Syscoin Bridge: How a Proof Parsing Flaw Minted 5 Billion SYS From Nothing

Key တစ်ခု ခိုးခံရတာ မဟုတ်ဘဲ၊ Cross Chain Proof ကို ဖတ်တဲ့ Code ထဲက Parsing Bug တစ်ခုကြောင့်ပဲ Token တွေ Mint ဖြစ်သွားခဲ့တဲ့ အကြောင်း။

2026 ခုနှစ် ဇွန်လ 7 ရက်နေ့မှာ၊ attacker တစ်ယောက်က Syscoin Bridge ကို **SYS 5 ဘီလီယံ** လောက် (အဲ့ဒီအချိန်က Circulating Supply တစ်ခုလုံးရဲ့ **568% လောက်**) ကို တကယ့် Coin တစ်ခုမှ Burn မလုပ်ဘဲ Mint လုပ်ခိုင်းနိုင်ခဲ့ပါတယ်။ Private Key ခိုးခံရတာ မဟုတ်ပါဘူး။ Signature ကို Forge လုပ်ခံရတာလည်း မဟုတ်ပါဘူး။ attacker က Valid Proof တစ်ခုနဲ့ တူနေအောင် ဖန်တီးထားတဲ့ Data တစ်ခုကို Bridge ဆီ ပေးလိုက်တာနဲ့ Bridge က ယုံသွားခဲ့တာပါ။

ဒီ Blog Post မှာ Burn and Mint Bridge တစ်ခု ဘယ်လို အလုပ်လုပ်လဲ၊ Proof ကို ဖတ်တဲ့ Code Path က ဘယ်မှာလဲ၊ အဲ့ဒီ Reader မှာ Bug တစ်ခုရှိရုံနဲ့ ဘာလို့ ပိုက်ဆံ Print ထုတ်သလို ဖြစ်သွားနိုင်လဲ၊ ပြီးတော့ တကယ့် On Chain မှာ ဘယ်လို ဖြစ်ပျက်သွားလဲဆိုတာ အသေးစိတ် ရှင်းပြပေးသွားပါမယ်။ တစ်ခု ပြောထားချင်တာက Syscoin နဲ့ Halborn တို့က Bug ရဲ့ အမျိုးအစား (Proof Parsing Flaw) ကိုတော့ အတည်ပြုထားပေမယ့်၊ ပျက်သွားတဲ့ တိကျတဲ့ Byte ကိုတော့ ဘယ်တုန်းကမှ ထုတ်ပြန်ခဲ့တာ မရှိပါဘူး။ ဒါကြောင့် ကျွန်တော်က vulnerability ဖြစ်ခဲ့တဲ့ တကယ့် Open Source Parsing Code ကို ပြပေးပါမယ်ဗျ။ ပြီးတော့ confirmed fact တွေနဲ့ ကျွန်တော့်ရဲ့ reasons တွေကို ခွဲခြားပြီး သေချာ ပြောပြပေးပါမယ်ဗျ။

အကျဉ်းချုပ် (Summary)

ဒီ Incident မှာ ဘာတွေ ဖြစ်ခဲ့လဲဆိုတာကို အရင်ဆုံး အကျဉ်းချုပ် ပြောပြချင်ပါတယ်။

- **Protocol:** Syscoin Bridge (UTXO Chain နဲ့ NEVM ကြား)၊ `SyscoinRelay` နဲ့ `SyscoinVaultManager` Contract တွေပါ။ ကိုးကားထားတဲ့ Code အားလုံးကို `syscoin/sysethereum-contracts` Repo ရဲ့ Commit `5405c13` ကို အခြေခံထားပါတယ်။
- **Bug အမျိုးအစား:** SPV Proof Parsing ထဲက Logic Error တစ်ခုပါ။ Bridge က Malformed (ပျက်နေတဲ့) Proof တစ်ခုကို Valid လို့ လက်ခံလိုက်တာပါ။
- **ဆုံးရှုံးမှု:** SYS 5,000,000,000 (5 ဘီလီယံ) ကို ဘာမှ မရှိဘဲကနေ Mint လုပ်ခဲ့တာ ဖြစ်ပြီး၊ ဇွန် 7 ရက် ဈေးနှုန်း $0.00171187 အရ $8.5M လောက် တန်ပါတယ်။ အဲ့ဒီ Token အသစ်တွေက Attack ပြီးနောက်မှာ Circulating Supply ရဲ့ 85% လောက် ရှိသွားပါတယ်။
- **နေ့စွဲ:** 2026 ခုနှစ် ဇွန်လ 7 ရက်။

- **လက်ရှိအခြေအနေ:** Bridge ကို နာရီပိုင်းအတွင်း ရပ်လိုက်ပါတယ်။ Preliminary Postmortem ကို တနေ့ချင်းမှာ ထုတ်ပြန်ခဲ့တယ်။ Whitehat Recovery Address ကို ဇွန် 9 ရက်မှာ တင်ပြီး၊ Funds တွေကို ဇွန် 11 ရက်မှာ ပြန်ရခဲ့ပါတယ်။ Proof Validation Path ကိုလည်း ပြင်ဆင်မှု လုပ်ပြီး Review လုပ်ခဲ့ပါတယ်။

နောက်ခံ (Background): Burn and Mint Bridge က ဘာလုပ်ပေးတာလဲ

Syscoin မှာ နှစ်ဘက် ရှိပါတယ်။

- **UTXO Chain** က Bitcoin ပုံစံ Chain ဖြစ်ပြီး SYS တွေက Unspent Transaction Output (UTXO) အနေနဲ့ ရှိနေတာပါ။ Bitcoin သုံးတဲ့ Model အတိုင်းပဲ။
- **NEVM** (Network Enhanced Virtual Machine) ကတော့ Ethereum လို EVM နဲ့ တွဲဖက် အလုပ်လုပ်နိုင်တဲ့ Smart Contract Chain ပါ။

Bridge က ဒီ နှစ်ဘက်ကြားမှာ SYS တွေ ရွှေ့ပြောင်းပေးတာပါ။ သူ့ကို ထိန်းထားတဲ့ rule က ရိုးရိုးလေးပေမယ့် တစ်ခုလုံးရဲ့ အဓိက အစိတ်အပိုင်းပါပဲ။

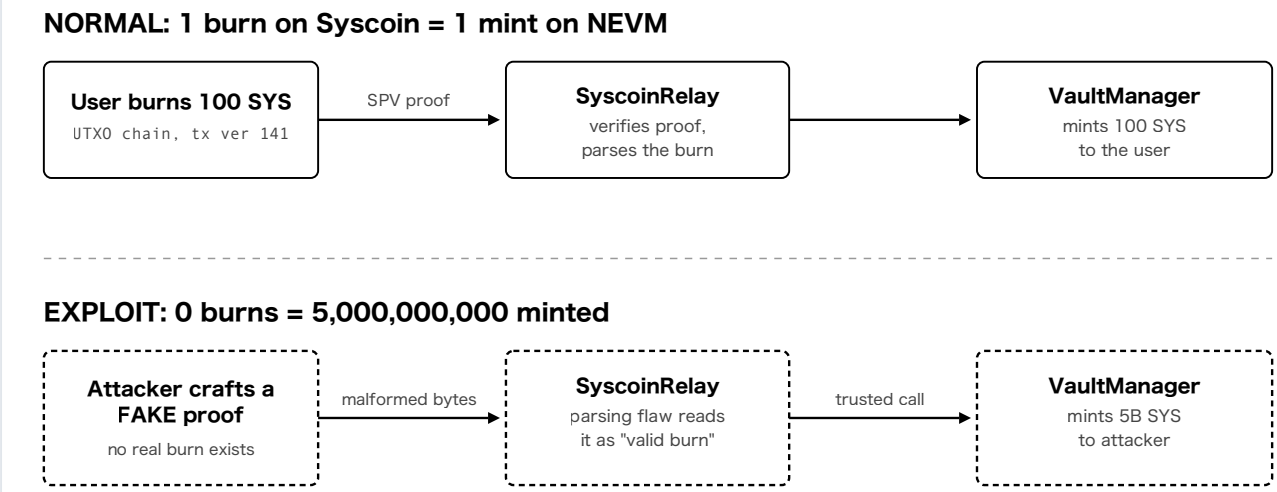
ဘက်တစ်ဘက်မှာ Mint လုပ်တဲ့ Coin တိုင်းဟာ၊ တခြားဘက်မှာ **Burn (ဖျက်ဆီး)** လုပ်လိုက်တဲ့ Coin တစ်ခုနဲ့ Back ဖြစ်နေရပါမယ်။ Burn တစ်ခုဝင်ရင် Mint တစ်ခုထွက်။ ဘယ်တော့မှ ပိုမထွက်ရပါဘူး။

UTXO Chain ကနေ NEVM Chain ဆီ SYS ရွှေ့တဲ့အခါ၊ UTXO ဘက်မှာ **Burn Transaction** အထူး တစ်ခု ဖန်တီးရပါတယ် (Syscoin က Transaction Version `141` နဲ့ မှတ်သားထားတယ်)။ အဲ့ဒီ Burn က "ဒီမှာ SYS 100 ကို ဖျက်လိုက်ပြီ၊ ကျွန်တော့် NEVM Address ဆီ SYS 100 ထုတ်ပေးပါ" လို့ ပြောလိုက်တာပါ။

ဒါပေမယ့် NEVM Contract က UTXO Chain ကို တိုက်ရိုက် မမြင်ရဘူး။ ဒါဆိုရင် Burn တကယ်ဖြစ်ခဲ့တာကို သူ ဘယ်လို သိမလဲ။ အဲ့ဒါကို **SPV Proof** နဲ့ စစ်တာပါ။

SPV ဆိုတာ Simplified Payment Verification ပါ။ Contract ကို Blockchain တစ်ခုလုံး မပေးဘဲ၊ Proof သေးသေးလေး တစ်ခုပဲ ပေးတာပါ။ အဲ့ဒီထဲမှာ Raw Burn Transaction၊ သူ ရှိနေတဲ့ Block Header၊ ပြီးတော့ **Merkle Siblings** short list ပါတယ်။ အဲ့တာတွေနဲ့ Contract က Merkle Root ကို ပြန်တွက်ပြီး၊ ဒီ Transaction က တကယ့် Mined Block တစ်ခုထဲမှာ ရှိနေတာ ဟုတ်မဟုတ် စစ်နိုင်ပါတယ်။ math match ဖြစ်တယ်ဆိုရင် Burn လုပ်တာ မှန်တာပါ။

ဒါကြောင့် Bridge ရဲ့ Trust Model က **Burn ကို သက်သေပြ၊ ပြီးမှ Mint လုပ်။** Proof က တစ်ခုတည်းသော Gatekeeper ပါ။ အဲ့ဒီအချက်ကို သိထားရပါမယ်ဗျ။



The relay is the only thing standing between a proof and a mint.
VaultManager.processTransaction() trusts the relay completely (onlyTrustedRelayer).
So if the relay's parser can be fooled, the mint follows automatically.

အပေါ်ပုံက ဖြစ်သင့်တဲ့ ပုံစံ၊ Burn တစ်ခါလုပ်တိုင်း Mint တစ်ခါသာ ဖြစ်ရမယ်။ အောက်တန်းကတော့ Exploit၊ နောက်ကွယ်မှာ တကယ့် Burn မရှိတဲ့ Fake Proof တစ်ခုက Mint လုပ်နိုင်ခဲ့တယ်။ ဘာလို့လဲဆိုတော့ Relay က proof ကို စစ်တဲ့ တစ်ခုတည်းသော နေရာ ဖြစ်ပြီး သူ့ Parser ကို attacker က fooled လုပ်လိုက်လို့ပါ။

အားနည်းချက်ရှိတဲ့ Code (Vulnerable Code)

Gatekeeper က `SyscoinRelay.relayTx()` ပါ။ ဘယ်သူမဆို Proof တစ်ခုနဲ့ သူ့ကို ခေါ်နိုင်တယ်။ တကယ့် Entry Point က ဒီမှာပါ (`SyscoinRelay.sol` , lines 277-312)။

```
function relayTx(
    uint64 _blockNumber,
    bytes memory _txBytes,        // the raw burn transaction (attacker supplied)
    uint _txIndex,
    uint[] memory _txSiblings,    // Merkle siblings (attacker supplied)
    bytes memory _syscoinBlockHeader
) external override returns (uint) {
    uint txHash = verifySPVProofs(          // (1) is this tx really in a block?
        _blockNumber, _syscoinBlockHeader, _txBytes, _txIndex, _txSiblings
    );
    require(txHash != 0, "SPV proof verification failed");

    (
        uint errorCode,
        uint value,                         // (2) how much was burned?
        address destinationAddress,         //     who gets the minted coins?
        uint64 assetGuid
    ) = parseBurnTx(_txBytes);              // (3) read those fields out of raw bytes
    if (errorCode != 0) {
        emit RelayTransaction(bytes32(txHash), errorCode);
        return errorCode;
    }

    syscoinVaultManager.processTransaction( // (4) mint to the destination
        txHash, value, destinationAddress, assetGuid
    );
    return value;
}
```

နံပါတ်တပ်ထားတဲ့ အဆင့် လေးဆင့်ကို Trust ရဲ့ chain တစ်ခုလို ဖတ်ကြည့်ပါ။

1. `verifySPVProofs()` က Block Header စစ်မှန်မှု ရှိမရှိ၊ Transaction က Block ရဲ့ Merkle Root အောက်မှာ ရှိမရှိ စစ်တယ်။
2. ပြီးတော့ `parseBurnTx()` က Raw Transaction Bytes တွေကို လျှောက်ဖတ်ပြီး၊ ပမာဏ၊ Destination Address နဲ့ Asset ကို ဆွဲထုတ်တယ်။
3. `processTransaction()` က Vault ပေါ်မှာ အဲဒီ ပမာဏကို အဲဒီ Address ဆီ Mint လုပ်ပေးတယ်။

အခု Mint လုပ်တဲ့ဘက်က ဘာလုပ်လဲ ကြည့်ရအောင် (`SyscoinVaultManager.sol` , lines 154-166)။

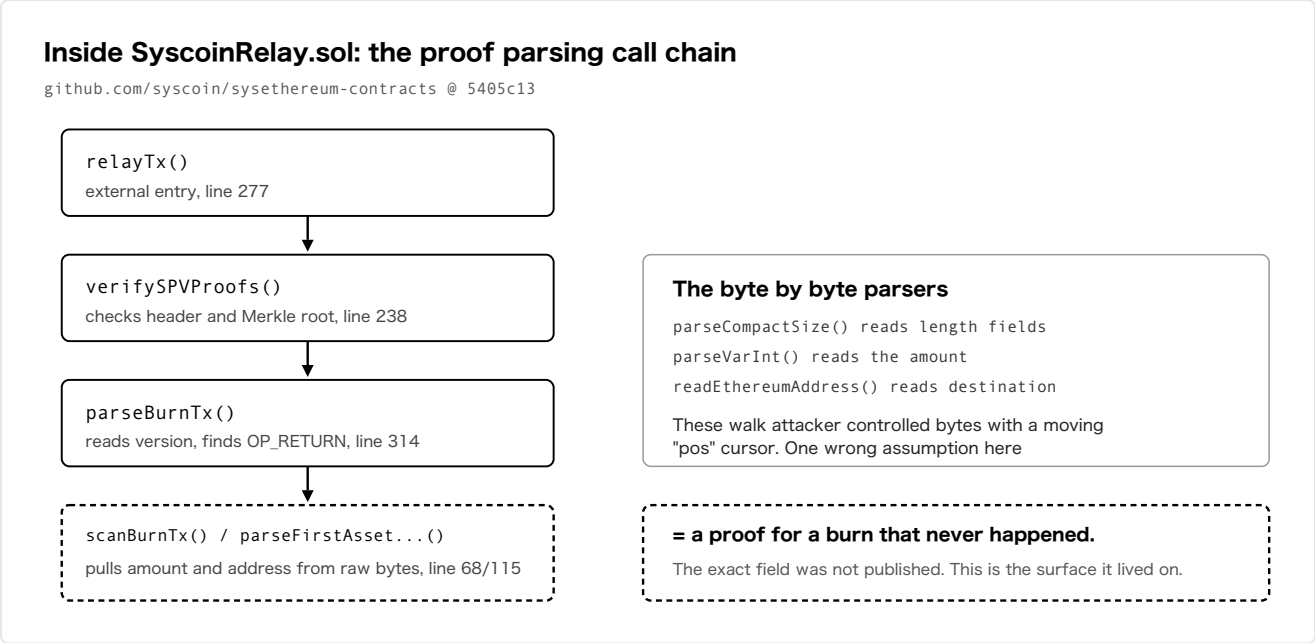
```
function processTransaction(
    uint txHash,
    uint value,
    address destination,
    uint64 assetGuid
) external override onlyTrustedRelayer nonReentrant whenNotPaused {
    require(_insert(txHash), "TX already processed");

    if (assetGuid == SYSAssetGuid) {
        require(value > 0, "Value must be positive");
        uint mintedAmount = scaleFromSatoshi(value, 18);
        _withdrawSYS(mintedAmount, payable(destination));   // the mint happens here
    }
    ...
}
```

`onlyTrustedRelayer` ဆိုတဲ့ Modifier ကို သတိထားကြည့်ပါ။ Vault က Burn တကယ်ဖြစ်ခဲ့လား ဆိုတာကို **ကိုယ်တိုင် ပြန်မစစ်ဘူး**။ Relay ပြောတာကို လုံးဝ ယုံတယ်။ ဒါက ဒီဇိုင်းကြောင့်ပါ။ ဘာလို့လဲဆိုတော့ Relay ကိုယ်တိုင်က Burn ကို prove လုပ်ရမယ့်သူ ဖြစ်လို့ပါ။ ဒါပေမယ့် ဒါက Mint ရဲ့ security တစ်ခုလုံးက မေးခွန်းတစ်ခုပေါ်မှာ မှီခိုနေတယ်လို့ ဆိုလိုတာပါ။ **Relay ရဲ့ Proof Reader ကို fooled လုပ်လို့ ရနိုင်လား?**

ဖတ်တဲ့ အလုပ်က `parseBurnTx()` ထဲမှာ ဖြစ်ပြီး သူက Byte တွေကို လျှောက်လှမ်းဖတ်တဲ့ Helper အသေးစား စုံတစ်ခုကို ခေါ်သုံးတယ် (`SyscoinRelay.sol` , lines 314-340)။

```
function parseBurnTx(bytes memory txBytes)
    public pure
    returns (uint errorCode, uint output_value, address destinationAddress, uint64 assetGuid)
{
    uint32 version;
    uint pos = 0;
    uint opIndex = 0;
    version = bytesToUint32Flipped(txBytes, pos);
    if (version != SYSCOIN_TX_VERSION_ALLOCATION_BURN_TO_NEVM) {   // must be ver 141
        return (ERR_PARSE_TX_SYS, 0, address(0), 0);
    }
    (opIndex, pos) = getOpReturnPos(txBytes, 4);                   // find the OP_RETURN
    (output_value, destinationAddress, assetGuid) = scanBurnTx(    // read amount + address
        txBytes, opIndex, pos
    );
    return (0, output_value, destinationAddress, assetGuid);
}
```

ဒီမှာ တန်ဖိုးတိုင်း (ပမာဏ၊ Destination၊ Asset ID) ကို `txBytes` ထဲက တိုက်ရိုက် ဖတ်တာပါ။ အဲ့ဒီ `txBytes` ဆိုတာ **ခေါ်တဲ့သူ ကိုယ်တိုင် ပေးလိုက်တဲ့** Data ပါ။ `pos` ဆိုတဲ့ Cursor တစ်ခုက Byte Array ထဲမှာ ရွှေ့သွားနေတယ်။ `parseCompactSize()` က length ကို ဖတ်တယ်၊ `parseVarInt()` က number ကို ဖတ်တယ်၊ `readEthereumAddress()` က Address အတွက် Byte 20 လုံး ဖတ်တယ်။ ဒါက Bitcoin transaction ရဲ့ raw data ကို Solidity နဲ့ ကိုယ်တိုင် (manually) parse လုပ်ထားတာ ဖြစ်ပါတယ်။



Attacker ပို့လိုက်တဲ့ transaction bytes တွေကို parser က byte တစ်ခုချင်းဆီ ဖတ်ပြီး "ဘယ်လောက် burn လုပ်ထားလဲ" နဲ့ "ဘယ် address ကို ပို့ရမလဲ" ဆိုတာ လုပ်ပါတယ်။ Vulnerability က ဒီ parsing ရဲ့ တစ်နေရာရာမှာ ရှိခဲ့ပေမယ့် ဘယ် field အတိအကျလဲဆိုတာကိုတော့ public မထုတ်ပြန်ထားပါဘူး။

အကြောင်းရင်း (Root Cause)

ဒီနေရာမှာ စနစ်က အမြဲတမ်း ထိန်းသိမ်းထားရမယ့် **invariant (မပျက်မကွက် မှန်ကန်နေရမယ့် စည်းမျဉ်း)** ကတော့ ဒီလိုပါ။

Proof တစ်ခုကို လက်ခံမယ်ဆိုရင် အဲဒီ proof ထဲက bytes တွေဟာ အမှန်တကယ် burn လုပ်ထားတဲ့ transaction တစ်ခုကို ကိုယ်စားပြုရမယ်၊ ပြီးတော့ အဲဒီ transaction ဟာ blockchain ပေါ်က block အစစ်တစ်ခုထဲမှာ အမှန်တကယ် mined ဖြစ်ပြီးသား ဖြစ်ရမယ်။

အဲဒီ proof ကို အမှန်တကယ် valid ဖြစ်တယ်လို့ သတ်မှတ်နိုင်ဖို့ **သီးခြား အရေးကြီးတဲ့ အချက် ၂ ချက်** မှန်ရပါမယ်။

ပထမအချက် က transaction ဟာ blockchain ရဲ့ block အစစ်တစ်ခုထဲမှာ အမှန်တကယ် ပါဝင်နေကြောင်း သက်သေပြနိုင်ရမယ်။ ဒါကို **Merkle proof နဲ့ block header verification** က စစ်ဆေးပေးပါတယ်။

ဒုတိယအချက် က transaction bytes တွေကို parse လုပ်တဲ့အခါ network ပေါ်က node တွေအားလုံး နားလည်တဲ့ အဓိပ္ပါယ်နဲ့ တူညီတဲ့ အဓိပ္ပါယ် ထွက်လာရမယ်။ ဒါကို **parsing check** လို့ ခေါ်ပါတယ်။

ဒီနှစ်ခုဟာ **မတူညီတဲ့ security guarantee နှစ်မျိုး** ဖြစ်ပါတယ်။ Bridge တစ်ခု လုံခြုံဖို့ဆိုရင် **နှစ်ခုလုံး လိုအပ်ပါတယ်။**

Syscoin Relay က ပထမတစ်ခုကို မှန်မှန်ကန်ကန် လုပ်ထားတယ်။ `verifySPVProofs()` က Block Hash ကို Precompile တစ်ခုကနေ ပြန်တွက်ပြီး Merkle Root ကို ပြန်လုပ်ပြီး စစ်ပါတယ်။ ဒါကြောင့် blockchain ထဲမှာ အမှန်တကယ် မရှိတဲ့ transaction တစ်ခုကို proof အဖြစ် တင်ပြဖို့ မဖြစ်နိုင်ပါဘူး။ Syscoin team နဲ့ Halborn

တို့ အတည်ပြုထားသလို vulnerability က **bridge relay ရဲ့ proof validation parsing logic** ထဲမှာ ဖြစ်ခဲ့တာပါ။ တစ်နည်းအားဖြင့် ပြဿနာက **ဒုတိယ security guarantee (parsing check)** မှာ ဖြစ်တာပါ။

rekt.news က ဒီဖြစ်ရပ်ကို အခုလို ရှင်းပြထားပါတယ်။

"Attacker က valid proof ကို forge လုပ်ခဲ့တာ မဟုတ်ဘူး။ Parsing code က valid proof လို့ ထင်အောင် လုပ်ထားတဲ့ data ကို forge လုပ်ခဲ့တာ ဖြစ်တယ်။"

ရိုးရိုးပြောရရင်၊ attacker က Byte blob တစ်ခုကို ဖန်တီးလိုက်တယ်။ Relay ရဲ့ handwriting Parser က အဲ့ဒီ bytes တွေကို ဖတ်တဲ့အခါမှာ "Burn Amount = 5,000,000,000 SYS, Destination = Attacker Address" လို့ ထင်သွားတယ်။ တကယ်တန်းတော့ Burn Amount = 0 ဒါမှမဟုတ် Burn လုပ်ခဲ့တာမျိုး မရှိခဲ့ပါဘူး။ Symptom က NEVM ပေါ်SYS 5 ဘီလီယံ ပေါ်လာတာပါ။ အကြောင်းအရင်း (Cause) ကတော့ Parser ရဲ့ "ဒီ Byte တွေက ဘာကို ဆိုလိုလဲ" ဆိုတဲ့ နားလည်မှုက reality နဲ့ match မဖြစ်ခဲ့တာပါ၊ ပြီးတော့ နောက်က ဘယ် Component ကမှ ဒါကို ထပ်မစစ်ခဲ့တာပါ။

ဒါက Bridge Bug တွေရဲ့ ထုံးစံအတိုင်း ပါပဲ။ **Nomad** Hack (2022) နဲ့ တူပါတယ် (Initialization မှားတာကြောင့် Bridge က Message မှန်သမျှကို သက်သေပြပြီးသား အဖြစ် မှတ်လိုက်တာမျိုးပါ)။ ပြီးတော့ **BNB Chain** Bridge Hack (2022) (Attacker က forged IAVL Merkle proof ကို ဖန်တီးနိုင်ခဲ့ပြီး Bridge က အဲဒီ forged proof ကို valid အဖြစ် လက်ခံခဲ့တာမျိုး) နဲ့လည်း တူတယ်။ Chain မတူ၊ Code မတူပေမယ့် root cause က အတူတူပါပဲ။ Proof verifier က ကိုယ်တိုင် ပြန်တွက်ချက်ပြီး အတည်ပြုသင့်တဲ့ အချက်တစ်ခုကို မျက်စိမှိတ် ယုံကြည်ခဲ့တာ ဖြစ်ပါတယ်။

Syscoin မဖြစ်ခဲ့တဲ့ vulnerability တစ်ခုအကြောင်းလည်း ပြောချင်တယ်။ Syscoin relay ဟာ Bitcoin SPV implementations တွေမှာ နာမည်ကြီးတဲ့ **64-byte transaction Merkle attack** ကို ကာကွယ်ထားပြီးသား ဖြစ်ပါတယ်။ **verifyTx() function** ထဲမှာ ဒီလို logic မျိုး ပါတယ်။

```
// inside verifyTx()
if (tx.length == 64) {
    revert();
}
```

transaction length = 64 bytes ဖြစ်ရင် reject လုပ်ပါတယ်။ ဒါကြောင့် vulnerability က **64-byte transaction attack** မဟုတ်ပါဘူး (line 224)။ အဲ့တာကြောင့် vulnerability က အပေါ်မှာ ပြထားတဲ့ transaction body parsing လုပ်ငန်းစဉ် (parsing surface) ရဲ့ တစ်နေရာရာမှာ ဖြစ်ခဲ့တာ ဖြစ်ပါတယ်။ Vulnerability ဖြစ်စေခဲ့တဲ့ field အတိအကျကို ထုတ်ဖော်ပြောကြားထားခြင်း မရှိတဲ့အတွက်၊ ကျွန်တော်က byte တစ်ခုခု သို့မဟုတ် code တစ်ခုကို ခန့်မှန်းပြီး အမှန်ပါလို့ မပြောလိုပါဘူး။ အဲဒီလို လုပ်လိုက်ရင်လည်း public record တွေက မထောက်ခံထားတဲ့ "ဘယ် line မှာ bug ရှိတယ်" ဆိုတဲ့ အဖြေကို ကိုယ်တိုင် ဖန်တီးပြောဆိုရာ ရောက်သွားမှာ ဖြစ်ပါတယ်။

တိုက်ခိုက်မှု အဆင့်ဆင့် (Exploit Walkthrough)

တကယ့် On Chain မှာ ဘယ်လို ဖြစ်သွားလဲ ကြည့်ရအောင်။ အောက်မှာ ပြထားတဲ့ Hash တိုင်းက public Postmortem ထဲက ဖြစ်ပါတယ်။

1. Mint လုပ်တာ။ attacker က သူ ဖန်တီးထားတဲ့ Proof ကို Bridge ဆီ တင်လိုက်တယ်။ Relay က လက်ခံပြီး `processTransaction` ကို ခေါ်လိုက်တယ်၊ SYS 5 ဘီလီယံလောက်က UTXO ဘက်မှာ attacker ထိန်းချုပ်တဲ့ Address ဆီ Mint ဖြစ်သွားတယ်။

- Mint tx: `a5b422abbbd89c8e316d1990f696e030d610cb527001ff97524f5317e87fa184`
- Attacker address: `sys1qgaelv690g7wwp2xchfdh0enf5uewzq5sm9wvcw`

2. ရွှေ့တာ။ အသစ် Mint ဖြစ်လာတဲ့ Coin တွေကို Mint လုပ်တဲ့ Address ကနေ ထုတ်ရွှေ့လိုက်တယ်။

- Transfer tx: `ba6798fac98eaf95f18e4622a6d46b5d8547f75d3912ed3665ee2e12537d5ff4`

3. ခွဲထုတ်တာ။ attacker က ရလာတဲ့ ပိုက်ဆံကို Output အများကြီးအဖြစ် ခွဲထုတ်လိုက်တယ်။ ဒါက Freeze လုပ်တာ၊ Trace လုပ်တာ ခက်အောင် လုပ်တဲ့ နည်းလမ်းတစ်ခုပါ။

- Split tx: `31e12b0dcd9aeffa12e596e0b16d75ce161667104c7e511bfafe67195117113c`
- အကြီးဆုံး tainted balance နှစ်ခုက ဒီ Address တွေဆီ ရောက်သွားတယ်။
  - `sys1q2k482wnachkgky4lw60973p4vcf7xlh9kzpv33` (SYS 4 ဘီလီယံ ခန့်)
  - `sys1qx6jjkq89sdaxftfgre3m0nv7vjfd4jeakg5t38` (SYS 1 ဘီလီယံ ခန့်)

ဒီမှာ **Flash Loan မရှိ၊ Setup အရင်းအနှီးလည်း မရှိ** ခဲ့ပါဘူး။ Price Manipulation Exploit တွေနဲ့ ယှဉ်ရင် Proof Validation Bug က ဘာလို့ အန္တရာယ်ကြီးလဲဆိုတာ ဒါက ပြတာပါ။ attacker ဘာမှ မလိုဘူး။ "Setup" တစ်ခုလုံးက Off Chain မှာ ဖြစ်တယ်။ ဆိုလိုတာက Malformed Proof Byte တွေ ဖန်တီးတာပါ။ Proof accept ဖြစ်တာနဲ့ attacker က mint လုပ်လို့ ရသွားပါပြီ။

Proof of Concept အကြောင်း ပြောရရင်၊ PoC ကို မထည့်ချင်ပါဘူး။ ဘာလို့လဲဆိုတော့ vulnerability ကို အသုံးချနိုင်တဲ့ အဓိက လက်နက် (weapon) ဟာ **malformed proof ကို ဘယ်လို တည်ဆောက်ရမလဲဆိုတဲ့ အသေးစိတ်နည်းလမ်း** ဖြစ်နေလို့ပါ။ Responsible disclosure ဆိုတာ bridge ရဲ့ အသေးစိတ်အချက်အလက်တွေ public ဖြစ်နေပြီဆိုရင်တောင် exploit ကို copy paste လုပ်ပြီး တိုက်ရိုက်အသုံးချနိုင်တဲ့ code ကို မထုတ်ပြန်ခြင်းကို ဆိုလိုပါတယ်။ ဖော်ပြနိုင်တာကတော့ call structure (call shape) ပဲ ဖြစ်ပါတယ်။

အဲဒါက `relayTx(...)` call တစ်ခုတည်း ဖြစ်ပြီး၊ အဲဒီ call ထဲမှာ

- block header
- transaction body
- Merkle siblings

တို့ ပါဝင်ပါတယ်။ ပြီးတော့ transaction body ကို parser က ဖတ်ပြီး attacker စိတ်ကြိုက် သတ်မှတ်ထားတဲ့ amount နဲ့ destination address အဖြစ် ပြောင်းလဲ နားလည်သွားပါတယ်။ ဒါက exploit ရဲ့ အခြေခံ structure ဖြစ်ပါတယ်။ ဒါပေမယ့် parser ကို fooled လုပ်ပြီး validation ကို ကျော်ဖြတ်သွားနိုင်တဲ့ **bytes အတိအကျ** ကတော့ အန္တရာယ်ရှိတဲ့ အစိတ်အပိုင်း ဖြစ်တဲ့အတွက် public ထုတ်ပြလို့ မရပါဘူးဗျ။

ဆုံးရှုံးမှု (Impact)

- **အန္တရာယ်ရှိတဲ့ / Mint ဖြစ်တဲ့ ပမာဏ:** SYS 5,000,000,000 ခန့်၊ Mint လုပ်ချိန်မှာ $8.5M လောက်။
- **Supply ထိခိုက်မှု:** အဲ့ဒီ Mint က Circulating Supply ကို 568% လောက် Inflation ဖြစ်စေပြီး Attack ပြီးနောက် Supply ရဲ့ 85% လောက် ရှိသွားတယ်။ သတင်းကြောင့် SYS ဈေး 20% လောက် ကျသွားတယ်။
- **ဘယ်သူတွေ ထိခိုက်လဲ:** Dilution နဲ့ ဈေးကျတာကြောင့် SYS ကိုင်ထားသူ တိုင်း၊ ပြီးတော့ tainted Coin တွေ ဝင်သွားနိုင်တဲ့ Exchange ဒါမှမဟုတ် Pool တွေ။
- **Preconditions:** attacker က Public ဖြစ်တဲ့ `relayTx` Entry Point ကို ဖန်တီးထားတဲ့ Proof တစ်ခုနဲ့ ခေါ်နိုင်ဖို့ပဲ လိုတယ်။ special access မလို၊ Key မလို၊ အရင်းအနှီး မလို။
- **အခု Exploit လုပ်လို့ ရသေးလား:** မရတော့ပါဘူး။ Bridge ကို နာရီပိုင်းအတွင်း ရပ်လိုက်ပြီး၊ Validation Path ကို Bridge ပြန်မဖွင့်ခင် ပြင်ပြီးသားပါ။

ပျက်စီးမှု (damage) ဟာ **"mint လုပ်ပြီး ထိန်းချုပ်နိုင်ခဲ့တဲ့ အဆင့်"** မှာပဲ ရပ်တန့်သွားပြီး **"mint လုပ် ငွေထုတ်ပြီး အမြတ်ထုတ်သွားတဲ့ အဆင့်"** အထိ မရောက်ခဲ့ရတဲ့ အကြောင်းရင်းကတော့ တို့ပြန်မှု မြန်ဆန်ခဲ့လို့ ဖြစ်ပါတယ်။ Tainted SYS token အများစုကတော့ open market တွေဆီကို မရောက်ရှိခဲ့ပါဘူး။

The Fix

Syscon က vulnerability ကို ပြင်ဆင်ထားတဲ့ code diff ကို line အဆင့်အထိ အသေးစိတ် မထုတ်ပြန်ထားတဲ့အတွက်၊ ကျွန်တော်လည်း မရှိတဲ့ အချက်အလက်ကို ခန့်မှန်းပြီး ဖန်တီးမပြောလိုပါဘူး။ Public statement တွေအရ Fix မှာ အပိုင်း နှစ်ပိုင်း ရှိပါတယ်။

1. **အရင်ဆုံး ရပ်လိုက်တာ။** `processTransaction` က `whenNotPaused` Modifier အောက်မှာ Run တာဖြစ်လို့၊ Bridge ကို ရပ်လိုက်တာနဲ့ နောက်ထပ် Mint တွေ ချက်ချင်း ရပ်သွားတယ်။ ဒါက Bridge ရဲ့ အရေးပေါ် ဘရိတ် ဆွဲလိုက်တာနဲ့ တူပါတယ်။
2. **Validation Path ကို ပြုပြင်တာ။** Team က Relay ရဲ့ Proof Validation Logic (အပေါ်က ဖော်ပြထားတဲ့ Parsing surface) ကို Fix တစ်ခု လုပ်ပြီး Review လုပ်ခဲ့တယ်။ ဒါက Malformed Proof တစ်ခုကို Valid Burn အဖြစ် ဖတ်လို့ မရတော့အောင်ပါ။

ဒီ Bug အမျိုးအစားအတွက် မှန်ကန်တဲ့ Fix ရဲ့ ပုံစံက strict ဖြစ်ပြီး ပြည့်စုံတဲ့ Parsing ပါ။ Verifier က Byte တွေ တိကျတဲ့ အဓိပ္ပါယ် တစ်ခုတည်း ဖြစ်အောင် Parse မဖြစ်တဲ့ Transaction မှန်သမျှကို ပယ်ရမယ်၊ length နဲ့

Offset တိုင်းကို Bounds Check လုပ်ရမယ်၊ ပြီးတော့ caller ပေးတဲ့ Layout ကို မယုံဘဲ တန်ဖိုးတိုင်းကို ကိုယ်တိုင် ပြန်တွက်ရမယ်။ သဘောကတော့ Before/After က ဒီလို ဖြစ်မယ်။

```
- // Merkle root ကိုက်ရင် proof လက်ခံ၊ ပြီးမှ field တွေ ဖတ်
- (value, dest, guid) = scanBurnTx(txBytes, opIndex, pos);
- processTransaction(txHash, value, dest, guid);

+ // bytes တွေ canonical burn တစ်ခုတည်းဖြစ်အောင် parse ဖြစ်မှသာ လက်ခံ၊
+ // field တိုင်းကို strict bounds နဲ့၊ ကျန် bytes / မရှင်းတဲ့ bytes မရှိစေဘဲ
+ require(isCanonicalBurnTx(txBytes), "not canonical tx");
+ (value, dest, guid) = scanBurnTxStrict(txBytes, opIndex, pos);
+ processTransaction(txHash, value, dest, guid);
```

ဒါက Root Cause (မရှင်းတဲ့ Parsing) ကို ဖြေရှင်းတာ ဖြစ်ပြီး၊ Symptom (တိကျတဲ့ Malformed Input တစ်ခု) ကို ဖြေရှင်းတာ မဟုတ်ဘူး။ attacker သုံးခဲ့တဲ့ Byte ကိုပဲ ပြင်ရင် Symptom Fix ပဲ ဖြစ်မယ်။ မှန်ကန်တဲ့ နည်းလမ်းက Parser ကို တကယ် Burn ကြောင်း သက်သေမပြနိုင်တဲ့ အရာ မှန်သမျှကို ပယ်ခိုင်းတာပါ။

Takeaways

Bridge တွေ တည်ဆောက်တယ်၊ Audit လုပ်တယ်၊ ဒါမှမဟုတ် သုံးတယ်ဆိုရင်၊ ဒီ Pattern တွေ မှတ်ထားသင့်ပါတယ်။

- **Proof Verifier တစ်ခု ဘယ်လောက် strong လဲဆိုတာ သူ့ရဲ့ Parser ပေါ်မှာပဲ မူတည်တယ်။** Merkle root ကို ပြန်တွက်ပြီး စစ်ဆေးနိုင်တာ တစ်ခုတည်းနဲ့ မလုံလောက်ပါဘူး။ Transaction ရဲ့ အဓိပ္ပါယ်ကို ဖတ်တဲ့အဆင့်ကို fool လုပ်နိုင်မယ်ဆိုရင် security ပျက်စီးသွားနိုင်ပါတယ်။ ဒါကြောင့် **"ဒီ transaction က block ထဲမှာ တကယ်ပါဝင်သလား"** ဆိုတာ နဲ့ **"ဒီ transaction က ဘာကို ဆိုလိုတာလဲ"** ဆိုတာ စစ်တဲ့ နှစ်ခုစလုံးဟာ အမှားအယွင်းမရှိအောင် strong နေရပါမယ်။ Bridge တွေ ပျက်စီးရတဲ့ အကြောင်းရင်းကတော့ အများအားဖြင့် ဒုတိယအချက်မှာ ဖြစ်ပြီး၊ Nomad, BNB Chain နဲ့ အခု Syscoin တို့ဟာ အဲဒီ ဥပမာတွေပဲ ဖြစ်ပါတယ်။
- **Solidity ထဲမှာ raw bytes တွေကို ကိုယ်တိုင် (handwritten) parse လုပ်တာဟာ risk အလွန်မြင့်တဲ့ attack surface တစ်ခု ဖြစ်ပါတယ်။** Cursor ကို ရွှေ့ရင်း transaction bytes တွေကို တစ်ဆင့်ချင်းဖတ်တာ၊ variable-length fields တွေကို parse လုပ်တာ၊ offsets တွေကို ယုံကြည်တာ စတဲ့နေရာတွေမှာ edge cases တွေ ဖြစ်နေနိုင်ပါတယ်။ ဒါကြောင့် ဒီလို parser မျိုးတွေကို attacker-controlled input တွေ ဝင်လာမယ်လို့ အမြဲယူဆပြီး bounds checking တွေကို အလွန်ဂရုစိုက် လုပ်သင့်ပါတယ်။ Ambiguous ဖြစ်နိုင်တဲ့ input မှန်သမျှကို reject လုပ်နိုင်တဲ့ canonical parser တွေကို အသုံးပြုတာ ပိုကောင်းပါတယ်။
- **Trust boundary တစ်ခုတည်းကို မစစ်ဆေးဘဲ ယုံကြည်လိုက်မယ်ဆိုရင် အခြား security control တွေအားလုံးကို ပျက်စီးစေနိုင်ပါတယ်။** SyscoinVaultManager မှာ reentrancy guard, pause switch နဲ့ asset-specific checks တွေလို security mechanisms တွေ ရှိခဲ့ပေမယ့် relay ကိုတော့ အပြည့်အဝ ယုံကြည်ထားခဲ့ပါတယ်။ Relay က မှားသွားတဲ့အချိန်မှာတော့ အဲဒီ သေချာရေးထားတဲ့ code တွေအားလုံးက

attacker အလိုကျ 5 billion SYS ကို mint လုပ်ပေးတဲ့ အခြေအနေ ဖြစ်သွားခဲ့ပါတယ်။ Upstream မှာ weak link တစ်ခုရှိနေရင် downstream က strong code တွေဟာ အဓိပ္ပါယ်မရှိတော့ပါဘူး။

- **Audit လုပ်တဲ့အခါ contract တွေတင် မဟုတ်ဘဲ relay path လိုမျိုး သာမန်ထင်ရတဲ့ code တွေကိုလည်း သေချာစစ်ဆေးသင့်ပါတယ်။** Postmortem အရ wallet, governance နဲ့ အခြား component တွေမှာ third-party audits တွေ ရှိခဲ့ပေမယ့် bridge relay ရဲ့ proof validation path ကိုတော့ audit မလုပ်ထားခဲ့ပါဘူး။ အမြင်မှာ မထင်ရှားတဲ့ code က အန္တရာယ်အရှိဆုံး authority ကို ကိုင်ထားတဲ့ အစိတ်အပိုင်း ဖြစ်ခဲ့ပါတယ်။
- **Pause switch တွေနဲ့ exchanges တွေနဲ့ ပူးပေါင်းဆောင်ရွက်နိုင်ခြင်းဟာလည်း security ရဲ့ အစိတ်အပိုင်းတစ်ခု ဖြစ်ပါတယ်။** ဒီအရာတွေက exploit ကို မတားဆီးနိုင်ခဲ့ပေမယ့် "funds returned" ဆိုတာ ဖြစ်လာစေခဲ့ပြီး "funds gone forever" ဆိုတာမျိုး မဖြစ်အောင် ကာကွယ်ပေးခဲ့ပါတယ်။ Incident response ဆိုတာ security ရဲ့ အစိတ်အပိုင်းတစ်ခု ဖြစ်ပြီး နောက်မှ စဉ်းစားရမယ့် အရာ မဟုတ်ပါဘူး။

Disclosure Timeline

နေ့စွဲ	ဖြစ်ရပ်
2026 ဇွန် 7	Exploit လုပ်ဆောင်ခဲ့တယ်၊ SYS 5B ခန့် Mint ဖြစ်တယ်။ Bridge ကို နာရီပိုင်းအတွင်း ရပ်လိုက်တယ်။
2026 ဇွန် 7	Preliminary Postmortem ကို တနေ့ချင်း ညနေပိုင်းမှာ ထုတ်ပြန်တယ်၊ Transaction Hash တွေနဲ့ tainted Address တွေ ပါတယ်။
2026 ဇွန် 9	Public Whitehat Recovery Address တင်တယ်၊ တရားစွဲ ခြိမ်းခြောက်မယ့်အစား Bounty ကမ်းလှမ်းတယ်။
2026 ဇွန် 11	attacker က Funds တွေ ပြန်ပေးတာ confirmed ဖြစ်တယ်။
နောက်ရက်များ	Bridge ပြန်ဖွင့်ခင် Proof Validation Path ကို Fix လုပ်ပြီး Review လုပ်တယ်။

Sources

- rekt.news, Syscoin Rekt (အဓိက Writeup၊ Transaction Hash များ၊ Address များ)
- Halborn, Explained: The Syscoin Bridge Hack (June 2026)
- BeInCrypto, Syscoin pauses bridge after 5 billion SYS exploit
- CryptoPotato, SYS drops 20% after 5B unauthorized tokens minted
- Syscoin bridge contracts source (`sysethereum-contracts`)
- Syscoin Bridge Dapp repository
- Syscoin documentation

- Syscoin postmortem (X/Twitter) နဲ့ recovery confirmation
- တူညီတဲ့ Bug family background: Nomad Bridge hack (2022)၊ BNB Chain bridge hack (2022)

Written by Khant Wai Yan Aung