **secure world (TrustZone)**, unlike Linux user processes.
- TAs are loaded in a **custom TA format**, not standard ELF with `PT_GNU_STACK`.

Memory Management in OP-TEE

- The **TEE core configures page tables** directly.
- Stacks are **always non-executable** (NX enforced).
- Enforcement happens **at runtime** by the TEE OS, not ELF flags.



No Legacy Compatibility Problems

- Linux BADASS comes from **legacy compatibility** issues.
- OP-TEE has **no such legacy requirements**.
- Even if `.note.GNU-stack` is missing, it does not affect execution in OP-TEE.

5. Comparison: Linux vs. OP-TEE

Feature	Linux (BADASS possible)	OP-TEE (BADASS impossible)
Stack permissions	Based on `.note.GNU-stack` + ELF	Enforced by MMU directly
Dependency chain	Compiler → Assembler → Linker → Loader → Kernel	Secure kernel only
Risk if `.note.GNU-stack` missing	Stack may become executable	No effect, stack always NX
Legacy support	Yes (nested functions, trampolines)	No legacy baggage
Real-world BADASS cases	Electron, VSCode, CockroachDB, IRMs	None possible

6. Conclusion

The paper shows that on Linux, **BADASS emerges because**:

- The GNU toolchain + ELF semantics require a subtle `.note.GNU-stack` directive.
- Developers (even experts) sometimes forget it in hand-written assembly.
- As a result, stacks can unintentionally become **executable**, enabling attacks.

In **OP-TEE**, this problem is **fundamentally not possible** because:

- Stacks are **always non-executable** at runtime.
- OP-TEE does not depend on ELF `PT_GNU_STACK` metadata.
- The secure-world environment is **closed and controlled**.
- Security policies are enforced directly by the **MMU and TEE OS**.

On Linux, BADASS requires vigilance and toolchain fixes.
On OP-TEE, architecture itself **prevents this class of issue by design**.