An Insecure Fork of MasterChef

An Insecure Fork

ในการพัฒนาซอฟต์แวร์ คำว่า fork เป็นได้ทั้งคำกริยาซึ่งหมายถึงการแยกการพัฒนาโครงการซอฟต์แวร์ด้วยการคัดลอกซอร์สโค้ดทั้งหมด แล้วนำมาพัฒนาโดยเป็นอิสระจากโครงการต้นฉบับ คำว่า fork ยังเป็นคำนามซึ่งหมายถึงโครงการซอฟต์แวร์ที่ถูกคัดลอกจากการ fork ได้อีกด้วย

หากใครเคยมีประสบการณ์กับแพลตฟอร์ม DeFi มาประมาณหนึ่ง เราอาจจะพอคุ้นกับประโยคว่า "*แพลตฟอร์มนี้ fork มาจากโครงการ A*" และถึงแม้บางท่านอาจจะไม่เคยได้ยินประโยคนี้มาก่อนเลย ผมเชื่อว่ามีไม่น้อยที่เราจะรู้สึกว่าบางแพลตฟอร์มนั้นมีลักษณะหรือวิธีการใช้งานที่แทบจะเหมือนกัน แตกต่างกันแค่ในรายละเอียด

การ fork แพลตฟอร์ม DeFi นั้นเป็นเรื่องทั่วไปที่เกิดขึ้นอยู่ตลอด แพลตฟอร์มหลายแพลตฟอร์มซึ่งเกิดขึ้นมาใหม่ถูกนำมาพัฒนาและให้บริการต่อผ่านการ fork จากแพลตฟอร์มที่มีอยู่ก่อนแล้ว การ fork เกิดขึ้นกับแพลตฟอร์ม DeFi ได้ทั้งแพลตฟอร์ม หรืออาจทำกับบางส่วนของแพลตฟอร์ม เช่น ส่วนของหน้าตา (frontend) หรือส่วนของ contract ก็ได้

เพราะการ fork จะทำให้เราได้สำเนาที่เหมือนกับต้นฉบับ หากต้นฉบับมีปัญหาในจุดใด สำเนาของเราย่อมมีปัญหาในจุดนั้นตาม และด้วยความอิสระระหว่างสำเนากับต้นฉบับ หากต้นฉบับรับรู้ถึงปัญหาและมีการแก้ไข การแก้ไขที่เกิดขึ้นกับต้นฉบับก็ย่อมไม่เกิดขึ้นกับสำเนาและทำให้สำเนายังคงมีปัญหาต่อไปหากผู้ที่ fork ไปไม่ทำการแก้ไขเอง

หากเราเป็นนักพัฒนา บทความนี้อาจช่วยเป็นจุดเริ่มต้นให้เรากลับไปทำความเข้าใจปัญหาที่มีอยู่เดิมในแพลตฟอร์มต้นฉบับ ซึ่งจะนำไปสู่การจัดการปัญหาที่เราอาจไม่เคยรู้มาก่อนได้ และอาจช่วยเราจากการเข้าไปมีส่วนร่วมกับปัญหาผ่านการพัฒนาต่อโดยไม่ได้ตั้งใจได้ด้วย

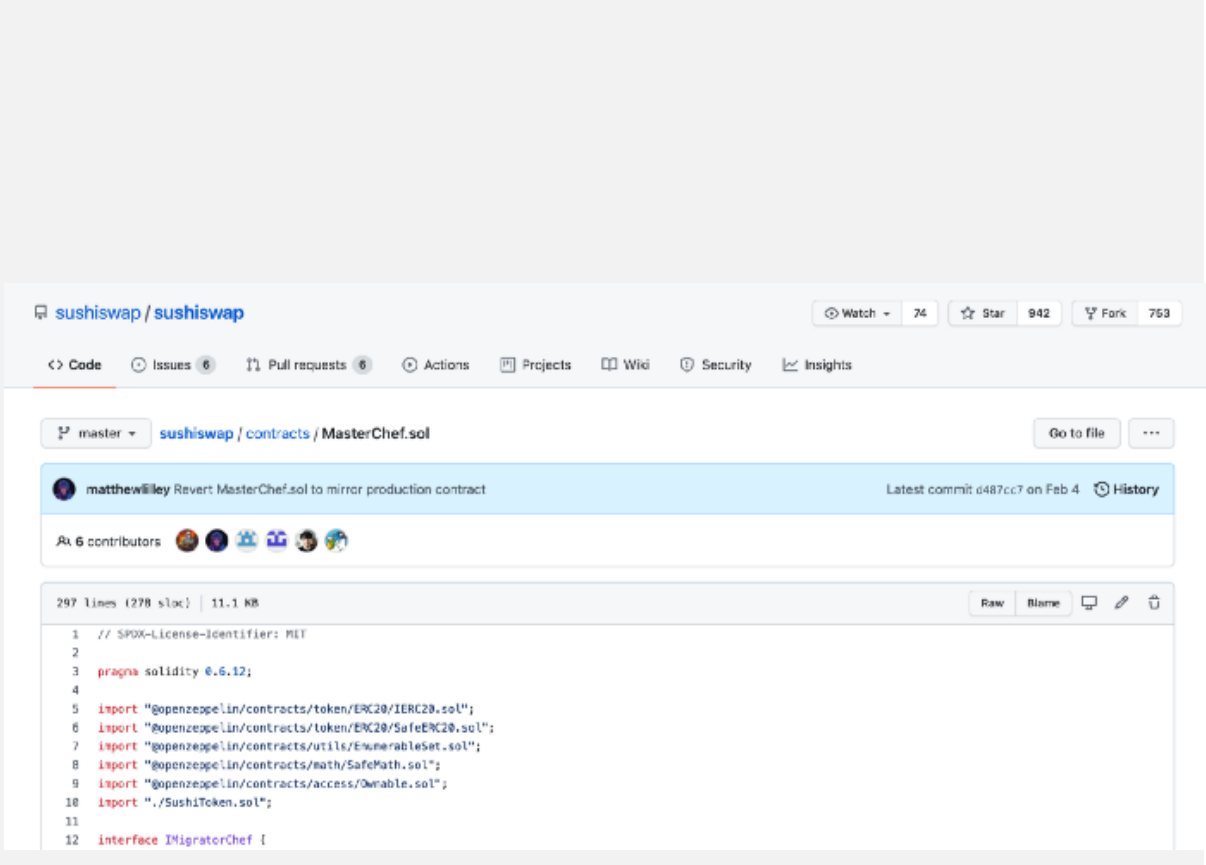
และ**หากเราเป็นผู้ใช้งาน** การเข้าใจความเสี่ยงอย่างถูกต้องนั้นเป็นสิ่งสำคัญอย่างยิ่งต่อการลงทุน บทความนี้อาจช่วยให้เรารู้ถึงความเสี่ยงของแพลตฟอร์มและอาจช่วยให้เราสังเกตเห็นหากผู้พัฒนาแพลตฟอร์ม DeFi ตั้งใจที่จะใช้ปัญหาที่ถูกสืบทอดมาเหล่านี้ในการหาประโยชน์เข้าตัวเองได้อีกด้วย

The MasterChef

MasterChef คือชื่อของ contract หนึ่งที่มีที่มาจากแพลตฟอร์ม SushiSwap (ถูกเปลี่ยนชื่อใหม่เป็น Sushi ในเวลาต่อมา)

หลังจากช่วงกลางปี 2020 ในยุคที่แพลตฟอร์ม DeFi อย่าง Uniswap ยึดครองตลาดภายใต้แนวคิดของการมี liquidity pool เพื่อกำหนดสภาพคล่องในการแลกเปลี่ยน พร้อมกับให้ค่าธรรมเนียมที่เกิดจากการแลกเปลี่ยนกลับไปยังผู้ที่เข้ามาให้สภาพคล่องหรือ liquidity provider เพื่อเป็นแรงจูงใจในการให้สภาพคล่องต่อ แพลตฟอร์ม SushiSwap ได้เกิดขึ้นมาภายใต้แนวคิดของการให้และใช้เหรียญที่มีชื่อว่า $SUSHI เป็นหนึ่งในกลไกเพื่อตอบแทนและจูงใจ liquidity provider รายละเอียดเพิ่มเติมในส่วนนี้สามารถอ่านได้จากบล็อกของ Sushi

เพื่อให้เกิด $SUSHI ที่แสนอร่อยหนึ่งคำ เราจึงจำเป็นต้องมีสุดยอดเชฟซูชิฝีมือดี หน้าที่นั้นจึงถูกมอบหมายให้กับ MasterChef



หน้าที่แต่เดิมของ MasterChef คือการรับฝาก liquidity provider (LP) token รวมไปถึงคำนวณการออก $SUSHI ให้กับ liquidity provider ซึ่งกลายมาเป็นส่วนหลักของแนวคิดการทำ yield farming

เมื่อความนิยมของแพลตฟอร์ม DeFi เริ่มเพิ่มมากขึ้นและตามมาด้วยการเกิดขึ้นใหม่ของแพลตฟอร์ม DeFi อีกเป็นจำนวนมาก MasterChef ก็ถูก fork และ

ถูกพัฒนาเกิดเป็นเวอร์ชันใหม่ ก่อนที่เวอร์ชันใหม่เหล่านั้นจะถูก fork และทำซ้ำต่อไปอีกเรื่อย ๆ

ด้วยหน้าที่ที่สำคัญและอาจจะซับซ้อนในบางกรณีของ MasterChef ที่สืบทอดมาพร้อมกับความเสี่ยงและความเป็นไปได้ที่ความซับซ้อนนั้นจะเกิดเป็นปัญหา เราจึงขอหยิบ MasterChef มาเป็นพระเอกที่มีหลายบทบาทในบทความนี้ พร้อมกับพูดถึงปัญหาและความเสี่ยงที่แฝงและถูกสืบทอดมาจากการ fork MasterChef ครับ

SushiSwap's Migrator

ตัวอย่างที่เราจะมาพูดถึงกันเป็นตัวอย่างแรกนั้นอยู่ไม่ไกลจากเรื่องราวในส่วนก่อนหน้าเพราะมันยังอยู่ใน MasterChef ของ SushiSwap โค้ดในส่วนนี้มีส่วนสำคัญในปฏิบัติการโอนถ่ายสภาพคล่องระหว่าง Uniswap และ SushiSwap ในอดีตภายใต้ชื่อเรียกเทคนิคว่า Vampire Attack โค้ดในส่วนนี้ประกอบไปด้วยฟังก์ชันทั้งหมด 2 ฟังก์ชัน คือ `setMigrator()` และ `migrate()` ซึ่งสามารถดูได้จากไฟล์ MasterChef.sol

```
// Set the migrator contract. Can only be called by the owner.
function setMigrator(IMigratorChef _migrator) public onlyOwner {
    migrator = _migrator;
}

// Migrate lp token to another lp contract. Can be called by anyone. We trust that migrator contract is good.
function migrate(uint256 _pid) public {
    require(address(migrator) != address(0), "migrate: no migrator");
    PoolInfo storage pool = poolInfo[_pid];
    IERC20 lpToken = pool.lpToken;
    uint256 bal = lpToken.balanceOf(address(this));
    lpToken.safeApprove(address(migrator), bal);
    IERC20 newLpToken = migrator.migrate(lpToken);
    require(bal == newLpToken.balanceOf(address(this)), "migrate: bad");
    pool.lpToken = newLpToken;
}
```

ฟังก์ชัน `migrate()` ซึ่งรับค่า `_pid` หรือลำดับของ pool ใน MasterChef และถูกกำหนดให้สามารถถูกเรียกใช้งานได้โดยผู้ใช้งานทุกคน เมื่อถูกเรียกใช้จะทำให้เกิดเหตุการณ์ดังต่อไปนี้

1. ทำการตรวจสอบว่าค่า address ที่อยู่ในตัวแปร `migrator` ว่าไม่ได้มีค่า `address` เท่ากับ 0 เงื่อนไขในบรรทัดนี้มีความหมายเท่ากับว่าการเรียกใช้ฟังก์ชัน `migrate()` จะต้องมีการกำหนดค่าของตัวแปร `migrator` ผ่านฟังก์ชัน `setMigrator()` ก่อนสำหรับกรณีนี้
2. ค่า `_pid` ถูกนำไปใช้อ้าง pool ที่มีอยู่เพื่อดึง address ของ LP token

3. มูลค่าของ LP token ถูกดึงมาเก็บไว้ในตัวแปร ค่านี้จะถูกใช้เพื่อเปรียบเทียบมูลค่าที่เปลี่ยนแปลงเพื่อยืนยันการโอนถ่ายว่าสมบูรณ์หรือไม่
4. ฟังก์ชัน `safeApprove()` ถูกเรียกใช้เพื่อกำหนดมูลค่าที่อนุญาตให้ address ของ `migrator` สามารถทำธุรกรรมได้
5. ฟังก์ชัน `migrate()` ซึ่งอยู่ภายใต้ Migrator contract ถูกเรียกเพื่อโอนถ่าย LP token ทั้งหมดไปยัง address ที่ถูกระบุไว้ในตัวแปร `migrator` พร้อมกับตรวจสอบค่าที่เปลี่ยนแปลงเพื่อยืนยันการโอนถ่ายและอัปเดตค่า `address`

เราสามารถเห็นได้ในกระบวนการทำงานของฟังก์ชัน `migrate()` ซึ่งปรากฏใน MasterChef ว่ามันมีพลังในการโอน LP token ไปยังบุคคลอื่นได้ ความสำคัญของฟังก์ชัน `migrate()` เชื่อมโยงไปยังฟังก์ชัน `setMigrator()` ในทันที เนื่องจากฟังก์ชัน `setMigrator()` ทำหน้าที่ในการกำหนด address ที่จะรับ LP token อ้างอิงจากโค้ดของฟังก์ชัน `setMigrator()` ผู้ที่จะสามารถเรียกใช้ฟังก์ชันนี้เพื่อกำหนด address ของ `migrator` ได้คือ owner

มาถึงจุดนึเราพอจะเห็นความเป็นไปได้แล้วใช่ไหมครับว่า 2 ฟังก์ชันนึสร้างสิ่งที่เรียกว่า rug pull ได้

การมีอยู่ของฟังก์ชันที่เกี่ยวข้องกับการโอนย้ายสภาพคล่องถูกพิสูจน์แล้วว่ามีความเสี่ยงจากการถูกใช้เพื่อทำ rug pull หลายต่อหลายครั้ง ความพยายามในการจัดการกับเหตุการณ์ในลักษณะนี้เพื่อไม่ให้เกิดซ้ำมีทั้งการใช้ฟังก์ชัน Timelock เพื่อเพิ่มโอกาสในการลดกระทบ หรือการป้องกันโดยการนำโค้ดในส่วนนี้ออกไปเลย

และนี่คือส่วนแรกที่เรานำมาพูดถึงสำหรับปัญหาและความเสี่ยงที่ถูกสืบทอดและส่งถ่ายมาใน MasterChef contract ครับ

PancakeSwap's Syrup Pools

PancakeSwap เป็นอีกหนึ่งแพลตฟอร์ม DeFi ในธีมอาหารที่ไม่ว่าใครก็ต้องเคยได้ยินชื่อ หากเราย้อนดูประวัติการพัฒนาไฟล์ MasterChef.sol ซึ่งเก็บ MasterChef contract เอาไว้ เราจะสามารถเห็นได้จากประวัติการพัฒนาว่าไฟล์ MasterChef.sol ของ PancakeSwap ก็ถูก fork มาจากไฟล์ MasterChef.sol ของ SushiSwap ในขณะนั้น

อย่างไรก็ตามจุดที่เราจะมาพูดถึงกันในส่วนนี้นั้นอยู่ในจุดที่ทีม PancakeSwap มีการพัฒนาขึ้นมาจากเดิมแต่ยังคงอยู่ใน MasterChef contract การพัฒนาเพิ่มขึ้นมาใหม่นี้ถูกรู้จักกันในชื่อของ Syrup Pools ซึ่งมาพร้อมกับ $SYRUP

เราอาจเรียก Syrup Pools ได้ว่าเป็นหนึ่งในความพยายามของทีม PancakeSwap ในการสร้างระบบนิเวศน์ใหม่ให้กับแพลตฟอร์มซึ่งอาจนำไปสู่แนวทางในการใช้ประโยชน์และเพิ่มมูลค่าให้กับเหรียญหรือกระบวนการที่เกี่ยวข้อง

การฝาก $CAKE ใน MasterChef contract จะมีการสร้าง $SYRUP ขึ้นมาให้กับผู้ใช้ โดยเหรียญ $SYRUP นี้สามารถนำมาฝากไว้ใน Syrup Pools เพื่อรับผลตอบแทนเป็นเหรียญในสกุลอื่น ๆ ได้ และหากผู้ใช้ต้องการถอน $CAKE ออกมาจาก MasterChef contract ผู้ใช้จำเป็นจะต้องคืน $SYRUP เป็นจำนวนเท่ากันกับที่ถูกสร้างขึ้นมาในตอนแรก

```
// Stake CAKE tokens to MasterChef
function enterStaking(uint256 _amount) public {
    PoolInfo storage pool = poolInfo[0];
    UserInfo storage user = userInfo[0][msg.sender];
    updatePool(0);
    if (user.amount > 0) {
        uint256 pending = user.amount.mul(pool.accCakePerShare).div(1e12).sub(user.rewardDebt);
        if(pending > 0) {
            safeCakeTransfer(msg.sender, pending);
        }
    }
    if(_amount > 0) {
        pool.lpToken.safeTransferFrom(address(msg.sender), address(this), _amount);
        user.amount = user.amount.add(_amount);
    }
    user.rewardDebt = user.amount.mul(pool.accCakePerShare).div(1e12);

    syrup.mint(msg.sender, _amount);
    emit Deposit(msg.sender, 0, _amount);
}

// Withdraw CAKE tokens from STAKING.
function leaveStaking(uint256 _amount) public {
    PoolInfo storage pool = poolInfo[0];
    UserInfo storage user = userInfo[0][msg.sender];
    require(user.amount >= _amount, "withdraw: not good");
    updatePool(0);
    uint256 pending = user.amount.mul(pool.accCakePerShare).div(1e12).sub(user.rewardDebt);
    if(pending > 0) {
        safeCakeTransfer(msg.sender, pending);
    }
    if(_amount > 0) {
        user.amount = user.amount.sub(_amount);
        pool.lpToken.safeTransfer(address(msg.sender), _amount);
    }
    user.rewardDebt = user.amount.mul(pool.accCakePerShare).div(1e12);

    syrup.burn(msg.sender, _amount);
    emit Withdraw(msg.sender, 0, _amount);
}
```

หลังจากที่ Syrup Pools และ $SYRUP ถูกเปิดตัวได้ไม่นาน ในวันที่ 3 พฤศจิกายน 2020 ทาง PancakeSwap ก็ออกประกาศหยุดการสนับสนุน $SYRUP โดยทันทีหลังจากมีการตรวจพบปัญหาใน contract ซึ่งทำให้เกิดการออก $SYRUP ได้มากเกินความจริง ปัญหานี้เกิดจากแนวคิดในการใช้

ฟังก์ชันซึ่งพบเห็นได้ทั่วไปแถมเป็นฟังก์ชันที่มีประโยชน์อีกเสียด้วยมาโจมตี ฟังก์ชันนั้นคือ `emergencyWithdraw()`

```
// Withdraw without caring about rewards. EMERGENCY ONLY.
function emergencyWithdraw(uint256 _pid) public {
    PoolInfo storage pool = poolInfo[_pid];
    UserInfo storage user = userInfo[_pid][msg.sender];
    pool.lpToken.safeTransfer(address(msg.sender), user.amount);
    emit EmergencyWithdraw(msg.sender, _pid, user.amount);
    user.amount = 0;
    user.rewardDebt = 0;
}
```

ฟังก์ชัน `emergencyWithdraw()` เป็นฟังก์ชันที่มีหน้าที่ตามชื่อคือทำให้เราสามารถทำการ “ถอนแบบฉุกเฉิน” ได้ในทันที ลองนึกถึงสถานการณ์ที่แพลตฟอร์ม DeFi ซึ่งเราใช้งานไม่สามารถเข้าถึงได้เพราะถูกโจมตีหรือเกิดปัญหา หากเราต้องการถอนทรัพย์สินที่ไปวางไว้ออก ฟังก์ชัน `emergencyWithdraw()` ก็สามารถถูกใช้เพื่อภารกิจนี้ได้ครับ

หากเรายังจำกันได้ว่ากลไกของการได้ $SYRUP มานั้นเกิดขึ้นได้จากการนำ $CAKE ไปวางทำให้ฟังก์ชัน `syrup.mint()` ถูกเรียกใช้งาน และต้องมีไปคืนถ้า

จะเอา $CAKE ออก ส่วนที่เราเอาไปคืนก็จะถูกทำลายทิ้งด้วยฟังก์ชัน `syrup.burn()` ถ้าหากเราไม่ออกจาก Syrup Pools ด้วยการนำ $SYRUP ไปคืน แต่ออกด้วยฟังก์ชัน `emergencyWithdraw()` ที่ไม่ได้เขียนให้มีการทำลายเหรียญ $SYRUP ทิ้งล่ะครับ อะไรจะเกิดขึ้น?

ผลลัพธ์ที่จะเกิดขึ้นคือเราสามารถสร้าง $SYRUP ได้มากเกินความเป็นจริง เพราะฟังก์ชัน `emergencyWithdraw()` นั้นไม่มีเงื่อนไขของการทำลาย $SYRUP ทิ้งเมื่อนำ $CAKE ออกมาอยู่

การโจมตีนี้สามารถถูกระบุได้ด้วยการหาการเรียกใช้ฟังก์ชัน `enterStaking()` ซึ่งมีการเรียก `syrup.mint()` อยู่ข้างใน และการเรียกใช้ฟังก์ชัน `emergencyWithdraw()` ปริมาณการเรียกใช้ทั้งสองฟังก์ชันก็เป็นจุดสังเกตที่ดีต่อการตรวจสอบการโจมตีด้วย

Timestamp	Block	Caller	Smart Contract	Method	Gas Cost	Transaction
2020-11-07 11:03:24	[redacted]	0x0	0x73feaa1ee314f8c655e3...	emergencyWithdraw	0.00	0x
2020-11-07 11:02:00		0x0	0x73feaa1ee314f8c655e3...	emergencyWithdraw	0.00	0x
2020-11-07 11:01:48		0x0	0x73feaa1ee314f8c655e3...	enterStaking	0.00	0x
2020-11-07 11:01:33		0x0	0x73feaa1ee314f8c655e3...	emergencyWithdraw	0.00	0x
2020-11-07 11:01:21		0x0	0x73feaa1ee314f8c655e3...	enterStaking	0.00	0x
2020-11-07 11:01:09		0x0	0x73feaa1ee314f8c655e3...	emergencyWithdraw	0.00	0x
2020-11-07 11:00:57		0x0	0x73feaa1ee314f8c655e3...	enterStaking	0.00	0x
2020-11-07 11:00:45		0x0	0x73feaa1ee314f8c655e3...	emergencyWithdraw	0.00	0x
2020-11-07 11:00:33		0x0	0x73feaa1ee314f8c655e3...	enterStaking	0.00	0x
2020-11-07 11:00:21		0x0	0x73feaa1ee314f8c655e3...	emergencyWithdraw	0.00	0x
2020-11-07 11:00:09		0x0	0x73feaa1ee314f8c655e3...	enterStaking	0.00	0x
2020-11-07 10:59:57		0x0	0x73feaa1ee314f8c655e3...	emergencyWithdraw	0.00	0x
2020-11-07 10:59:42		0x0	0x73feaa1ee314f8c655e3...	enterStaking	0.00	0x
2020-11-07 10:59:30		0x0	0x73feaa1ee314f8c655e3...	emergencyWithdraw	0.00	0x
2020-11-07 10:59:18		0x0	0x73feaa1ee314f8c655e3...	enterStaking	0.00	0x
2020-11-07 10:59:06		0x0	0x73feaa1ee314f8c655e3...	emergencyWithdraw	0.00	0x
2020-11-07 10:58:54		0x0	0x73feaa1ee314f8c655e3...	enterStaking	0.00	0x
2020-11-07 10:58:42		0x0	0x73feaa1ee314f8c655e3...	emergencyWithdraw	0.00	0x
2020-11-07 10:58:30		0x0	0x73feaa1ee314f8c655e3...	enterStaking	0.00	0x
2020-11-07 10:58:18		0x0	0x73feaa1ee314f8c655e3...	emergencyWithdraw	0.00	0x
2020-11-07 10:58:06		0x0	0x73feaa1ee314f8c655e3...	enterStaking	0.00	0x
2020-11-07 10:57:51		0x0	0x73feaa1ee314f8c655e3...	emergencyWithdraw	0.00	0x
2020-11-07 10:57:39		0x0	0x73feaa1ee314f8c655e3...	enterStaking	0.00	0x
2020-11-07 10:57:30		0x0	0x73feaa1ee314f8c655e3...	emergencyWithdraw	0.00	0x
2020-11-07 10:57:15		0x0	0x73feaa1ee314f8c655e3...	enterStaking	0.00	0x
2020-11-07 10:57:03		0x0	0x73feaa1ee314f8c655e3...	emergencyWithdraw	0.00	0x
2020-11-07 10:56:51		0x0	0x73feaa1ee314f8c655e3...	enterStaking	0.00	0x
2020-11-07 10:56:39		0x0	0x73feaa1ee314f8c655e3...	emergencyWithdraw	0.00	0x
2020-11-07 10:56:27		0x0	0x73feaa1ee314f8c655e3...	enterStaking	0.00	0x
2020-11-07 10:56:15		0x0	0x73feaa1ee314f8c655e3...	emergencyWithdraw	0.00	0x
2020-11-07 10:55:42		0x0	0x73feaa1ee314f8c655e3...	enterStaking	0.00	0x
2020-11-07 10:38:03		0x0	0x73feaa1ee314f8c655e3...	emergencyWithdraw	0.00	0x
2020-11-07 10:37:51		0x0	0x73feaa1ee314f8c655e3...	enterStaking	0.00	0x
2020-11-07 10:27:15		0x0	0x73feaa1ee314f8c655e3...	emergencyWithdraw	0.00	0x
2020-11-07 10:27:03		0x0	0x73feaa1ee314f8c655e3...	enterStaking	0.00	0x
2020-11-07 10:26:51		0x0	0x73feaa1ee314f8c655e3...	emergencyWithdraw	0.00	0x

ตัวอย่างของพฤติกรรมที่มีลักษณะสอดคล้องกับแนวทางในการโจมตีช่องโหว่

หลังจากตรวจพบการโจมตี ทีม PancakeSwap ได้มีการดำเนินการเพื่อจัดการกับสถานการณ์ที่เกิดขึ้นโดยทันที อย่างไรก็ตามการดำเนินการเพื่อจัดการปัญหานั้นไม่ปรากฏถึงการแก้ไข MasterChef contract และ SyrupBar contract ซึ่งเป็น contract ที่เกี่ยวข้อง ส่งผลให้โค้ดซึ่งทำให้เกิดเหตุการณ์นี้ยังคงอยู่ใน MasterChef contract เช่นเดิม ทีม PancakeSwap ได้อธิบายถึงการ

ตัดสินใจนี้เอาไว้โดยพูดถึงผลกระทบที่จะเกิดขึ้นกับผู้ใช้งาน PancakeSwap ทุกคน

การแก้ไขปัญหาของ PancakeSwap แม้จะดีต่อตัวแพลตฟอร์มเอง แต่การตัดสินใจที่จะไม่แก้ MasterChef contract นั้นทำให้ความเสี่ยงที่เกิดจาก MasterChef contract ยังคงอยู่ และทำให้การ fork กลายเป็นเรื่องที่เสี่ยงได้

การมีอยู่ของโค้ดซึ่งเกี่ยวข้องกับเหตุการณ์นี้คงไว้ซึ่งความเสี่ยงที่ควรถูกพูดถึงแม้จะไม่สร้างผลกระทบโดยตรงสำหรับแพลตฟอร์ม DeFi ที่ fork โค้ดไปพัฒนาต่อยกเว้นแต่ในกรณีที่แพลตฟอร์มจะมีการเข้าไปยุ่งเกี่ยวกับ $SYRUP เรามองว่าประเด็นในส่วนนี้เป็นประเด็นที่ควรถูกนำมาสร้างความตระหนักรู้เพื่อลดโอกาสที่จะทำให้เกิดข้อผิดพลาดในอนาคตครับ

The “Less Reward” Bug

สำหรับปัญหาสุดท้ายที่เราจะมาพูดถึงกันในครั้งนี้นั้นคือปัญหาที่พบได้จาก MasterChef contract ภายใต้กลไกของการยอมให้มีการวางเหรียญสกุลเดียวกับเหรียญซึ่งเป็นผลตอบแทนครับ

แพลตฟอร์ม DeFi โดยส่วนใหญ่มีการเปิด pool ให้ผู้ใช้งานนำเหรียญมาวางเพื่อรับผลตอบแทน โดยเจ้าของแพลตฟอร์มสามารถเลือกเปิด pool สำหรับเหรียญใดก็ได้ แต่มักเปิดให้ใช้ LP token ที่ได้รับจากการเป็น liquidity provider มาวางเพื่อสร้างอรรถประโยชน์ของเหรียญนั้น

การคำนวณผลตอบแทนของ pool จะต้องมีการนำตัวแปรอย่างปริมาณของเหรียญที่ถูกนำมาวางไว้มาคิด ทำให้มีโอกาสผิดพลาดในการคำนวณได้ หากปริมาณที่มีการนำมาใช้นั้นไม่ถูกต้อง

```
// Update reward variables of the given pool to be up-to-date.
function updatePool(uint256 _pid) public {
    PoolInfo storage pool = poolInfo[_pid];
    if (block.number <= pool.lastRewardBlock) {
        return;
    }
    uint256 lpSupply = pool.lpToken.balanceOf(address(this));
    if (lpSupply == 0) {
        pool.lastRewardBlock = block.number;
        return;
    }
    uint256 multiplier = getMultiplier(pool.lastRewardBlock, block.number);
    uint256 sushiReward =
        multiplier.mul(sushiPerBlock).mul(pool.allocPoint).div(
            totalAllocPoint
        );
    sushi.mint(devaddr, sushiReward.div(10));
    sushi.mint(address(this), sushiReward);
    pool.accSushiPerShare = pool.accSushiPerShare.add(
        sushiReward.mul(1e12).div(lpSupply)
    );
    pool.lastRewardBlock = block.number;
}
```

เราขอนำตัวอย่างของกระบวนการนี้มาจาก MasterChef ของ SushiSwap ในฟังก์ชัน `updatePool()` ปริมาณของเหรียญที่ผู้ใช้มาวางไว้จะถูกเก็บอยู่ในตัวแปร `lpSupply` ที่ได้จากการดูปริมาณเหรียญของ pool นั้น ๆ ใน MasterChef contract หากมีการเปิด pool ที่อนุญาตให้ใช้เหรียญสกุลเดียวกันกับผลตอบแทนมาวาง เช่นวางเหรียญ $A เพื่อรับผลตอบแทนเป็น $A ปริมาณของ `lpSupply` ก็จะเพิ่มขึ้นอย่างไม่ถูกต้องจากผลตอบแทนที่ถูกสร้างขึ้นมาเก็บไว้ใน MasterChef contract เรื่อย ๆ และส่งผลให้ผลตอบแทนจากการการนำค่า `lpSupply` ไปใช้เป็นตัวแปรมีค่าไม่ถูกต้องตามไปด้วย

ผลลัพธ์ของปัญหานี้กระทบโดยตรงกับผลประโยชน์ที่ผู้ใช้แพลตฟอร์ม DeFi พึงได้จากตัวแพลตฟอร์มเอง เพราะในทันทีที่การคำนวณในขั้นตอนนี้ผิดพลาด ผลประโยชน์ที่ผู้ใช้แพลตฟอร์ม DeFi ควรจะได้จะน้อยลงทันที และจะน้อยลงตามผลตอบแทนที่ยังไม่ได้ถูกนำเอาออกไปจาก MasterChef contract โดยผู้ใช้คนอื่น ๆ

สำหรับนักพัฒนา ปัญหานี้สามารถแก้ไขได้หลายวิธีการ ทั้งการตรวจสอบสกุลของเหรียญที่อนุญาตใน pool ก่อน หรือในกรณีที่การวางเหรียญในสกุลเดียวกับผลตอบแทนมีความจำเป็นสำหรับแพลตฟอร์มนั้น ๆ นักพัฒนาก็สามารถใช้วิธีแยก contract ซึ่งมีหน้าที่ในการเก็บผลตอบแทนเป็น contract ใหม่ เพื่อไม่ให้ปริมาณของเหรียญถูกคำนวณผิดได้

What Can We Do?

เพราะต้นกำเนิดของปัญหาและความเสี่ยงเหล่านี้มาจากการ fork และใช้งาน MasterChef contract ซึ่งมีปัญหา การตรวจสอบ MasterChef contract ก่อนตัดสินใจใช้งานแพลตฟอร์มถือเป็นทางเลือกหนึ่งที่ทำให้เราสามารถประเมินความเสี่ยงของแพลตฟอร์มได้ด้วยตัวเอง ในการตรวจสอบ Masterchef

contract เว็บไซต์ RugDoc ได้มีการรวบรวมวิธีการและแนวทางเอาไว้ ซึ่งสามารถปฏิบัติตามได้ตามลำดับดังนี้ครับ

1. วิธีในการระบุหา MasterChef contract และการนำมาระบุหาความแตกต่างกับ MasterChef contract ที่น่าจะเป็นต้นฉบับเพื่อดูว่ามีการแก้ไขอย่างไรบ้าง (ดูเพิ่มเติม)
2. ตัวอย่างของลายเซ็นต์หรือจุดที่เป็นเอกลักษณ์ของ MasterChef contract ที่อาจช่วยให้เราระบุได้ว่า MasterChef contract ที่เราตรวจสอบอยู่นั้นถูก fork มาจากแหล่งใด (ดูเพิ่มเติม)
3. ตัวอย่างของโค้ดที่มักถูกเพิ่มเข้าไปใน MasterChef รวมไปถึงโค้ดที่ใช้เพื่อโจมตี สำหรับการนำมาใช้เพื่อระบุหาการแก้ไข MasterChef contract ได้ (ดูเพิ่มเติม)

แม้ว่าเราจะผ่อนถ่ายภาระของตัวเองที่จะต้องตรวจสอบ MasterChef contract ไปให้กับผู้ให้บริการด้านการตรวจสอบ smart contract และเลือกที่จะเชื่อผลลัพธ์ของการถูกตรวจสอบได้ การเข้าใจอำนาจและความรับผิดชอบต่อตัวเองซึ่งเป็นคุณสมบัติของระบบแบบ Decentralized ที่ให้เรามานั้นถือว่าเป็นเรื่องที่

สำคัญยิ่งกว่า หวังว่าบทความนี้จะช่วยให้เราเข้าใจปัญหา ความเสี่ยงและสิ่งที่เราทำได้ในฐานะของพลเมืองของโลก DeFi อย่างเป็นประโยชน์ครับ