White Hat

© Mattie Schraeder 2018-2020, @AFrozenPeach, mattie@mattie.lgbt

2 - 6 players

90-120 minutes

Theme

White Hat is an action selection style game where each player leads a team of hackers trying to patch vulnerabilities on their computer. Players race to be the first team to patch all 4 vulnerabilities, ending the game and hopefully scoring the most points in the process.

These are not your Hollywood hackers. Real hacking is done heads down at a computer, writing code, penetration and quality assurance testing, building applications to aid research, and maintaining computer hardware. White Hat does its best to simulate the real world of hacking and computer development in a way that anyone can understand and play, without having to know the intricacies of computer hacking.

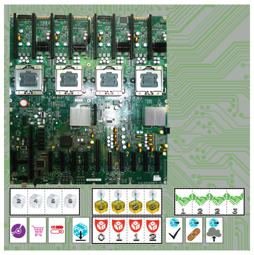
White hats are hackers that aren't malicious in their hacking. If they find vulnerabilities in someone's code, they report it, often doing so for monetary rewards, rather than exploiting it to do damage. **Black hats**, by comparison, exploit vulnerabilities maliciously and use them for personal gain. The term comes from old wild west movies where the heroes wear white hats, and the villains wear black.

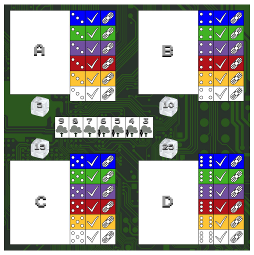
Capture the Flag is an event at many computer security and programming conferences and conventions. There are two types of Capture the Flag events. The most common is a Jeopardy style game where you get points for finding and patching vulnerabilities based on short clues. White Hat is based on a versus style event where each team of hackers is working against each other more directly. Each team starts with an identical computer running critical software that has a number of purposeful vulnerabilities. The goal is to find and patch those vulnerabilities.

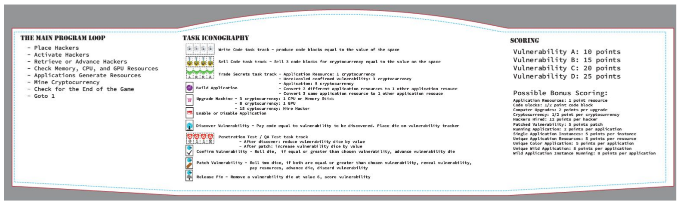
White Hat also has a bit of **cryptocurrency** as a mechanic. Cryptocurrencies are electronically generated currencies that don't exist in a physical form. Instead, an individual wallet's funds are stored in a digital ledger, and transactions pass between wallets update the ledger allowing users to track the movement of funds through the system. **Bitcoin**, the most popular cryptocurrency, is "mined" by randomly searching for a string of letters and numbers that matches the next coin to be found. White Hat uses a simplistic version of this to simulate mining cryptocurrency and using that to purchase upgrades for a player's computer as well as additional hackers.

Components

1. 6 player boards
2. 1 vulnerability board
3. 6 player screens
4. 6 cryptocurrency miner cards
5. 24 CPU tokens
6. 48 memory tokens
7. 12 GPU tokens
8. 12 end game scoring cards
9. 60 application cards
10. 100 vulnerability cards
11. 6 preparation cards
12. 1 score sheet
13. 52 dice
    - 8 Red
    - 8 Blue
    - 8 Purple
    - 8 Green
    - 8 Yellow
    - 8 White
    - 4 Brown
14. 60 meeples
    - 10 Red
    - 10 Blue
    - 10 Purple
    - 10 Green
    - 10 Yellow
    - 10 White
15. 550 cubes
    - 48 transparent orange cubes - CPU cores
    - 24 transparent blue cubes - GPU cores
    - 50 transparent clear cubes - Written code
    - 50 large transparent clear cubes - 5x Written code
    - 50 yellow cubes - Cryptocurrency
    - 50 large yellow cubes - 5x Cryptocurrency
    - 125 application cubes
        - 25 Red
        - 25 Blue
        - 25 Purple
        - 25 Green
        - 25 Pink







THE MAIN PROGRAM LOOP

- Place Hackers
- Activate Hackers
- Retrieve or Advance Hackers
- Check Memory, CPU, and GPU Resources
- Applications Generate Resources
- Mine Cryptocurrency
- Check for the End of the Game
- Goto 1

TASK ICONOGRAPHY

Write Code task track - produce code blocks equal to the value of the space

Sell Code task track - Sell 3 code blocks for cryptocurrency equal to the value on the space

Trade Secrets task track - Application Resource: 1 cryptocurrency
- Unrelvealed confirmed vulnerability: 3 cryptocurrency
- Application: 5 cryptocurrency
- Convert 2 different application resources to 1 other application resouce
- Convert 3 same application resource to 1 other application resouce

Build Application

Upgrade Machine - 3 cryptocurrency: 1 CPU or Memory Stick
- 8 cryptocurrency: 1 GPU
- 15 cyptocurrency: Hire Hacker

Enable or Disable Application

Discover Vulnerability - Pay code equal to vulnerability to be discovered. Place die on vulnerability tracker

Penetration Test / QA Test task track
- After discover: reduce vulnerability dice by value
- After patch: increase vulnerability dice by value

Confirm Vulnerability - Roll die, if equal or greater than chosen vulnerability, advance vulnerability die

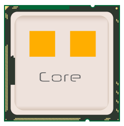
Patch Vulnerability - Roll two dice, if both are equal or greater than chosen vulnerability, reveal vulnerability, pay resources, advance die, discard vulnerability

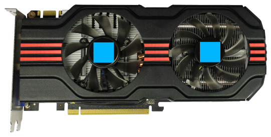
Release Fix - Remove a vulnerability die at value 6, score vulnerability

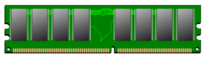
SCORING

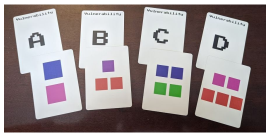
Vulnerability A: 10 points
Vulnerability B: 15 points
Vulnerability C: 20 points
Vulnerability D: 25 points

Possible Bonus Scoring:
Application Resources: 1 point resource
Code Blocks: 1/2 point code block
Computer Upgrades: 2 points per upgrade
Cryptocurrency: 1/2 point per cryptocurrency
Hackers Hired: 12 points per hacker
Patched Vulnerability: 5 points patch
Running Application: 2 points per application
Single Application Instances: 5 points per instance
Unique Application Resources: 5 points per resource
Unique Color Application: 5 points per application
Unique Wild Application: 8 points per application
Wild Application Instance Running: 8 points per application









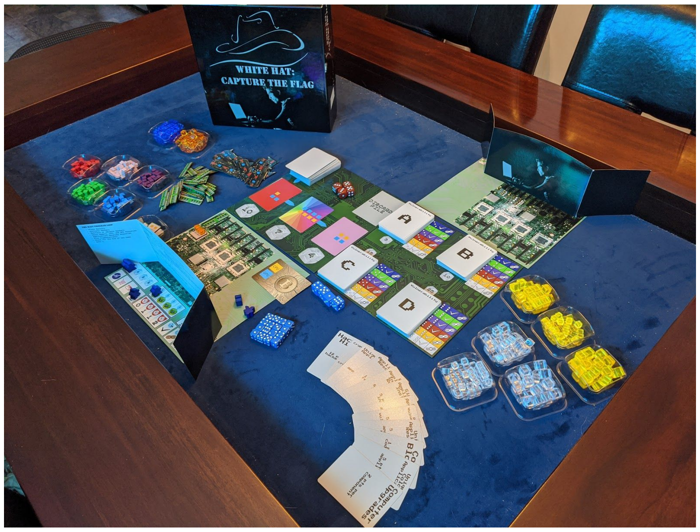
Goal

Ultimately each player's goal is to have the most points at the end of the game. Points are scored by releasing fixes for vulnerabilities as well as from secret goal cards that get chosen by each player.

Each vulnerability has a set of resources that are required to patch it, and once patched are worth a number of victory points equal to five times the number of resources required to patch it. Partially patching a vulnerability isn't enough, so you cannot get partial points.

Each secret goal card is worth up to 25 points depending on the goal. At the end of the game, each player in turn reveals their secret goal card, and then all players score that goal. Players who pay attention to what their opponents are focusing on will gain an advantage over others who focus purely on their own objectives. Secret goal cards encourage players to go above and beyond the bare minimum required to patch and release a fix for a vulnerability.

Setup



1) Give each player a bag with the following contents:
    a) 8 dice, 4 large and 4 small
    b) 6 hacker meeples
    c) A cryptocurrency miner application card
    d) 1 CPU
    e) 1 memory
    f) 2 CPU Cores
    g) A preparation card
2) Give each player a motherboard
    a) Place the CPU, CPU cores, and the memory on their designated spaces
3) Give each player a player screen
    a) For a player's first game it is recommended to not use the player screens, as it adds a bit of complexity, but in future games it makes much of the game more strategic.
4) Shuffle the end game scoring cards and deal two to each player
    a) Each player picks one of the two and returns the other back to the box face down
    b) The other is placed between themselves and the player to their right.
    c) Each player may look at the bonus card that was placed to their left.
        i) In the case of a 2 player game, each player only gets to look at their own end game scoring card
5) Each player starts with 4 hacker meeples. The rest are kept in their personal supply.
6) Place the vulnerability and application boards in the middle of the table
7) Shuffle each vulnerability deck and place them face down on the corresponding place on the vulnerability board
8) Shuffle the deck of application cards and place it face down on the blank spot on the application offer
9) Fill the application offer with the top 3 cards of the application deck

The Main Program Loop

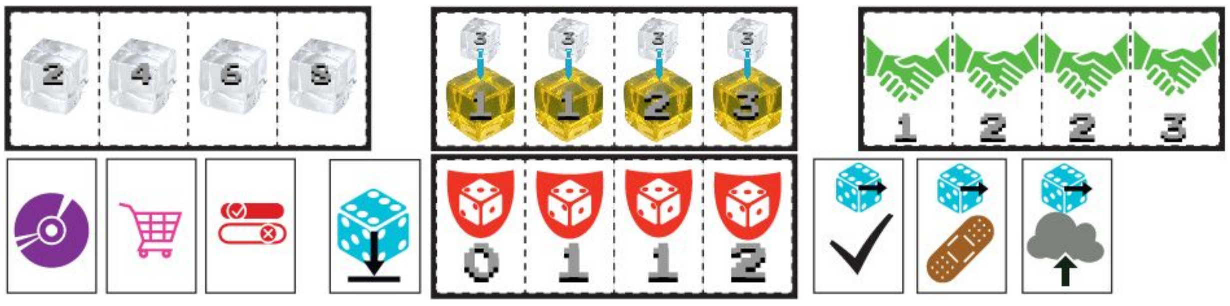
Computer programs are a series of one or more “loops”, the primary one of which is called the **Main Program Loop**. So too does White Hat. The Main Program Loop is where all the action happens, and repeats until a player has successfully patched all 4 vulnerabilities and released fixes for each of them. Each time the loop is fully performed it is called an **iteration**. Each step of the main program loop is a **function call**. In real world programming functions make it easier for code to be reused multiple times throughout the course of development.

The functions are as follows:

1. Place hackers
2. Activate hackers
3. Retrieve or advance hackers
4. Check memory/CPU/GPU allocations
5. Applications generate resources
6. Mine cryptocurrency
7. Check for the end of the game
8. Goto 1

1. Place Hackers

Each player starts with 4 hacker meeples on their team and two more can be hired throughout the course of the game. During the **Place Hackers** function players place all the hackers they have on their team on action spaces, which represent various **tasks** a hacker can perform, on their individual motherboards. Hackers can team up and be working on the same tasks, or can be split up amongst several different tasks.



Some tasks (**Write Code**, **Sell Code**, **Trade Secrets**, and **Pen/QA Test**), designated by a thick line surrounding four individual dashed action spaces, have **task tracks**. Whenever hackers are being placed, they start on the left most action space of that task. This remains true even if another hacker has already progressed further along the same track. During the **Retrieve or Advance Hackers** function they will progress along the track, therefore getting better at the task they were assigned.

At the start of the **Place Hackers** function, players should place their player screens so that they are blocking the other players’ view of the motherboard’s action spaces while allowing the components on the motherboard to still be visible. Players use the Red/Green preparation cards to signal when they’re ready.

2. Activate Hackers

Once all hackers have been placed, players execute the **Activate Hackers** function. Player screens are moved so that they reveal their hackers' task, but in a way that still hides the various resources players have collected. Tasks are done in order from top left to bottom right. Each task can be done independently among players with the exception of the **Build Applications** task. If only one player has hackers on a task, that player gets a bonus for taking a task alone. The bonus is different for each action

We'll go into each task in more detail later in the rules. Any resource gained in a previous task is available to be used in future tasks. For example, any code written during the write code task can immediately be sold during the following sell code task.

For the first game it is recommended that players perform each task one by one, with each player simultaneously performing that task, assuming they have one or more hackers assigned to it. As players become more experienced with the rules some tasks, such as writing and selling code can be easily combined.

3. Retrieve or Advance Hackers

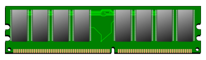
During the **Retrieve or Advance Hackers** function, each player simultaneously chooses to either advance a hacker along one of the four action tracks, or to retrieve them to be placed on a new action space in the next round. This is repeated for each hacker until all hackers have either been advanced or retrieved.

Hackers that are advanced along a task track will be better at that action in future rounds. It is advantageous to leave hackers on a task track and allow them to advance, as they will be more productive.

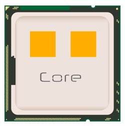
4. Check Memory, CPU, and GPU allocations

Computers have a limited amount of physical resources to run applications. Overloading a computer causes it to **crash**. If a player's computer crashes then the rest of the main program loop is forfeited. Their applications will not generate resources, their **Cryptocurrency Miner** will not produce more cryptocurrency, and they cannot cause the game to end.

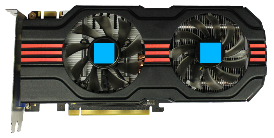
There are three types of physical resources: **Memory**, **CPUs**, and **GPUs**



Memory affects how many applications the computer can run simultaneously. Each memory allows one unique application to run on their computer, be it an application that generates resources for patching or the **Cryptocurrency Miner** that each player starts with.



CPUs allow players to run applications a number of times dependent on how many **CPU cores** it takes to run. Each **CPU** has two **cores**. Each **core** can be assigned to a single application, and some applications require multiple cores to run. **CPU Cores** are represented by transparent orange cubes.



GPUs are more advanced, and count as both **Memory** and a **CPU** for the purposes of allocating resources to applications. **GPU Cores** are represented by transparent blue cubes.

Some applications run better on **CPUs** while others run better on **GPUs**. The **Cryptocurrency Miner**, for example, has higher chances of producing cryptocurrency when running on a **GPU**.

5. Applications Generate Resources

Assuming a player's computer hasn't crashed, applications running on that computer generate resources that can be used for patching vulnerabilities. **Applications** are finished programs that perform tasks automatically as long as they are running on a computer.

In White Hat, applications produce **Application Resources** that represent work that the hackers are automating so that they don't have to do it themselves manually.

The color of the application card matches the color of **Application Resource** that it generates. Individual applications can be running on a computer multiple times, and each individual time is referred to as an **instance**. Each **instance** of that application running on a computer requires a number of **CPU** or **GPU cores** and generates one **resource cube**. The orange and blue squares on application cards represent how many **CPU cores** or **GPU cores** it takes to run a single **instance**.

6. Mine Cryptocurrency

Cryptocurrency is "mined" by solving complex mathematical problems. In White Hat mining cryptocurrency is simulated by rolling dice. Each player starts the game with one application, a **Cryptocurrency Miner**. This application works differently than the applications that players can build throughout the course of the game.

The **Cryptocurrency Miner** can run either on one **CPU core** or one **GPU core**. Just like normal applications it can be run multiple times by assigning more than one CPU core or GPU core to the Cryptocurrency Miner and players may even choose to run their Cryptocurrency Miner on both at the same time.

The four **brown** dice in the game represent the mathematical problem that the **Cryptocurrency Miner** must solve to generate one cryptocurrency. These dice are the **hash dice**. A hash is a series of random numbers and letters that represent the solution to the mathematical problem the **Cryptocurrency Miner** must solve.

The **Mine Cryptocurrency** function starts by rolling these four brown dice. Then each player rolls a number of their own colored dice decided by how many instances of their **Cryptocurrency Miner** are running and whether it's running on **CPU** or **GPU**. For each **CPU core** allocated to the Cryptocurrency

Miner players get one die of their color. For each **GPU core** allocated, players instead get two dice of their color.

Each die a player rolls that matches one of the **hash dice** receives that value in **cryptocurrency**, represented by transparent yellow cubes. Each **hash die** can only be matched once. Each player die can only match one **hash die**. If no dice match, the player may either reroll their dice once hoping for a better result, or instead gain one bitcoin per die rolled.

In the case where a player may roll more than four dice, they should reroll dice that do not match the hash dice until either they have rolled a number of times equal to their allotment or they match all four hash dice.

7. Check For the End of the Game

If any player has successfully released fixes for all four vulnerabilities, the game ends and final scores are calculated.

8. Goto 1

If the game has not ended, a new round of play continues with the **Place Hackers** function and through the rest of the **Main Program Loop** repeatedly until the end of the game has been triggered.

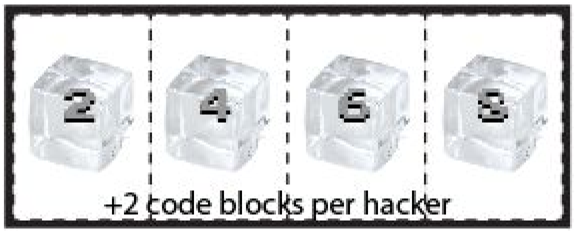
Early computer programs used the phrase “goto” as a way to jump to a specific section of code. In White Hat, once each function has been run in order, we “jump” back to the top of the code, and run each function again. This series of functions followed by a goto represent the **Main Program Loop**. When the end of the game has been reached, the Main Program Loop is “broken out of” and scores are calculated.

Tasks

Hackers get assigned to various tasks in the **Place Hackers** function. Each task represents a small bit of work that the hacker has completed, usually resulting in a resource that can be used in other tasks. With the exception of the **Build Applications** function, players may take these actions at their own pace. For the first few rounds of the game we recommend each player taking each task at the same time, but this isn’t necessary as players become familiar with the flow of the game.

Write Code

The **Write Code** task produces **code blocks** represented by clear transparent cubes. A hacker working on the write code task produces a number of **code blocks** equal to the value of the space they are on.



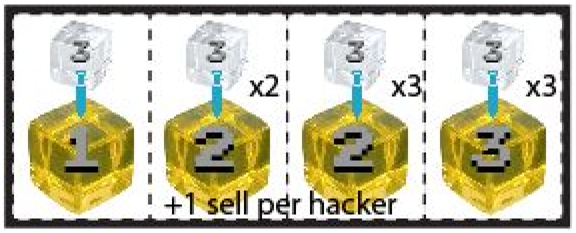
In White Hat, **code blocks** represent nonfunctioning, but still useful, code that the hackers have written and can be used for various nondescript purposes, as well as code that has been studied well enough that the hacker knows how it works without having to think about it.

Individual **code blocks** aren't worth much, but when combined into useful functions and applications, they can be sold or used to automate work on behalf of the hacker.

Alone bonus: +2 code blocks per hacker

Sell Code

A hacker assigned to the **Sell Code** task can sell 3 **code blocks** and receives a number of **cryptocurrency** equal to the value of the space on the **task track**. Each hacker is allowed a number of sell actions listed on the right side of the task.



Alone bonus: +1 sell per hacker

Trade Secret

The **Trade Secret** task allows a hacker to sell applications, vulnerabilities that their team has confirmed, and application resources a number of times equal to the value of the space they are on.

Each of the following counts as one trade:

- Application resource -> 2 cryptocurrency
- Application -> 10 cryptocurrency
    - If the application being sold was running on a players' computer any **CPU cores** or **GPU cores** are returned to their **CPU** or **GPU**.
    - The bitcoin miner may not be sold.
- Convert 2 **different** application resources to 1 of any other application resource.
- Convert 3 of the **same** application resource to 1 of any other application resource.

1
2
2
3
+1 trade per hacker

Alone bonus: +1 trade per hacker

Build Application

Applications in White Hat represent code that successfully performs a work on behalf of the hacker. Hackers assigned to this task may draw two applications off the application deck per two code blocks spent. For each hacker on that task the player may keep one of those cards.

Alone bonus: draw +2 cards

Upgrade Machine

The **Upgrade Machine** task allows the hacker to either upgrade their computer with a new component or hire a new hacker to join their team.

- 3 cryptocurrency -> purchase one **CPU** or **Memory**
  - Players may purchase up to 3 additional CPUs, for a total of 4 CPUs.
  - Players may purchase up to 7 additional Memory, for a total of 8 Memory
- 8 cryptocurrency -> purchase one **GPU**
  - Players may purchase up to 2 GPUs.
- 15 cryptocurrency -> hire one additional **hacker**
  - A total of 2 additional hackers may be purchased throughout the game, bringing a player's team count to 6 hackers. The newly hired hacker can be placed in the next round's **Place Hackers** task.

In the case of multiple hackers being assigned to this task, each hacker may purchase one upgrade. Multiple hackers may purchase the same upgrade, or they may purchase different upgrades.

Alone bonus: +1 purchase

Enable or Disable Application

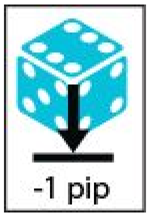
One of the most important aspects of software development is managing the resources of the computer your software runs on. The **Enable or Disable Application** task allows the assigned hacker to either assign a set of **CPU cores** or **GPU cores** to an application, or return a set of **CPU cores** or **GPU cores** to a **CPU** or **GPU**, respectively.

Each hacker either enables or disables one **instance** of one application. An **instance** requires a number of **CPU** or **GPU cores**, represented by the orange and blue squares on the application card. Applications can be enabled more than once, as long as the player has spare cores, and they can be run either on CPU, GPU, or both at the same time. Disabling an instance returns the used **CPU** and **GPU cores** to a **CPU** or **GPU** to be used again by other applications. An application can be disabled by one hacker, and another can be enabled by a second hacker in the same round.

Remember: each **Memory** and **GPU** allows one unique application to run. For example, a player whose **motherboard** has 2 Memory and 1 GPU can only have 3 unique applications running.

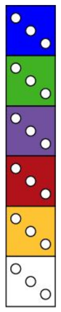
Alone bonus: +1 disable/enable action

Discover Vulnerability



The first step to completing the goal of releasing fixes for vulnerabilities requires discovering a vulnerability. A hacker assigned to the **Discover Vulnerability** task returns a number of **code blocks** to the supply equal to the value of the vulnerability they wish to discover on the **Vulnerability Board**.

A vulnerability is marked as discovered by placing a large die of the player’s color next to the vulnerability being discovered. The die should be placed with a value as follows::

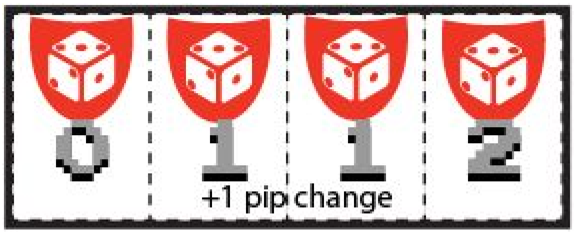


- Vulnerability A: 3
- Vulnerability B: 4
- Vulnerability C: 5
- Vulnerability D: 6

This die represents how difficult it is to confirm and fix the vulnerability.

Alone bonus: -1 pip when placing the die

Pen/QA Test



An often ignored aspect to software development is penetration and quality assurance testing. Penetration testing is the use of hacking to simulate a malicious hacker exploiting a vulnerability. White Hats use penetration testing to confirm whether or not a particular vulnerability exists on their computer. Quality Assurance testing is used to verify that a proposed fix for a vulnerability both fixes the vulnerability and doesn’t cause more issues with the application that the developer might not have expected.

In White Hat, the **Pen Test** task is used to make it easier to **confirm** a vulnerability. In order to **Pen Test**, a hacker assigned to the **Pen/QA Test** task may **reduce** the value of one of their color dice on the **Vulnerability Board** by the value of the space on the **task track** that the hacker is on.

The **QA Test** task is used to make it easier to **release** a vulnerability. In order to **QA Test**, a hacker assigned to the **Pen/QA Test** task may **increase** the value of one of their color dice on the **Vulnerability Board**, but only if that vulnerability has been patched.

Alone bonus: +1 pip change

Confirm Vulnerability

Once a vulnerability has been discovered, it is important to confirm that what is actually happening is exactly what the hacker suspects is happening. This is often done by writing code that purposefully exploits the vulnerability, but in a way that doesn’t do any actual damage to the computer it’s exploiting. Some companies even offer bounties to hackers for vulnerabilities that they have been able to confirm.

The **Confirm Vulnerability** task allows a hacker to attempt to prove that they found the exact cause of the vulnerability. For each hacker assigned to the **Confirm Vulnerability** task, the player names a vulnerability they are attempting to confirm and rolls a die of their color. If the roll results in a value equal to or greater than the value of the die assigned to that vulnerability, the vulnerability has been confirmed. A successful confirmation allows the player to move their **vulnerability die** to the checkmark space on that vulnerability's **patch track**.

Once a player has a vulnerability die on a confirmed space, they may peek at the top card of the associated vulnerability deck at any time. If all players have successfully confirmed a vulnerability, the card is flipped faceup and revealed for easier reference.

Alone bonus: reroll your confirm check on failure

Patch Vulnerability

Free trade

Patching vulnerabilities often requires writing new code, or replacing faulty code, in an application. The face of the top card on the **vulnerability deck** depicts the **application resources** required to patch that vulnerability.

Each hacker assigned to the **Patch Vulnerability** task may make an attempt to patch a vulnerability by rolling **two** dice. Both of the dice's results must be equal to or greater than the value of the die assigned to that vulnerability in order to successfully patch it. If a hacker is successful, they remove the top card of the vulnerability deck for that vulnerability and discard it, along with the **application resources** depicted on the front of the card. The player then takes the **vulnerability die** and moves it to the patch icon on the **vulnerability track** to show that it has been patched. The value on the die should remain the same.

Discarding a vulnerability card means that players in future turns attempting to patch the same vulnerability will require different **application resources**. This simulates the fact that different hackers might find different ways of fixing the same problem.

Players who have previously **confirmed** the patched vulnerability may now peek at the new requirements for the vulnerability at any time, and if all players have confirmed the vulnerability, the new vulnerability card is revealed just like the previous one.

Alone bonus: free trade action to get application resources before patching

Release Fix

Once a vulnerability has been patched, the fix for it needs to be released to the public. This is performed using the **Release Fix** task. A fix cannot be released until it has been run through a **QA Test**. This task may not be assigned until a vulnerability die has been increased to a value of 6. The die is removed from the **vulnerability track,** placed on the highest value bonus score in the center of the **vulnerability board** for that vulnerability. If two or more players patch the same vulnerability on the same turn, they all place their dice on the highest value bonus score. Then, subsequent bonus spaces are not used. Those players score the total of all the spaces for the tie, rounded down.

For example: Red and Blue both patch vulnerability A at the same time. They both put their dice on the 5 point space, then when Purple patches the next vulnerability A they place their die on the 2 point space. Red and Blue split 8 points (the 5 and 3 space) for 4 points each.

Scoring

During the **Check For the End of the Game** function, if a player has released fixes for all four vulnerabilities the game is over and scores are tallied using the included score sheets.

Each player's dice that made it onto the vulnerability board bonus track gains the bonus victory points for that die. Tied players score the points of all that number of spaces, divided amongst the tied players, rounded down.

Each player then one at a time reveals their secret end game goal. All players score the revealed end game goal, up to a maximum of 25 points per goal.

The player with the most points wins. In the case of a tie, the player with the most leftover **cryptocurrency** is the winner. In the case of a further tie, use leftover **application resources**, followed by **code blocks**. If a tie continues, all players share the victory.