MOVE 课程3-Move漏洞分析

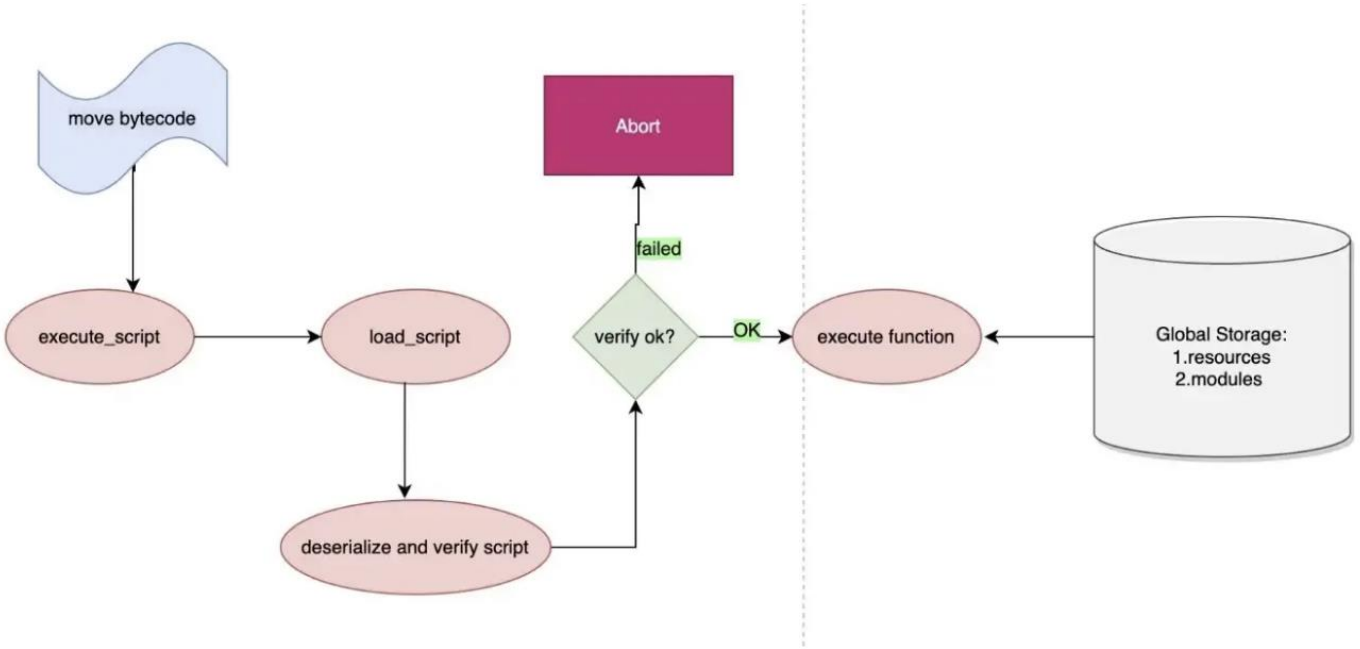
WEB3 and Beyond

Speaker:Nolan Wang

MOVE Vulnerability anaylsis

operate stack integer overflow

https://medium.com/numen-cyber-labs/analysis-of-the-first-critical-0-day-vulnerability-of-aptos-move-vm-8c1fd6c2b98e



1.将字节码通过函数execute_script被加载进来
2.执行load_script函数，这个函数主要用来反序列化字节码，并校验字节码是否合法，如果校验失败，就会返回失败
3.校验成功之后就会开始执行真正的字节码代码
4.执行字节码，访问或修改全局存储的状态，包括资源，modules

operate stack integer overflow

•BoundsChecker，边界检查，主要是用来检查module与script的边界安全。具体包括检查signature，constants等的边界
•DuplicationChecker， 该模块实现了一个检查器，用于验证 CompiledModule 中的每个向量是否包含不同的值
•SignatureChecker，用于检查signature被用于函数参数，本地变量，结构体成员时，字段结构正确
•InstructionConsistency，验证指令一致性
•constants用于验证常量，常量的类型必须是原始类型，常量的数据正确的序列化为其类型
•CodeUnitVerifier，验证函数体代码的正确性，分别通过stack_usage_verifier.rs与abstract_interpreter.rs来达到目的
•script_signature，用于验证一个脚本或入口函数是否是一个有效的签名

```
pub fn verify_script_with_config(config: &VerifierConfig, script: &(
    BoundsChecker::verify_script(script).map_err(|e| e.finish(Locat
    DuplicationChecker::verify_script(script)?;
    SignatureChecker::verify_script(script)?;
    InstructionConsistency::verify_script(script)?;
    constants::verify_script(script)?;
    CodeUnitVerifier::verify_script(config, script)?;
    script_signature::verify_script(script, no_additional_script_sig
}
```

漏洞发生在verify环节 CodeUnitVerifier::verify_script(config, script)?;函

栈安全校验(StackUsageVerifier::verify)

该模块用于验证函数的字节码指令序列中的基本块是否以平衡的方式使用。 每个基本块除了那些以 Ret（返回给调用者）操作码结尾的，必须确保离开block时候栈高度 与开头时候相同。
循环校验所有代码块是否满足以上条件：

```
impl<'a> StackUsageVerifier<'a> {
    pub(crate) fn verify(
        resolver: &'a BinaryIndexedView<'a>,
        function_view: &'a FunctionView,
    ) -> PartialVMResult<()> {
        let verifier = Self {
            resolver,
            current_function: function_view.index(),
            code: function_view.code(),
            return_: function_view.return_(),
        };

        for block_id in function_view.cfg().blocks() {
            verifier.verify_block(block_id, function_view.cfg())?
        }
        Ok(())
    }
```

漏洞详情

由于movevm是栈虚拟机，在验证指令合法性的时候：
第一需要确保指令字节码是否正确
第二需要确保栈空间经过一个block代码块调用之后，栈内存合法，即栈操作之后，栈保持平衡。

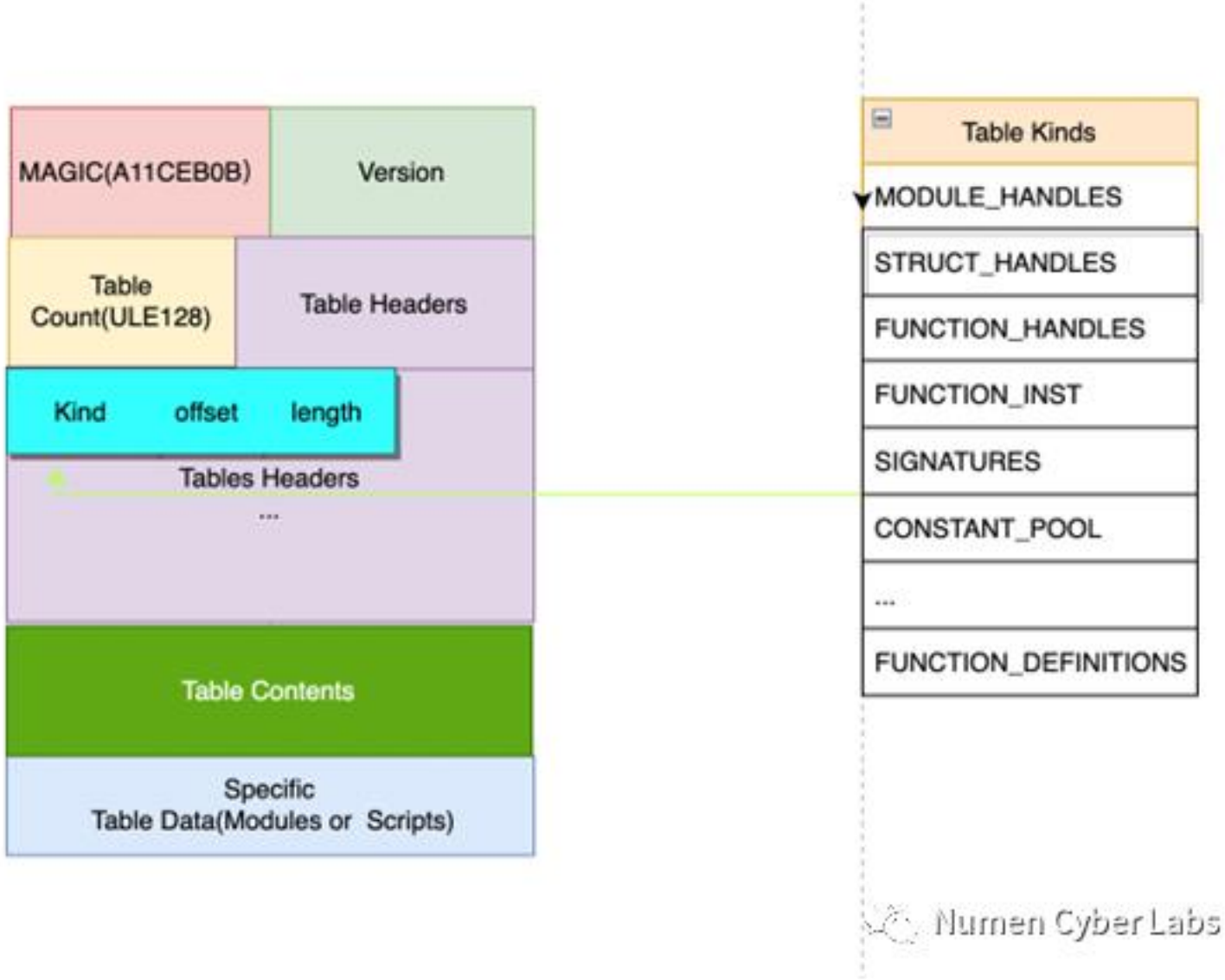
```
fn verify_block(&self, block_id: BlockId, cfg: &dyn ControlFlowGraph)
        let code = &self.code.code;
        let mut stack_size_increment = 0;
        let block_start = cfg.block_start(block_id);
        for i in block_start..=cfg.block_end(block_id) {
            let (num_pops, num_pushes) = self.instruction_effect(&code
            // Check that the stack height is sufficient to accommodat
            // of pops this instruction does
            if stack_size_increment < num_pops {
                return Err(
                    PartialVMError::new(StatusCode::NEGATIVE_STACK_SIZ
                        .at_code_offset(self.current_function(), block
                );
            }
            stack_size_increment -= num_pops;
            stack_size_increment += num_pushes;
        }

        if stack_size_increment == 0 {
            Ok(())
        } else {
            Err(
                PartialVMError::new(StatusCode::POSITIVE_STACK_SIZE_AT
                    .at_code_offset(self.current_function(), block_sta
            )
        }
    }

```

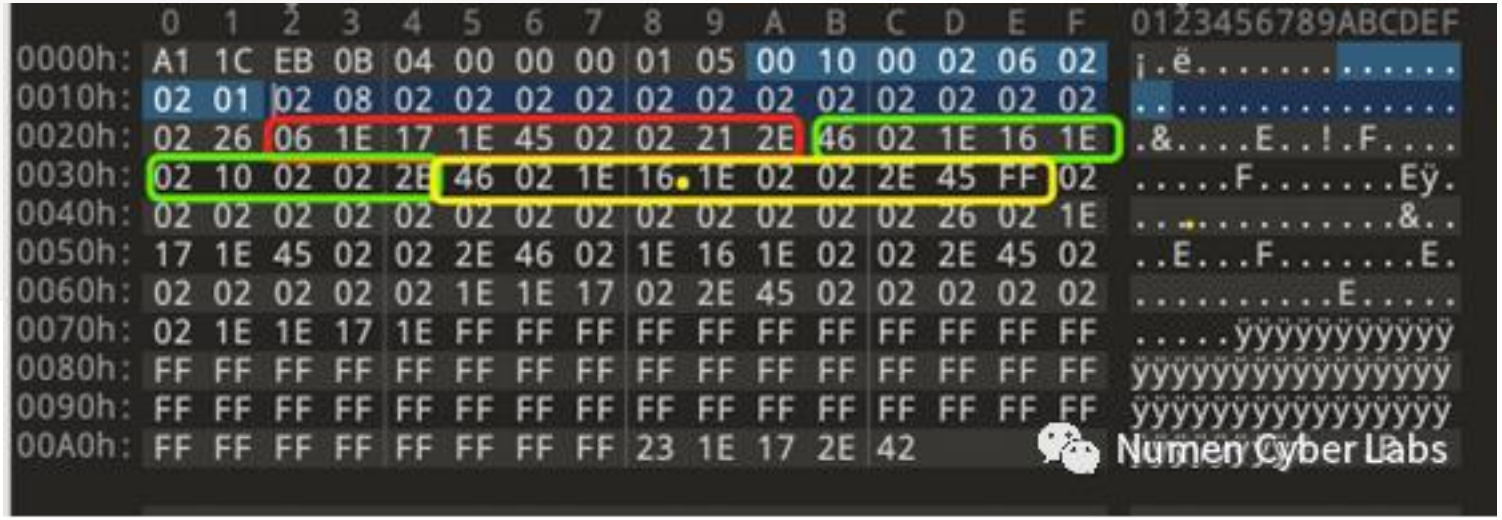
看起来这里似乎没什么问题，但是由于这里在执行16行代码的时候，没有去判断是否存在整数溢出，导致可以通过构造超大num_pushes，间接控制stack_size_increment，从而产生整数溢出漏洞。那么如何构造构造这样一个巨大的push数目呢?

move bytecode 文件格式



```
// Move bytecode v4
script {


main<Ty0: drop, Ty1: drop>(Arg0: u8) {
B0:
        0: LdU64(3323940208748926750)
        1: VecUnpack(2, 3315214543476364830)
        2: VecUnpack(2, 18394158839224997406)
        3: Ret
B1:
        4: Ret
B2:
        5: Ret
```



PoC构造

执行完instruction_effect函数:
1. 第一次返回(1,3315214543476364830)，此时stack_size_increment为0,num_pops为1，num_pushes为3315214543476364830
2.执行第二次返回(1,18394158839224997406)。当再次执行 stack_size_increment += num_pushes;
stack_size_increment已经为0x2e020210021e161d(33152145434763648 29)，
num_pushes为0xff452e02021e161e(18394158839224997406)，当两者相加之后，大于u64的最大值

- 产生了数据截断，stack_size_increment的值成为了0x12d473012043c2c3b，造成了整数溢出。

Move DoS vulnerability

Numen Cyber Technology

中文：https://mp.weixin.qq.com/s/IY8nj73J1oyOJ-oomDcOPg
英文：https://medium.com/numen-cyber-labs/the-story-of-a-high-vulnerability-in-move-reference-safety-verify-module-2340f3d8c642

Numen Cyber独家发现Move语言又一高危漏洞

Original Numen Cyber Labs Numen Cyber Labs 2022-11-24 11:53 Posted on 新加坡

收录于合集
#安全漏洞 9 #Web3安全 22

0x0 前言

之前， 我们发现了一个Aptos Move VM的严重漏洞，经过深入研究，我们发现了另外一个新的整数溢出漏洞。这次的漏洞触发过程相对更有趣一点。 下面是对这个漏洞的深入分析，里面包含了很多Move语言本身的背景知识。通过本文讲解相信你会对Move语言有更深入的理解。

众所周知，Move 语言在执行字节码之前会验证代码单元。验证代码单元的过程，分为4步。这个漏洞就出现在 reference_safety 的步骤中。

```
fn verify_common(&self) -> PartialVMResult<()> {
        StackUsageVerifier::verify(&self.resolver, &self.function_view)
```

总结

- 没有绝对安全的系统
- 没有绝对安全的语言
- 安全都是相对的

NUMEN

Thank you

for your attention

Contact us at

+65 6355 5555

Contact@numencyber.com

Find us at