Gravity Bridge : Ethereum နဲ့ Cosmos Bridge မှ ဒေါ်လာ ၅.၄ သန်း ဆုံးရှုံးစေခဲ့သော ခိုးယူခံရသည့် Signing Key ဖြစ်ရပ်

လူအများထင်နေကြသလို "Denom Mapping Bug" မဟုတ်ခဲ့ရသည့် အကြောင်း

အကျဉ်းချုပ် (Summary)

ဒီ Incident မှာ ဘာတွေဖြစ်ခဲ့တယ်ဆိုတာကို အရင် အကျဉ်းချုပ် အရင်ပြောချင်ပါတယ်။

Protocol နဲ့ Commit

Gravity Bridge က Ethereum နဲ့ Cosmos Ecosystem ကို ချိတ်ပေးထားတဲ့ Cross-Chain Bridge တစ်ခုပါ။ Ethereum ဘက်က Main Bridge Contract Address က `0xa4108aA1Ec4967F8b52220a4f7e94A8201F2D906` ဖြစ်ပြီးတော့ Etherscan မှာ "Gravity Bridge: Bridge" အဖြစ် Verify လုပ်ထားပါတယ်။ အခု Blog post ရဲ့ ကိုးကားထားတဲ့ Code အားလုံးကို `althea-net/cosmos-gravity-bridge` Repository ရဲ့ Commit `92d0e12cea813305e6472851beeb80bd2eaf858d` ကို အခြေခံထားပါတယ်။

Vulnerability အမျိုးအစား

ဒီ Incident က Smart Contract Logic Bug ကြောင့် ဖြစ်ခဲ့တာ မဟုတ်ပါဘူး။ Root Cause ကတော့

- Access Control Failure
- Key Management Failure

တို့နဲ့ ဆိုင်ပါတယ်။ တစ်စုံတစ်ယောက်က Bridge ရဲ့ Signing Key ကို control လုပ် နိုင်ခဲ့ပြီးတော့ Authorization Process ကို အသုံးချပြီး Asset တွေ ထုတ်ယူသွားနိုင်ခဲ့တာပါ။ အဲ့တာကြောင့် ဒီ Incident ကို "Denom Mapping Bug" လို့ပြောလို့မရနိုင်ပါဘူး။

ဆုံးရှုံးမှု (Impact)

Attack ကြောင့် စုစုပေါင်း ဒေါ်လာ ၅.၄ သန်းခန့် ဆုံးရှုံးခဲ့ရပါတယ်။ ခိုးယူခံရတဲ့ Asset တွေက

- USDC – ခန့်မှန်းအားဖြင့် $4.3 Million
- ETH နှင့် WETH 274 ETH – ခန့်မှန်းအားဖြင့် $553,000
- USDT – ခန့်မှန်းအားဖြင့် $434,000
- PAXG 14.164 Tokens – ခန့်မှန်းအားဖြင့် $64,000

တို့ ဖြစ်ပါတယ်။

လက်ရှိအခြေအနေ (Status)

ဒီ Incident က 2026 ခုနှစ် မေလ 30 ရက်နေ့မှာ တကယ် ဖြစ်ခဲ့တဲ့ Real-World Security Incident ဖြစ်ပါတယ်။ Attack ဖြစ်ပြီးနောက်မှာ Bridge ကို ရပ်ခဲ့ရပါတယ်။ ခိုးယူထားတဲ့ Funds တစ်ချို့ကို ChangeNOW, Binance တွေကတစ်ဆင့် Cash Out ပြုလုပ်ခဲ့ပြီး တစ်ချို့ကို Tornado Cash ကနေ တစ်ဆင့် လမ်းကြောင်းဖျောက်ဖို့ သုံးခဲ့ကြပါတယ်။ အခုလက်ရှိ report ရေးတဲ့အချိန်ဖြစ်တဲ့ 2026 ခုနှစ် ဇွန်လ 11 ရက်နေ့အထိ Project Team ဘက်မှ Official Postmortem Report ကို မထုတ်ပြန်ရသေးပါဘူး။

အရေးကြီးသော အချက်တစ်ခု

ဒီ Incident ကို မလေ့လာခင် ရှင်းပြဖို့ လိုအပ်တဲ့ အချက်တစ်ခု ရှိပါတယ်။ လူအများစုက ဒီ Attack ကို "Denom Mapping Exploit" လို့ ခေါ်နေကြပါတယ်။ ဒါပေမယ့် လက်ရှိအချိန်အထိ ရထားတဲ့ Evidence တွေ အရတော့ တကယ် ဖြစ်ခဲ့တာက အဲ့ဒီလို မဟုတ်ပါဘူး လို့ပြောချင်ပါတယ်။ Funds တွေ ဆုံးရှုံးခဲ့ရတဲ့ အဓိက အကြောင်းရင်းကတော့ Bridge ကို ထိန်းချုပ်နိုင်တဲ့ Signing Key ကို တစ်ယောက်ယောက်က ရသွားဖို့ဖြစ်ပါတယ်။ တစ်နည်းအားဖြင့်တော့ Code မှာ Bug ရှိလို့ Hack ခံရတာ မဟုတ်ဘဲနဲ့ Bridge ရဲ့ Authorization Mechanism ကြောင့် ဖြစ်ပါတယ်။ ဒါက Key Management Failure ဖြစ်ပြီးတော့ Smart Contract Logic Flaw မဟုတ်ပါဘူး။

ဒါဆို Denom Mapping Bug ဆိုတာ ဘာလဲ?

Denom Mapping နဲ့ ပတ်သက်တဲ့ Vulnerability တွေက တကယ်ရှိခဲ့ပါတယ်။ ဒါပေမယ့် အဲ့ဒီ Vulnerability တွေက 2020 နဲ့ 2021 အတွင်းမှာ Audit တွေနဲ့ Research တွေမှာ တွေ့ခဲ့တဲ့ သီးခြား Security Issue များ ဖြစ်ပါတယ်။ အဆိုပါ Historical Vulnerability များနဲ့ 2026 ခုနှစ်ရဲ့ $5.4 Million Incident အကြားမှာတော့ တိုက်ရိုက် ဆက်စပ်မှု ရှိတယ် ဆိုတဲ့ သက်သေ မတွေ့ရသေးပါဘူး။ အဲ့တာကြောင့် 2026 Gravity Bridge

Incident နဲ့ Historical Denom Mapping Vulnerabilities ကို အတူတူပဲလို မထင်စေချင်ပါဘူး။ အခု blog post မှာ အဲ့ဒီ အကြောင်းအရာ နှစ်ခုကို သပ်သပ်စီခွဲပြီး ရှင်းရှင်း လင်းလင်း ရှင်းပြပေးသွားပါမယ်ဗျ။

Background

Cross-Chain Bridge ဆိုတာက တစ်ခုနှင့်တစ်ခုကို တိုက်ရိုက် ဆက်သွယ်နိုင်စွမ်း မရှိတဲ့ Blockchain နှစ်ခု ကြား Asset တွေကို လွှဲပြောင်းနိုင်အောင် ပြုလုပ်ပေးထားတဲ့ Infrastructure တစ်ခု ဖြစ်ပါတယ်။ Gravity Bridge က Ethereum နဲ့ Cosmos Ecosystem ကို ချိတ်ပေးထားတဲ့ Bridge တစ်ခု ဖြစ်ပြီးတော့ Ethereum ပေါ်မှာရှိတဲ့ Asset တွေကို Cosmos Network ထဲ ပြောင်းရွှေ့နိုင်အောင် လုပ်ပေးပါတယ်။ Bridge ရဲ့ အခြေခံ အလုပ်လုပ်ပုံကို Lock and Mint Model လို့ခေါ်ပါတယ်။ အလုပ်လုပ်ပုံတွေကို step တစ်ခုချင်းစီအလိုက် ရှင်း ပြပေးပါမယ်။

Step 1: Ethereum ဘက်မှာ Token ကို Lock လုပ်ခြင်း

ဥပမာအားဖြင့် User တစ်ယောက်က USDC ကို Cosmos Network ဆီ ပို့ချင်တယ်ဆိုပါစို့။ အဲ့ဒီ User က USDC Token တွေကို Gravity Bridge ရဲ့ Ethereum Contract ထဲကို Deposit လုပ်ရမယ်။ Deposit လုပ် လိုက်တဲ့ Token တွေကို Bridge Contract က Lock လုပ်ထားမှာ ဖြစ်ပြီးတော့ User အနေနဲ့ တိုက်ရိုက် အသုံးပြုနိုင်တော့မှာ မဟုတ်ပါဘူး။

Step 2: Validators တွေက Deposit ကို အတည်ပြုခြင်း

Cosmos Network ပေါ်မှာ လုပ်ဆောင်နေတဲ့ Validator တွေက Ethereum Blockchain ကို အမြဲစောင့်ကြည့် နေကြပါတယ်။ User ရဲ့ Deposit Transaction ကို Validator အများစုက မြင်တွေ့ပြီးတော့ အတည်ပြုတဲ့ အခါ Deposit လုပ်ခဲ့ကြောင်း Consensus ရရှိသွားပါတယ်။ ပြီးတော့ Validators တွေက Deposit ကို အတည်ပြုတဲ့ Signature တွေကို ထုတ်ပေးကြတယ်။

Step 3: Cosmos ဘက်မှာ Matching Token ကို Mint လုပ်ခြင်း

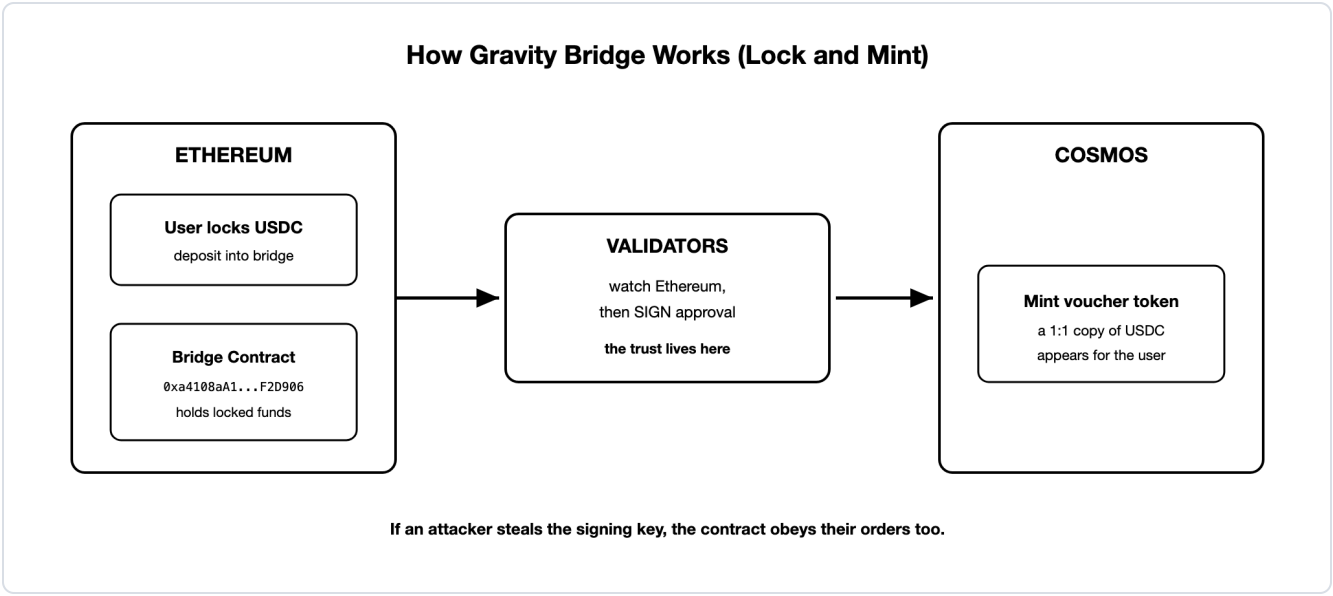
Validator တွေရဲ့ Approval ရပြီးတော့ Cosmos Chain က Deposit လုပ်ထားတဲ့ Token နဲ့ တန်ဖိုးညီမျှတဲ့ Representation Token ကို User အတွက် Mint လုပ်ပေးပါတယ်။ ဥပမာ Ethereum ဘက်မှာ USDC 100 Deposit လုပ်ခဲ့ရင် Cosmos ဘက်မှာ USDC 100 နဲ့ ညီမျှတဲ့ Bridge Token 100 ကို User ရမှာဖြစ်ပါတယ်။

Cosmos ကနေ Ethereum ကို ပြန်ထုတ်လိုတဲ့အခါ

Asset ကို Ethereum ဘက် ပြန်ယူချင်ရင် Process က ပြောင်းပြန် ပါ။

- User က Cosmos ဘက်မှာရှိတဲ့ Token ကို Burn လုပ်ရမယ်။
- Validators တွေက Burn Event ကို အတည်ပြုရမယ်။
- Validator Signatures လုံလုံလောက်လောက် ရရှိပြီးနောက် Ethereum ဘက်ရှိတဲ့ Locked Asset တွေကို Release ပြန်လုပ်ပေးရမှာ ဖြစ်ပါတယ်။

အဲ့တာကြောင့် Gravity Bridge ရဲ့ Security Model တစ်ခုလုံးက Validator တွေရဲ့ Signing Process နဲ့ Authorization Mechanism အပေါ်မှာ အများကြီး မူတည်နေပါတယ်။ Signing Infrastructure သို့မဟုတ် Validator Authorization System တစ်ခုခု Compromise ဖြစ်သွားရင် Bridge အတွင်းမှာရှိတဲ့ Asset တွေ အားလုံး အန္တရာယ်ရှိလာနိုင်ပါတယ်။



ဒီ Security Model တစ်ခုလုံးက အရေးကြီးတဲ့ ယုံကြည်မှုတစ်ခုအပေါ် အခြေခံထားပါတယ်။ Transfer တစ်ခုကို Approve လုပ်နိုင်တာဟာ တရားဝင် Validator တွေရဲ့ Private Keys တွေသာ ဖြစ်ရမယ် ဆိုတဲ့ ယုံကြည်ချက် ဖြစ်ပါတယ်။ Ethereum ဘက်မှာရှိတဲ့ Bridge Contract က Asset တစ်ခုကို Release မလုပ်ခင် Validator တွေရဲ့ Signature တွေကို စစ်ဆေးပြီး အတည်ပြုပေးပါတယ်။ Signature တွေ မှန်ကန်ပြီးတော့ လိုအပ်တဲ့ Approval Threshold ပြည့်မီတယ်ဆိုရင် Contract က Transfer ကို Execute လုပ်ပေးပါတယ်။

ဒါပေမယ့် Attackers တွေက Validator Signing Keys တွေကို ရယူနိုင်ခဲ့တယ် ဆိုရင် အခြေအနေက လုံးဝ ပြောင်းလဲသွားပါလိမ့်မယ်။ Contract ရဲ့ အမြင်မှာ

- Signature တွေက Valid ဖြစ်နေတယ်။
- Approval က လုံလောက်တယ်။

- Authorization Process ကလည်း မှန်ကန်နေတယ်။

အဲ့တာကြောင့် Contract က အဆိုပါ Request တွေကို Legitimate Transaction များအဖြစ် ယုံကြည်ပြီးတော့ Execute လုပ်ပေးမှာပါ။

တစ်နည်းအားဖြင့် လွယ်လွယ် ပြောရရင် Contract ကို လှည့်စားတာမျိုး မဟုတ်ပါဘူး။ Contract က ယုံကြည်ထားတဲ့ Signing Authority ကို ထိန်းချုပ်နိုင်ခဲ့တာ ဖြစ်ပါတယ်။ ဒီအချက်ကို မှတ်ထားပေးပါ။ ဘာကြောင့်လဲဆိုတော့ Gravity Bridge Incident တစ်ခုလုံးကို အကျဉ်းချုပ်ရမယ်ဆိုရင် "Attackers တွေက Bridge ရဲ့ Signing Authority ကို ရယူနိုင်ခဲ့ပြီးတော့ Contract က သူတို့ရဲ့ order တွေကို တရားဝင် Validator တွေရဲ့ order တွေဆိုလို့ ယုံကြည်ပြီးတော့ Execute လုပ်ပေးခဲ့တာပါ" လို့ ပြောနိုင်ပါတယ်။

Gravity Bridge ရဲ့ Architecture နဲ့ လုပ်ဆောင်ပုံ အသေးစိတ်ကို လေ့လာလိုပါက Official Documentation များကိုလည်း ကိုးကားဖတ်ရှုနိုင်ပါတယ် Gravity Bridge Official Website နဲ့ Gravity Bridge Documentation Repository။

Vulnerable Code

ဒီ Incident ကို နားလည်ဖို့အတွက် Code နဲ့ပတ်သက်ပြီး မတူညီတဲ့ Story နှစ်ခု ရှိနေတယ်ဆိုတာ သတိထားဖို့ လိုပါတယ်။ ဒါကြောင့် ရောထွေးမသွားအောင် သီးခြားခွဲပြီး ရှင်းပြပါမယ်။

Thread A: တကယ့် Attack ဖြစ်ပွားစေခဲ့တဲ့ အကြောင်းရင်း

2026 ခုနှစ် မေလ 30 ရက်နေ့က ဖြစ်ပွားခဲ့တဲ့ $5.4 Million Incident ဟာ Gravity Bridge ရဲ့ Solidity Smart Contract ထဲက Vulnerability တစ်ခုခုကို Exploit လုပ်ခဲ့တာ မဟုတ်ပါဘူး။ Bridge Contract က

- Validator Signatures ကို မှန်ကန်စွာ စစ်ဆေးခဲ့တယ်။
- Authorization Rules တွေကို မှန်ကန်စွာ လိုက်နာခဲ့တယ်။
- Withdrawal Process ကိုလည်း Design အတိုင်း Execute လုပ်ခဲ့တယ်။

တစ်နည်းအားဖြင့် Contract Logic က သူ့အလုပ်သူ မှန်မှန်ကန်ကန် လုပ်နေခဲ့တာ ဖြစ်ပါတယ်။ ဒါကြောင့် Reentrancy Bug၊ Access Control Bug သို့မဟုတ် Integer Overflow Bug တွေလို "ဒီ Function က Vulnerable ဖြစ်တယ်" ဆိုပြီး Code ထဲက Line တစ်ခုကို ထောက်ပြနိုင်တဲ့ အခြေအနေမှာ မရှိပါဘူး။

အမှန်တကယ် ပြဿနာ ဖြစ်ပွားခဲ့တဲ့ နေရာက Blockchain ပေါ်မှာ မဟုတ်ဘဲ Blockchain အပြင်ဘက်မှာ ဖြစ်ပါတယ်။ Bridge ကို ထိန်းချုပ်နိုင်တဲ့ Secret Signing Key ဟာ Attackers တွေရဲ့ လက်ထဲ ရောက်သွားခဲ့တာ

ဖြစ်ပါတယ်။ Contract ရဲ့ အမြင်ကနေ ကြည့်မယ်ဆိုရင်

- Signature တွေက Valid ဖြစ်နေတယ်။
- Approval တွေကလည်း လုံလောက်နေတယ်။

ဒါကြောင့် Withdrawal Request တွေကို Legitimate Transaction တွေအဖြစ် သတ်မှတ်ပြီး Funds တွေကို Release လုပ်ပေးလိုက်တာ ဖြစ်ပါတယ်။

ဒီ Incident မှာ "Vulnerable Function" တစ်ခုကို ပြသဖို့ မဖြစ်နိုင်တဲ့ အကြောင်းရင်းကလည်း ဒီနေရာမှာပဲ ဖြစ်ပါတယ်။ Failure ဖြစ်ခဲ့တာက Smart Contract Code ထဲမှာ မဟုတ်ဘဲ Signing Key ကို ဘယ်လို သိမ်းဆည်းထားသလဲ၊ ဘယ်လို ကာကွယ်ထားသလဲ ဆိုတဲ့ Operational Security နှင့် Key Management Process ထဲမှာ ဖြစ်ပါတယ်။ ဒီ Key Compromise က ဘယ်လို Protocol တစ်ခုလုံးကို ထိခိုက်စေခဲ့သလဲဆိုတာကို နောက်လာမယ့် Root Cause Section မှာ အသေးစိတ် ဆက်လက်ပြောပြပေးသွားပါမယ်။

Thread B: Denom Mapping Vulnerabilities များသည် အမှန်တကယ် ရှိခဲ့သော်လည်း 2026 Incident ဖြစ်စေခဲ့တဲ့ အကြောင်းရင်း မဟုတ်ခဲ့ပါ

"Denom Mapping Exploit" ဆိုတဲ့ အမည် ပေါ်ပေါက်လာရတဲ့ အကြောင်းရင်းက ဒီနေရာမှာ စတင်ပါတယ်။ Denom Mapping နဲ့ ပတ်သက်တဲ့ Vulnerability တွေဟာ တကယ်ရှိခဲ့ပါတယ်။ ဒါပေမယ့် အဲ့ဒီ Vulnerability တွေကို 2020 နှင့် 2021 ကာလများအတွင်း Security Researchers နှင့် Auditors တွေက တွေ့ရှိခဲ့ပြီးတော့ 2026 ခုနှစ်မှာ ဖြစ်ပွားခဲ့တဲ့ $5.4 Million Theft နဲ့ ဆက်စပ်မှုရှိကြောင်း သက်သေ မတွေ့ရသေးပါဘူး။ ဒါကြောင့် ဒီ Section မှာ ဖော်ပြမယ့် အကြောင်းအရာတွေဟာ Historical Audit Findings ဖြစ်ပြီး 2026 Incident မှာ အသုံးပြုခဲ့တဲ့ Attack Technique မဟုတ်ပါ။

Denom Mapping ဆိုတာ ဘာလဲ?

Cosmos Ecosystem မှာ Token တစ်ခုချင်းစီကို သတ်မှတ်ဖို့ "Denom" (Denomination) လို့ ခေါ်တဲ့ Identifier တစ်ခု ရှိပါတယ်။ Ethereum မှ ERC20 Token တစ်ခု Bridge ကို ဖြတ်ပြီး Cosmos Network ထဲကို ဝင်လာတဲ့အခါ Cosmos ဘက်က "ဒီ ERC20 Token ဟာ ဘယ် Denom နဲ့ သက်ဆိုင်သလဲ?" ဆိုတာကို ဆုံးဖြတ်ရပါတယ်။ ဒီ ERC20 Contract Address နဲ့ Cosmos Denom ကြား ချိတ်ဆက်ထားတဲ့ Mapping ကို Denom Mapping လို့ ခေါ်ပါတယ်။

Bridge က ERC20 Contract Address နဲ့ Cosmos Denom ကို မှန်မှန်ကန်ကန် Mapping လုပ်နိုင်ရပါမယ်။ ဒီ Mapping မှားသွားရင် မတူညီတဲ့ Token နှစ်ခုကို Cosmos ဘက်မှာ တူညီတဲ့ Asset အဖြစ် မြင်သွားနိုင်ပါတယ်။ အဲ့ဒီအချိန်မှာ Asset Integrity ပြဿနာတွေ ဖြစ်လာနိုင်ပါတယ်။

Original Duplicate Token Vulnerability

Denom Mapping နှင့် ပတ်သက်တဲ့ အစောဆုံး Security Issue တွေထဲက တစ်ခုကို 2020 ခုနှစ် April 13 ရက်နေ့မှာ GitHub Issue #181 အဖြစ် Report လုပ်ခဲ့ပါတယ်။ Report ထဲမှာ ဖော်ပြထားတာက Attackers တွေက Bridge ကို ဖြတ်ပြီး ဝင်လာခဲ့ဖူးတဲ့ ERC20 Token တစ်ခုရဲ့ Copy Contract ကို Ethereum ပေါ်မှာ ပြန်ဖန်တီးနိုင်ပါတယ်။ ထို့နောက် အဆိုပါ Copy Token တွေကို Bridge ကနေတစ်ဆင့် Cosmos Network ကို ပို့ဆောင်နိုင်ပါတယ်။ Cosmos ဘက်ကနေ ကြည့်ရင် Copy Token တွေနဲ့ Original Token တွေကို မခွဲခြားနိုင်ပါဘူး။

ဒီ Vulnerability ရဲ့ အဓိက ပြဿနာက Cosmos ဘက်မှာ Token Identity ကို မှန်မှန်ကန်ကန် ခွဲခြားသတ်မှတ်နိုင်ခြင်း မရှိတဲ့ အခြေအနေ ဖြစ်ပေါ်လာနိုင်ခြင်း ဖြစ်ပါတယ်။ တစ်နည်းအားဖြင့် Original USDC နဲ့ Fake USDC ကို Cosmos ဘက်က တူညီတဲ့ Asset အဖြစ် မှတ်ယူသွားနိုင်တဲ့ Risk ရှိခဲ့တာ ဖြစ်ပါတယ်။

Source: https://github.com/cosmos/gravity-bridge/issues/181

Denom Mapping နဲ့ ပတ်သက်တဲ့ Logic တွေကို Gravity Bridge ရဲ့ `module/x/gravity/keeper/ethereum_event_handler.go` ဖိုင်အတွင်းမှာ တွေ့နိုင်ပါတယ်။

```
// Is this token originally from Cosmos, or from Ethereum?
isCosmosOriginated, denom := a.keeper.ERC20ToDenomLookup(ctx, event.TokenContract)

if !isCosmosOriginated {
    // Ethereum native token: mint a Cosmos voucher AFTER a safety check.
    // DetectMaliciousSupply(...) guards against an impossible supply,
    // ... then MintCoins(...)
}

// Overflow guard so a token cannot claim an absurd supply.
if newSupply.BitLen() > 256 {
    return sdkerrors.Wrap(types.ErrSupplyOverflow, denom)
}
```

Ethereum ဘက်မှာရှိတဲ့ Commit `92d0e12` `solidity/contracts/Gravity.sol` (line 611) မှာတော့ ဘယ်သူမဆို Token တစ်ခုကို Register လုပ်နိုင်ပါတယ်။

```
// Gravity.sol, lines 546 to 565 (commit 92d0e12)
function deployERC20(
    string memory _cosmosDenom,
    string memory _name,
    string memory _symbol,
    uint8 _decimals
) public {                      // public, with NO access control: anyone can call it
    // deploys a fresh ERC20 and emits an event that the oracles must parse
}
```

`public` ဖြစ်ပြီး Access Control မရှိတာက ဒီပြဿနာရဲ့ အစဖြစ်လာနိုင်တဲ့ နေရာပါ။ Oracle တွေက Token ရဲ့ name၊ symbol နဲ့ denom တွေကို ဘယ်လိုဖတ်သလဲဆိုတာနဲ့ တွဲကြည့်လိုက်တဲ့အခါ Code4rena Audit (2021 August 26 မှ September 8 အထိ) မှာ အောက်က ပြဿနာတွေကို တွေ့ခဲ့ပါတယ်။

ID	Severity	ရှင်းလင်းချက်
H-02	High	Token name ထဲမှာ UTF-8 မဟုတ်တဲ့ ထူးဆန်းတဲ့ Character တွေ ထည့်လိုက်ရင် Oracle ရဲ့ Log Reader က Error တက်သွားနိုင်ပါတယ်။ အဲ့ဒီနောက် Fail Loop ထဲမှာ ပိတ်မိသွားပြီး Attestation Process တွေ ရပ်သွားတယ်။ ရလဒ်အနေနဲ့ Bridge က Transfer တွေကို ဆက်ပြီး Confirm မလုပ်နိုင်တော့ပါဘူး။
H-03	High	`deployERC20` ကနေ Denom၊ Token Name ဒါမှမဟုတ် Symbol ကို အရမ်းရှည်တဲ့ Data တွေ ထည့်ပို့လိုက်ရင် Oracle ရဲ့ Log Size Limit (ခန့်မှန်း 10MB) ကို ကျော်သွားနိုင်ပါတယ်။ အဲ့ဒါကြောင့် Oracle တွေ Sync မဖြစ်တော့ဘဲ Network နဲ့ Out of Sync ဖြစ်သွားနိုင်ပါတယ်။

ID	Severity	ရှင်းလင်းချက်
M-04	Medium	`sendToCosmos` Function က User ပြောတဲ့ Transfer Amount ကိုပဲ ယုံကြည်ထားပါတယ်။ Transfer လုပ်တိုင်း Fee တစ်စိတ်တစ်ပိုင်း Burn လုပ်တဲ့ Token မျိုးဆိုရင် Contract က တကယ်လက်ခံရရှိတဲ့ Amount က ပို့လိုက်တဲ့ Amount ထက် လျော့နည်းနေပါလိမ့်မယ်။ ဒါပေမယ့် Cosmos ဘက်ကတော့ Original Amount အတိုင်း Credit ပေးလိုက်တဲ့အတွက် တကယ်ရောက်လာတဲ့ ပမာဏထက် ပိုပြီး Credit ရသွားနိုင်ပါတယ်။

Audit အပြည့်အစုံကို https://code4rena.com/reports/2021-08-gravitybridge မှာ သွားဖတ်လို့ရပါတယ်ဗျ။

Root Cause

2026 Attack: ပြဿနာက Code မှာ မဟုတ်ဘူး၊ Key မှာ ဖြစ်တာ

ဒီ Bridge ရဲ့ Security Model မှာ အခြေခံ Rule တစ်ခု ရှိပါတယ်။ Ethereum Contract ကနေ Withdrawal တစ်ခုကို Approve လုပ်နိုင်တာ တကယ့် Bridge Signers တွေပဲ ဖြစ်ရမယ်။ 2026 May 30 မှာ အဲ့ဒီ Rule ပျက်သွားခဲ့တယ်။ ဒါပေမယ့် Contract က Logic မှားသွားလို့ မဟုတ်ဘူး။ Contract က သူ့အလုပ်သူ မှန်မှန်ကန်ကန် လုပ်နေခဲ့တာပါ။ တကယ့် ပြဿနာကတော့ Bridge ကို ထိန်းချုပ်တဲ့ Secret Signing Key က Attacker လက်ထဲ ရောက်သွားတာ ဖြစ်ပါတယ်။ Key ကို ရသွားပြီဆိုရင် Attacker လုပ်တဲ့ Signature အားလုံးက Contract အတွက် တရားဝင် Signature တွေလိုပဲ မြင်ရပါတယ်။ Contract က ဘာမှ မမှားခဲ့ဘူး။ ဒါပေမယ့် orders ပေးနေတဲ့ လူက မှားနေခဲ့တာပါ။

Incident ကို စတင် ဖော်ထုတ်ခဲ့တဲ့ PeckShield၊ Cyvers နဲ့ Specter ဆိုတဲ့ Researcher တို့ အားလုံးကလည်း တူညီတဲ့ အချက်ကို ညွှန်ပြကြပါတယ်။ Bridge Contract Key ဒါမှမဟုတ် Signing Path တစ်ခုခု Compromise ဖြစ်ခဲ့ပြီး ပြဿနာက Smart Contract Code ထဲမှာ မဟုတ်ဘူး ဆိုတာပါပဲ။

ဒီနေရာမှာ Symptom နဲ့ Cause ကို သီးသန့် ခွဲကြည့်ရင် ပိုရှင်းပါတယ်။

- **Symptom** Funds တွေက ပုံမှန် Signed Withdrawal တွေလိုမျိုး Contract ထဲက ထွက်သွားခဲ့တယ်။
- **Cause** Attacker က အဲ့ဒီ Signature တွေ ထုတ်ပေးနိုင်တဲ့ Key ကို ထိန်းချုပ်နိုင်ခဲ့တယ်။

ဒီလို Attack မျိုးကို Solidity Audit တစ်ခုတည်းနဲ့ ဖမ်းမိဖို့ မဖြစ်နိုင်ပါဘူး။ ဘာကြောင့်လဲဆိုတော့ Vulnerability က Solidity Code ထဲမှာ မရှိလို့ပါ။ ပြဿနာက Chain အပြင်ဘက်မှာ ရှိတဲ့

- Key ကို ဘယ်လို သိမ်းထားသလဲ
- ဘယ်သူတွေနဲ့ မျှဝေထားသလဲ
- ဘယ်လို ကာကွယ်ထားသလဲ

ဆိုတဲ့ Operational Security ပိုင်းမှာ ရှိနေတာ ဖြစ်ပါတယ်။

အခုအချိန်ထိ Key ဘယ်လို ပေါက်ကြားသွားတယ်ဆိုတာကို မသိရသေးပါဘူး။ ဥပမာ

- Phishing Attack ဖြစ်ခဲ့တာလား
- Server Hack ခံရတာလား
- Secret တစ်ခု Leak ဖြစ်သွားတာလား
- Insider တစ်ယောက် ပါဝင်ခဲ့တာလား

ဆိုတာတွေကို Team ဘက်က ထုတ်ပြန်ထားခြင်း မရှိသေးပါဘူး။

Denom Mapping Bugs: အတုလုပ်လို့ရတဲ့ Identity ပြဿနာ

Audit တွေမှာ တွေ့ခဲ့တဲ့ Denom Mapping Vulnerability တွေကတော့ လုံးဝ မတူတဲ့ ပြဿနာတစ်မျိုး ဖြစ်ပါတယ်။ အဲ့ဒီ Vulnerability တွေရဲ့ အခြေခံ Rule က Cosmos ဘက်မှာ Token တစ်ခုရဲ့ Identity ကို Attacker အတုလုပ်လို့ မရတဲ့ အရာနဲ့ ချိတ်ထားရမယ် ဆိုတာပါ။ အဲ့ဒီ အရာက Token ရဲ့ တကယ့် Contract Address ဖြစ်ပါတယ်။ ဒါ့အပြင် Attacker က ပေးလိုက်တဲ့ String တွေ၊ Amount တွေကိုလည်း အမြဲတမ်း မယုံသင့်ပါဘူး။

အစောပိုင်း Design တွေမှာတော့

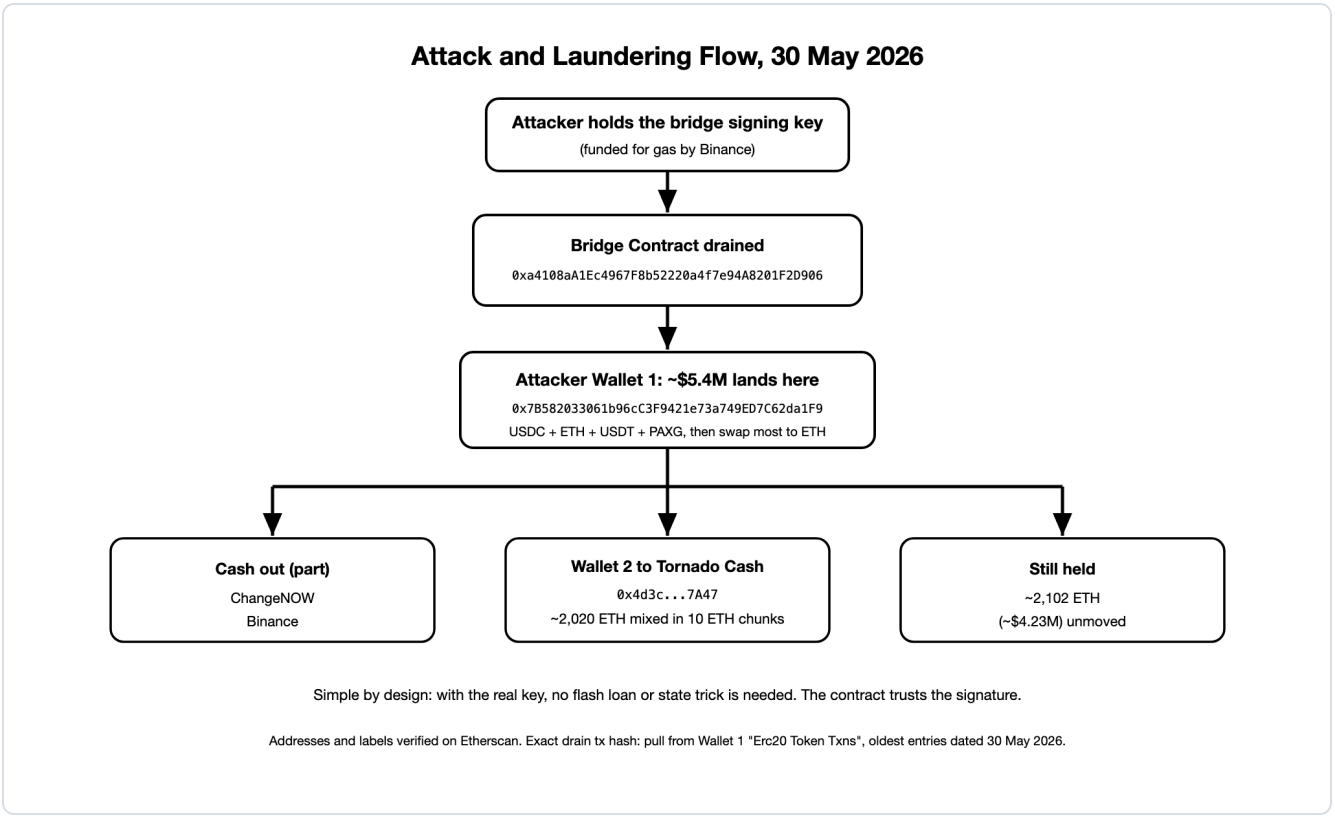
- Copy လုပ်လို့ရတဲ့ Metadata (name, symbol စတာတွေ)
- Sender က သူ့ဘာသာ ပြောတဲ့ Amount

တွေကို အလွန်အကျွံ ယုံကြည်တာတွေ။ ဒါက Web3 မှာ မကြာခဏ တွေ့ရတဲ့ Classic Mistake တစ်ခုပါ။ Attacker ထည့်ပေးတဲ့ Input ကို Trusted Data လို့ သတ်မှတ်မိတာ ဖြစ်ပါတယ်။

Exploit Walkthrough

ဒီ Attack က Operational Attack ဖြစ်တဲ့အတွက် Flash Loan Attack မဟုတ်သလို State ကို လှည့်စားတဲ့ နည်းလမ်းမျိုးလည်း မပါဝင်ပါဘူး။ Signing Key ကို ရသွားတဲ့ အချိန်မှာတော့ Attack တစ်ခုလုံးက ရိုးရိုးရှင်းရှင်း ဖြစ်သွားပါတယ်။

Wallet နဲ့ Contract Address တွေကို Etherscan မှာ Verify လုပ်ပြီးနောက် Attack က ဘယ်လို ဖြစ်ခဲ့သလဲ၊ ခိုးယူထားတဲ့ Funds တွေကို ဘယ်လို လမ်းကြောင်းဖျောက်ခဲ့သလဲ ဆိုတာကို အဆင့်လိုက် ကြည့်ကြရအောင်။



Attack နဲ့ Laundering Flow တစ်ခုလုံး Wallet နဲ့ Contract Address တွေကို Etherscan မှာ Verify လုပ်ထားပါတယ်။

1. **Setup.** အစမှာ Attacker က Binance ကနေ Wallet 1 ထဲကို Gas Fee အတွက် ETH ပို့ပါတယ်။ Signing Key ကို ရထားပြီးသားဖြစ်တာကြောင့် Flash Loan ယူဖို့၊ Position ဖွင့်ဖို့ ဒါမှမဟုတ် ရှုပ်ထွေးတဲ့ Strategy တွေ သုံးဖို့ မလိုအပ်တော့ပါဘူး။
2. **Trigger.** နောက်တစ်ဆင့်မှာ Attacker က Bridge Contract ဆီကို Signed Withdrawal Orders တွေ ပို့လိုက်ပါတယ်။ Signature တွေက မှန်ကန်နေတဲ့အတွက် Contract ကလည်း Locked Token တွေကို Release လုပ်ပေးလိုက်ပါတယ်။

3. **Drain.** အဲ့ဒီနောက် USDC ခန့်မှန်း $4.3 Million, ETH နှင့် WETH 274 ETH, USDT ခန့်မှန်း $434,000, PAXG 14.164 တို့ကို Wallet 1 ထဲသို့ လွှဲပြောင်းနိုင်ခဲ့ပါတယ်။ စုစုပေါင်း တန်ဖိုးကတော့ $5.4 Million လောက် ရှိပါတယ်။

4. **Convert.** ခိုးယူထားတဲ့ Stablecoin အများစုကို ETH အဖြစ် ပြောင်းလဲခဲ့ပါတယ်။ ဘာကြောင့်လဲဆိုတော့ ETH က လွှဲပြောင်းရ ပိုလွယ်ပြီး Funds တွေရဲ့ လမ်းကြောင်းကို ဖျောက်ဖို့လည်း ပိုအဆင်ပြေပါတယ်။

5. **Launder.** ခိုးယူထားတဲ့ Funds အချို့ကို ChangeNOW နဲ့ Binance တို့ကနေတစ်ဆင့် Cash Out လုပ်ခဲ့ပါတယ်။ ETH တွေကိုတော့ Wallet 2 ဆီ ပို့ခဲ့ပြီး Wallet 2 ကနေ Tornado Cash ထဲကို ETH 2,020 ခန့် ကို 10 ETH စီ ခွဲပြီး ပို့ခဲ့ပါတယ်။ Wallet 2 မှာ တွေ့ရတဲ့ Transaction ID အချို့ကတော့ `0x989127...`, `0xcaddca...`, `0xf2d528...`, `0x934dd3...` တို့ ဖြစ်ပါတယ်။

6. **ကျန်တဲ့ Funds တွေ.** Report ထွက်လာတဲ့ အချိန်အထိ Attacker ဆီမှာ ETH 2,102 ခန့် ကျန်ရှိနေသေးပြီး တန်ဖိုးအားဖြင့် $4.23 Million လောက် ရှိပါတယ်။ အဲ့ဒီ Funds တွေဟာ On-Chain ပေါ်မှာပဲ ရှိနေသေးပြီး ရွှေ့ပြောင်းထားခြင်း မရှိသေးပါဘူး။

အောက်မှာ ဖော်ပြထားတဲ့ Address အားလုံးကို Etherscan မှာ ကိုယ်တိုင် သွားစစ်ကြည့်နိုင်ပါတယ်။

Role	Address	Etherscan Label
Bridge contract that was robbed	`0xa4108aA1Ec4967F8b52220a4f7e94A8201F2D906`	"Gravity Bridge: Bridge" (verified)
Attacker Wallet 1, the main one	`0x7B582033061b96cC3F9421e73a749ED7C62da1F9`	"Gravity Bridge Exploiter"
Attacker Wallet 2, the helper	`0x4d3ca32e687e871a58b78AcAc73bE59AC37C7A47`	"Gravity Bridge Exploiter 2"
Mixer that was used	`0xd90e2f925da726b50c4ed8d0fb90ad053324f31b`	"Tornado.Cash: Router"

Theft Transaction ID အတိအကျနဲ့ ပတ်သက်ပြီး

ဒီနေရာမှာ တစ်ခု ရိုးရိုးသားသား ပြောရမယ်ဆိုရင် လက်ရှိ Public Source တွေထဲမှာ Drain ဖြစ်ခဲ့တဲ့ Transaction ID အတိအကျကို ထုတ်ပြန်ထားတာ မတွေ့ရပါဘူး။ တစ်ဖက်မှာလည်း Etherscan က Date Filter Page တွေကို Automated Tools တွေ အသုံးပြုလို့ မရအောင် ပိတ်ထားပါတယ်။ ပုံမှန် Transaction View မှာလည်း နောက်ဆုံး Transaction 25 ခုကိုပဲ ပြသပေးတာကြောင့် Hack ဖြစ်ခဲ့တာက အဲ့ဒီထက် စောနေ

တဲ့အတွက် တိုက်ရိုက် မမြင်ရတော့ပါဘူး။ ဒါပေမယ့် ကိုယ်တိုင် စစ်ကြည့်ချင်ရင် နှစ်မိနစ်လောက်နဲ့ ရှာလို့ရပါတယ်။

1. Wallet 1 ကို ဖွင့်ပါ
   https://etherscan.io/address/0x7B582033061b96cC3F9421e73a749ED7C62da1F9
2. "Erc20 Token Txns" Tab ကို နှိပ်ပါ။
3. 2026 May 30 ရက်နေ့က Transaction တွေအထိ Scroll ဆင်းသွားပါ။ From Field မှာ Bridge Contract Address ဖြစ်တဲ့ `0xa4108aA1...` ကနေ လာတဲ့ USDC Transfer အကြီးကြီးကို တွေ့ရပါလိမ့်မယ်။ အဲ့ဒီ Transaction ကို နှိပ်လိုက်ရင် Full Transaction ID နဲ့ Block Number ကို ကြည့်နိုင်ပါတယ်။

Proof of Concept

2026 Incident အတွက်တော့ Proof of Concept (PoC) ရေးပြဖို့ မရှိပါဘူး။ ဘာကြောင့်လဲဆိုတော့ ဒီ Exploit က Smart Contract Bug ကို Exploit လုပ်တာ မဟုတ်ဘဲ Secret Key ကို ရသွားတာ ဖြစ်လို့ပါ။ Key Compromise ကို Test Fork တစ်ခုပေါ်မှာ Reproduce လုပ်ပြလို့ မရပါဘူး။ PoC က အများဆုံး "တရားဝင် Signer ရဲ့ Key နဲ့ Withdraw ခေါ်လိုက်" လို့ပဲ ဖြစ်သွားမှာပါ။ ဒါက ဘာမှ သက်သေမပြနိုင်ပါဘူး။ ဒီလို Live Key Compromise Incident မျိုးမှာတော့ အပေါ်မှာ ပြထားတဲ့ On-Chain Transaction Data တွေကပဲ အကောင်းဆုံး သက်သေ ဖြစ်ပါတယ်။

Denom Mapping အပိုင်းနဲ့ ပတ်သက်ရင်တော့ M-04 Fee Token Issue ကို နားလည်လွယ်အောင် ပြထားတဲ့ ဥပမာတစ်ခု ရှိပါတယ်။

```
// ILLUSTRATIVE ONLY. The idea behind the M-04 fee token mismatch.
// A deflationary token takes a cut on every transfer.
contract FeeToken is ERC20 {
    function transferFrom(address f, address t, uint256 amt) public override returns (bool)
{
        uint256 fee = amt / 100;        // 1 percent burned on transfer
        super.transferFrom(f, t, amt - fee);
        return true;
    }
}

// The old sendToCosmos trusted `amount`, not the real change in balance:
//   token.safeTransferFrom(msg.sender, address(this), amount);   // received is less than
amount
//   state.lastEventNonce++;                                      // but it credits the full
amount
// So the Cosmos side mints more than the bridge actually holds.
```

ဒါပေမယ့် အဲ့ဒါက teaching sketch example သာ ဖြစ်ပြီး Tested Exploit Run မဟုတ်ပါဘူး။ ပြီးတော့ Vulnerability ကိုလည်း Patch လုပ်ပြီးသား ဖြစ်ပါတယ်။ ဒါကြောင့် Exploit Code ဒါမှမဟုတ် Attack Script တစ်ခုအဖြစ် မယူဆသင့်ပါဘူး။

Impact

2026 Incident မှာ ဘယ်လောက် ဆုံးရှုံးခဲ့လဲ

စုစုပေါင်း ဆုံးရှုံးမှုက $5.4 Million လောက် ရှိပါတယ်။

- USDC ခန့်မှန်း $4.3M
- ETH နဲ့ WETH 274 ETH (တန်ဖိုးအားဖြင့် $553K ခန့်)
- USDT ခန့်မှန်း $434K
- PAXG 14.164 (တန်ဖိုးအားဖြင့် $64K ခန့်)

ဘယ်သူတွေ ထိခိုက်သွားလဲ

Attack ဖြစ်တဲ့ အချိန်မှာ Bridge ထဲမှာ Asset Lock လုပ်ထားတဲ့ User တွေ အားလုံး ထိခိုက်သွားပါတယ်။ ဒါ့အပြင် Cosmos နဲ့ Ethereum ကြား Liquidity အပေါ်မှီခိုနေတဲ့ Ecosystem တစ်ခုလုံးလည်း သက်ရောက်မှု ရှိခဲ့ပါတယ်။

Attacker က ဘာလိုအပ်ခဲ့လဲ

လိုအပ်တာ တစ်ခုပဲ ရှိပါတယ်။ Signing Key ကို ရဖို့ပါပဲ။ ဒါက Attack အတွက် တစ်ခုတည်းသော Precondition ဖြစ်သလို ရဖို့ အခက်ခဲဆုံး အရာလည်း ဖြစ်ပါတယ်။ ဒါကြောင့် Key Compromise Attack တွေက ဖြစ်ခဲပေမယ့် ဖြစ်သွားရင်တော့ ဆိုးဆိုးရွားရွား ထိခိုက်စေတတ်ပါတယ်။

အခုချိန်မှာ ထပ် Exploit လုပ်လို့ ရသေးလား

Bridge ကို ရပ်ထားပြီး ဖြစ်ပါတယ်။ Key ကို Abuse လုပ်ပြီးသား ဖြစ်တဲ့အတွက် Fix က Smart Contract ဘက်မှာ မဟုတ်ဘဲ Operational ဘက်မှာ ဖြစ်ပါတယ်။ ဥပမာ

- Signing Keys အသစ် ပြောင်းထုတ်တာ
- Signer Infrastructure ကို ပြန်တည်ဆောက်တာ
- Key ဘယ်လို ပေါက်ကြားသွားလဲဆိုတာ စုံစမ်းဖော်ထုတ်တာ

စတဲ့ လုပ်ဆောင်ချက်တွေ လိုအပ်ပါတယ်။

Denom Mapping Vulnerabilities တွေကို လွန်ခဲ့တဲ့ နှစ်ပေါင်းများစွာကတည်းက Patch လုပ်ပြီးသား ဖြစ်ပါတယ်။ 2026 Incident မှာလည်း အသုံးပြုခဲ့တာ မဟုတ်ပါဘူး။ အခုရေးတဲ့ အထဲမှာ ထည့်သွင်းဖော်ပြထားတာကတော့ Historical Security Lessons အနေနဲ့ လေ့လာနိုင်ဖို့အတွက်သက်သက်သာ ဖြစ်ပါတယ်။

The Fix

2026 Incident မှာ ဖြစ်ခဲ့တဲ့ Key Compromise အတွက် Fix က Code ပြင်ရုံနဲ့ မပြီးပါဘူး။ ဒါက Operational Security Problem ဖြစ်တာကြောင့် Solution ကလည်း Operational Side မှာပဲ ရှိပါတယ်။ ပထမဆုံး Signer Key အားလုံးကို rotate လုပ်ရပါမယ်။ ပြီးရင် Key တွေကို Hardware Security Module (HSM) ထဲမှာ သိမ်းတာမျိုး ဒါမှမဟုတ် ပိုပြီး လုံခြုံတဲ့ Multisig Setup ကို ပြောင်းသုံးတာမျိုး လုပ်သင့်ပါတယ်။ အဲ့ဒီအပြင် Key ဘယ်လို ပေါက်ကြားသွားခဲ့လဲဆိုတာကိုလည်း သေချာ Audit လုပ်ပြီး စုံစမ်းရပါမယ်။ အရေးကြီးဆုံးကတော့ ခိုးယူခံထားရတဲ့ Key ကို Code Patch တစ်ခုနဲ့ ပြန်မပြင်နိုင်ပါဘူး။ Key ကို အသစ်ပြောင်းရမယ်၊ ပြီးတော့ နောက်နောင်မှာ ထပ်မဖြစ်အောင် သိမ်းဆည်းတဲ့ နည်းလမ်းကို ပိုမို လုံခြုံအောင် လုပ်ရပါမယ်။

Denom Mapping Vulnerability တွေဘက်မှာတော့ Fix က Code Level မှာ ရှိပါတယ်။ Duplicate Token Problem ကို ဖြေရှင်းဖို့ Token တစ်ခုရဲ့ Identity ကို name ဒါမှမဟုတ် symbol ပေါ်မူတည်မနေဘဲ တကယ့်

ERC20 Contract Address နဲ့ တိုက်ရိုက် ချိတ်ဆက်ထားပါတယ်။ ဒါ့အပြင် Mapping ကို Direction နှစ်ဖက် စလုံးမှာ သိမ်းဆည်းထားတဲ့အတွက် Copycat Contract တစ်ခု ဖန်တီးလိုက်ရင်လည်း Original Token နဲ့ Denom တူသွားမှာ မဟုတ်တော့ပါဘူး။ အဲ့ဒီ Copycat Token က Original Token အဖြစ် pretend လုပ်နိုင်တော့မှာ မဟုတ်ပါဘူး။

```
// cosmos_originated.go: store the mapping in BOTH directions, keyed by contract address
+ store.Set(types.MakeDenomToERC20Key(denom), tokenContract.Bytes())
+ store.Set(types.MakeERC20ToDenomKey(tokenContract), []byte(denom))
  // A copied ERC20 has a DIFFERENT address, so it gets a DIFFERENT denom. No impersonation.
```

M-04 အတွက်တော့ Document ထဲမှာ ဖော်ပြထားတဲ့ Fix က Transfer မတိုင်ခင်နဲ့ Transfer ပြီးနောက် Balance ကို တိုင်းတာပြီး တကယ်ရောက်လာတဲ့ Amount ကိုပဲ Credit ပေးဖို့ ဖြစ်ပါတယ်။

```
- token.safeTransferFrom(msg.sender, address(this), _amount);
- // credited _amount blindly
+ uint256 beforeBal = token.balanceOf(address(this));
+ token.safeTransferFrom(msg.sender, address(this), _amount);
+ uint256 received = token.balanceOf(address(this)) - beforeBal;
+ // credit `received`, never the requested `_amount`
```

ဒီ Fix တွေက အပေါ်ယံမြင်ရတဲ့ Symptom ကိုပဲ ဖြေရှင်းတာ မဟုတ်ပါဘူး။ အမှန်တကယ် Root Cause ဖြစ်တဲ့

- အတုလုပ်လို့ရတဲ့ Identity (Fakeable Identity)
- ယုံကြည်မသင့်တဲ့ Input ကို ယုံကြည်ထားခြင်း (Trusted Input)

ဆိုတဲ့ ပြဿနာတွေကို ဖြေရှင်းပေးတာ ဖြစ်ပါတယ်။ ဒီ Code Diff တွေက Document တွေမှာ ဖော်ပြထားတဲ့ Fix Approach ကို အခြေခံပြီး ပြန်လည်တည်ဆောက်ထားတာ ဖြစ်ပါတယ်။ Finding တစ်ခုချင်းစီအတွက် အတိအကျ Merge လုပ်ခဲ့တဲ့ Commit Hash ကို ကျွန်တော် မရှာနိုင်ခဲ့ပါဘူး။ ဒါကြောင့် ဒီ Diff တွေကို Exact Fix Commit လို့ မယူဆဘဲ Mitigation ကိုပြန်လည်ဖော်ပြထားတဲ့ Reconstruction အနေနဲ့ပဲ ယူဆသင့်ပါတယ်။

Takeaways

1. **Bridge Hack အများစုမှာ ဆုံးရှုံးမှု ဖြစ်ရတာက Logic Bug ကြောင့်ထက် Key Compromise ကြောင့် ပိုများပါတယ်။** Smart Contract က ဘယ်လောက်ပဲ Perfect ဖြစ်နေပါစေ၊ အဲ့ဒီ Contract ကို ထိန်းချုပ်တဲ့ Keys တွေ လုံခြုံမှုသာ တကယ်လုံခြုံမှာ ဖြစ်ပါတယ်။ Bridge တစ်ခုကို Threat Model လုပ်တဲ့အခါ Solidity Code ကို Audit လုပ်တာတင် မဟုတ်ဘဲ Hardware Security Modules (HSM), Multisig Setup, Signer Rotation, Key Custody လို Key Management အပိုင်းတွေကိုလည်း အလားတူ အရေးကြီးတယ်လို့ သတ်မှတ်ပြီး အချိန်ပေး စဉ်းစားသင့်ပါတယ်။
2. **Contract တစ်ခု Verify ဖြစ်နေတယ်ဆိုတာ Protocol တစ်ခု လုံခြုံတယ်လို့ မဆိုလိုပါဘူး။** Key တွေကို ဘယ်သူကိုင်ထားလဲ၊ ဘယ်လောက် လုံခြုံလဲဆိုတာလည်း မပြောပေးနိုင်ပါဘူး။
3. **Attacker ထိန်းချုပ်နိုင်တဲ့ Input ကို ဘယ်တော့မှ မယုံပါနဲ့။** Denom Mapping Vulnerability တွေ အားလုံးရဲ့ Root Cause က Name ကို ယုံတာ၊ Symbol ကို ယုံတာ၊ User ပြောတဲ့ Amount ကို ယုံတာတွေ ဖြစ်ပါတယ်။ Token Identity ကို Attacker အတုလုပ်လို့ မရတဲ့ အရာတွေ (ဥပမာ Contract Address) နဲ့ ချိတ်ထားသင့်ပါတယ်။ Amount တွေကိုလည်း User ပြောတာကို မယုံဘဲ တကယ် ရောက်လာတဲ့ Balance ကို တိုင်းတာပြီး ဆုံးဖြတ်သင့်ပါတယ်။
4. **Bug ကို မှန်မှန်ကန်ကန် နာမည်ပေးသင့်ပါတယ်။** ဒီ Incident ကို Denom Mapping Exploit လို့ ခေါ်လိုက်ရင် Defender တွေက မှားတဲ့နေရာကို သွား Fix မိနိုင်ပါတယ်။ Symptom နဲ့ Cause ကို သီးခြား ခွဲမြင်ဖို့ လိုပါတယ်။ Symptom က Signed Withdrawals တွေ ဖြစ်ခဲ့တာ၊ Cause က Signing Key ခိုးယူခံခဲ့ရတာ ဖြစ်ပါတယ်။
5. **Code ကိုပဲ မကြည့်ပါနဲ့၊ ပိုက်ဆံ ဘယ်သွားလဲဆိုတာကိုလည်း လိုက်ကြည့်ပါ။** Real Incident တစ်ခုမှာ On-Chain Data က အမှန်တရားကို ပြောပေးတဲ့ အရေးကြီးဆုံး သက်သေ ဖြစ်ပါတယ်။ Address Labels တွေ၊ Binance ကနေ Gas Fee ဝင်လာတဲ့ Funding Path တွေ၊ Tornado Cash ထဲကို 10 ETH စီ ခွဲပို့တဲ့ Mixer Flow တွေက Code Review တစ်ခုတည်းနဲ့ မမြင်ရနိုင်တဲ့ အချက်အလက်တွေကို ဖော်ထုတ်ပေးနိုင်ပါတယ်။

Disclosure Timeline

Date	Event
13 April 2020	The denom mapping duplicate token issue, #181, is opened. Historical and separate.

Date	Event
26 Aug to 8 Sep 2021	The Code4rena audit surfaces the denom, name, and symbol findings, H-02, H-03, and M-04. Fixed later.
30 May 2026	The bridge is drained of about $5.4 million. Flagged the same day by PeckShield and Cyvers. Specter pins it on a signing key compromise. The bridge is halted.
30 May to early June 2026	The attacker swaps to ETH, cashes out through ChangeNOW and Binance, mixes about 2,020 ETH through Tornado Cash, and still holds about 2,102 ETH.
11 June 2026	No official Gravity Bridge postmortem has been published yet.

Responsible Disclosure အနေနဲ့ ပြောရရင် ဒီ Post ထဲမှာ ပါတဲ့ အချက်အလက်အားလုံးက Public ဖြစ်ပြီးသား အချက်အလက်တွေပါ။ ပြောထားတဲ့ Code Vulnerability တွေကိုလည်း လွန်ခဲ့တဲ့ နှစ်များကတည်းက Fix လုပ်ပြီးသား ဖြစ်ပါတယ်။ ဒါကြောင့် ဒီ Post ထဲမှာ ပါတဲ့ အကြောင်းအရာတွေထဲက ဘယ်တစ်ခုမှ လက်ရှိ အချိန်မှာ အသုံးချလို့ရတဲ့ Attack Method ဒါမှမဟုတ် Live Weapon မဟုတ်ပါဘူး။

Sources

- Etherscan, bridge contract: https://etherscan.io/address/0xa4108aa1ec4967f8b52220a4f7e94a8201f2d906
- Etherscan, Exploiter 1: https://etherscan.io/address/0x7B582033061b96cC3F9421e73a749ED7C62da1F9
- Etherscan, Exploiter 2: https://etherscan.io/address/0x4d3ca32e687e871a58b78AcAc73bE59AC37C7A47
- PeckShield via Bitcoin.com: https://news.bitcoin.com/gravity-bridge-exploit-5-4-million-binance-changenow-2026/
- Cryptopolitan: https://www.cryptopolitan.com/hacker-drain-gravity-bridge/
- BeInCrypto on the key compromise: https://beincrypto.com/gravity-bridge-hack-key-compromise-5m/

- CryptoTimes: https://www.cryptotimes.io/2026/05/30/gravity-bridge-hit-in-5-4m-exploit-amid-suspected-key-compromise/
- GitHub issue #181: https://github.com/cosmos/gravity-bridge/issues/181
- Gravity.sol at commit 92d0e12: https://github.com/althea-net/cosmos-gravity-bridge/blob/92d0e12cea813305e6472851beeb80bd2eaf858d/solidity/contracts/Gravity.sol
- Code4rena audit: https://code4rena.com/reports/2021-08-gravitybridge