Netsim: Network simulation and hacking for high schoolers

Erinn Atwater, Cecylia Bocovich, Urs Hengartner, Ian Goldberg
Cheriton School of Computer Science
University of Waterloo
{*erinn.atwater, cbocovic, urs.hengartner, iang*}*@uwaterloo.ca*

Abstract

This paper presents Netsim, a web-based game intended to teach high school aged children the basics of network routing and how common attacks are performed against it by hackers. Netsim is implemented in the form of a network simulator, with levels depicting how common protocols operate, and accompanying tutorial text explaining the protocol or level. Users craft network packets, with a focus on manipulating the header fields, and inject them into the network via computers they control. Goals of the game include spoofing a source address to steal data, or inducing a smurf attack to perform a distributed denial of service.

We present a technical description of the game and how it is implemented. We provide a case study of our experiences running the game as a workshop for both high schoolers and educators several times, and the improvements we made to Netsim as a result. Netsim is available free and open source, and is also available as a hosted webapp that is free for users to access.

1 Introduction

Netsim is a webapp implementing a real network simulator, with gamification achieved by giving players the ability to craft arbitrary packets and inject them into the network, with the goal of achieving various network reconnaissance/hacking objectives. Players work through a series of levels, slowly introducing them to the simulator and network basics, and build up to performing attacks inspired by real network vulnerabilities, such as address spoofing, denial of service, and smurf attacks. Each level is accompanied by a tutorial describing how the concepts work in real life, explaining the objectives of the level, and giving hints about how to achieve these objectives. The intention of the game itself is to drive interest in computer science, and particularly computer networks and security, by encouraging students to become "hackers" in a realistic game. In contrast to neon-on-dark themed popular hacking games,[1][2] however, Netsim aims to show that network security is primarily an engineering pursuit.

The game can be played alone, but it is targeted at high school teenagers in a workshop setting. The remainder of this section presents an overview of the tool, and an example of how a typical 90-minute workshop might go. Section 2 gives technical details of Netsim, and Section 3 recounts our experiences running the workshop several times (and the subsequent improvements made as a result). Section 4 outlines how we intend to continue improving the game, and Section 5 provides the code and a playable instance.

1.1 Gameplay overview

Netsim levels are composed of three objects: Devices, Links, and Packets. Each level has a pre-specified layout of Devices and the Links that connect them, representing a network topology.

Devices include both computers and networking equipment such as modems, switches and routers. These Devices can only send Packets to each other if they are connected by Links. Some Devices are specified by the level definition as player-controlled, which means the player "owns" that device and can send Packets from it. These player-initiated Packets are created by the player specifying packet header fields in a GUI, and then launching them at any point during the simulation. Other Packets can be pre-specified by the level definition itself and launched at specific simulation times, or can be sent by Devices according to their device *script*. For example, most Devices respond to all ping packets; when received, their script will send a response Packet to the device that sent the ping.

To play the game, players must specify headers that manipulate the Devices into sending/receiving Packets

[1]http://www.introversion.co.uk/uplink/about.html
[2]https://www.exosyphen.com/page_hackerevolution.html

configured in a certain way (as specified in the level definition). For example, players learn in an early level that the "source IP address" need not be written honestly; they are free to send Packets with spoofed source addresses to impersonate other Devices on the network. To win this level, they must send a spoofed packet to the intended target, which triggers the win-condition and allows them to progress to the next level.

The interface for the game (shown in Figure 1) is that of a typical simulator game: it can be paused, slowed down, and restarted. Pausing is a key element to playing the game, as it allows players to take their time and inspect Devices and Packets by clicking on them. This allows them to view the attributes of any Device (namely, its IP address) or Packet (with all of its packet headers, which they may need to copy).

We currently have levels walking players through networking basics, a variety of spoofing attacks, (distributed) denial of service attacks, and an attack on a simple encryption protocol. Some simplifications have been made (vis-à-vis real computer networks) based on our experience with the game. IP addresses are represented as friendly names (such as "Alice", "Bob", or "Router"). We present packet headers grouped into traditional network layers, but omit the link layer entirely. We also ignore the majority of packet headers, and network transmission is automatically lossless and order preserving.

1.2 Typical workshop plan

We traditionally use Netsim as the main activity in a 90 minute workshop for students in grades 9–12. Recruitment was focused on students with "an interest in computer science, but no previous experience". The workshop begins with a short introduction to the history of computer networking, and an explanation of how the Internet is a graph structure of interconnected devices and ISPs. Participants are then directed to the URL for the game, and left to explore the interface and levels for themselves. Each level is accompanied by an in-game explanation of the concepts in the level, and the first few levels are designed to introduce them to the game interface. We have experimented with having the participants work in pairs, but found it works better if each student has their own computer, allowing them to read the accompanying level explanations and experiment at their own pace.

In 90 minutes, several students will complete all the levels while most other students will get close to the end. Periodically, based on how the students are progressing in the game overall, the instructor will walk through some levels on a projector, allowing the students who are struggling to catch up. We also had the privilege of having workshop volunteers with experience in networking and security (CS graduate students) wander around the lab, helping students when they have questions, but also relating concepts from the game to the real world and generally chatting with students about being a student in computer science.

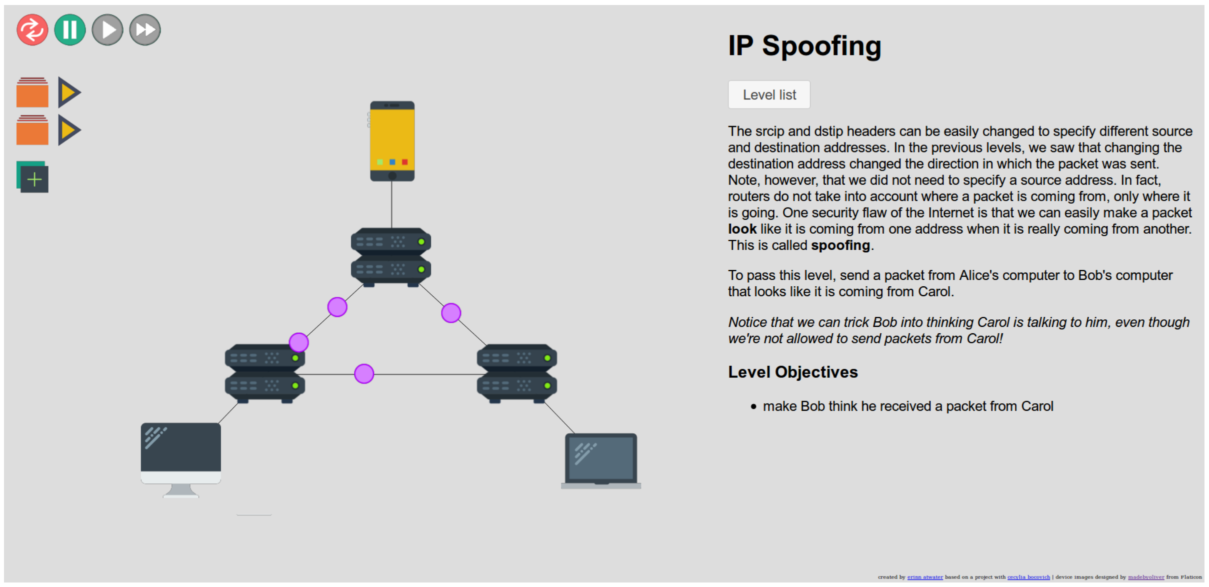
The requirements for this workshop are at least one Internet-connected computer (or tablet) for every pair of students, with a single instructor who has walked through the entire game and is familiar with how to win the levels. Additional teaching assistants are helpful, depending on the number of participants, but the game is designed to allow students to explore on their own with minimal outside assistance. Instructors should test the game on the target computers first, as some older computers may be laggy enough to cause frustration by the players. Dedicated graphics cards make the gameplay smoother, but should not be necessary to run a successful workshop.

2 Technical overview

Netsim is implemented as a web app. The simulator graphics are written using Phaser[3], a javascript game library. The user interface is written using jQuery UI, a javascript webpage framework. The backend, which stores user accounts, player-crafted packets, and level progress, is implemented using PHP and an SQLite database. Level definitions are currently stored as JSON files (which are easy to hand-edit), and the level descriptions as plain HTML files. These choices were made so that the entire game is easy to install by simply copying the files to any webserver that already has PHP installed. The URL for an installed copy of the game can be given to students participating in a workshop, and thus no local installation of any software is necessary on individual computers. The game has worked well so far in all modern browsers and operating systems we have tested, and is even playable on tablets. The only performance issue encountered so far is that some computers animate packet movement in a laggy manner, but this does not render the game unplayable.

The bulk of the game is written in javascript, and the code structure has an emphasis on making it extremely easy to create new levels. An example of a level file is shown in Figure 2. A basic level definition can specify Device objects with X and Y coordinates, whether they are player-controlled, and whether the device should use a device script. Links are defined by the Devices they connect, and Packets can be written as simple JSON objects specifying when in the simulation they should be launched and what their packet headers should be. Win-conditions are specified as triggers: a collection of packet headers that must be received by certain devices to win

[3]https://phaser.io/


IP Spoofing

Level list

The srcip and dstip headers can be easily changed to specify different source and destination addresses. In the previous levels, we saw that changing the destination address changed the direction in which the packet was sent. Note, however, that we did not need to specify a source address. In fact, routers do not take into account where a packet is coming from, only where it is going. One security flaw of the Internet is that we can easily make a packet **look** like it is coming from one address when it is really coming from another. This is called **spoofing**.

To pass this level, send a packet from Alice's computer to Bob's computer that looks like it is coming from Carol.

Notice that we can trick Bob into thinking Carol is talking to him, even though we're not allowed to send packets from Carol!

Level Objectives

- make Bob think he received a packet from Carol

created by erinn atwater based on a project with cecylia bocovich | device images designed by madebyoliver from Flaticon

Figure 1: A typical level in the Netsim simulator.

the level. Device scripts are also written in javascript, which have a simple `onPacketReceived` method defining how to handle incoming packets. A collection of common device scripts (such as an echo server, a switch, and a manually configured router) are already available by name and can be simply referenced from the level definition, or a new device script can be written in the level file itself. Sending a packet from a device script is a simple call to a `sendPacket` method with a javascript object specifying the packet headers. Animating the packet, user interaction with it, users pausing/unpausing the simulation, and passing it to the recipient device are handled by the game engine.

When players create packets through the interface by specifying packet headers, these are also converted to javascript objects by the game engine. They become available in a "launcher" sidebar on the interface, and players can continue to edit them any time, or launch them, which is the same as using the `sendPacket` method that device scripts use. This allows players to iteratively work on their solution for the level until they work out the solution. Players register an account to log in to the game (which only takes several seconds and does not require an email address), so the game can track their progress. It saves any edits players make to their user-defined packets, and which levels they have won so far. The levels themselves typically take less than a minute to play out in the simulator, so it is easy to reset the simulation frequently from the beginning and thus there is no need to track progress mid-simulation.

```
devices:[
        {
                id:"Alice",
                ports:1,
                image:"iphone-1",
                x:0.25,
                y:0.5,
                player:true
        },
        {
                id:"Bob",
                ports:1,
                x:0.75,
                y:0.5
        }
],
links:[
        {
                src:"Alice", srcport:0,
                dst:"Bob", dstport:0
        }
],
timeline:[
        {
                type:"packet",
                at:500,
                from:"Bob",
                payload:{
                        network:{srcip:"Bob",dstip:"Alice"}
                }
        }
],
triggers:[
        {
                type:"packet",
                device:"Bob",
                payload:{
                        network:{srcip:"Alice", dstip:"Bob"}
                }
        }
],
nextLevel:3
```

Figure 2: The definition file for a simple level, which contains two devices: Alice and Bob, connected by a single link. In the timeline section, we see that a packet is automatically launched from Bob to Alice at 500 ms into the simulation. The win condition is triggered when Bob receives a packet from Alice.

3 Case study

In this section, we describe three informal case studies we performed on our educational game. We conducted two sessions with a group of about twenty high-school aged students and an hour-long session with high school computer science teachers. Both the students and teachers had varying levels of computer science background and knowledge. For example, some students in the sessions had done some introductory programming, while others were unfamiliar with computer science but had a general interest in STEM. Based on feedback from both types of groups, we made several changes to the UI and game mechanics. Here we discuss those changes as well as general lessons we learned about the design of educational games and tools.

Before each of the sessions, we gave a brief overview of network security and emphasized that the goal of the game was for the players to explore and discover the various pitfalls and vulnerabilities of common networks. We drew comparisons between the Internet and other modes of communication such as paper mail and telephone networks. The main lessons we hoped students would learn throughout the introduction and the activity were:

1. Systems and their usages change; a protocol or program that was secure in one context or usage may not be secure in another.
2. Designing secure systems (and programs) requires thinking like an attacker. Designers should consider what happens when someone does not use a system the way they are *supposed to*, but rather in ways that they *can*.
3. Learning is about exploration and making mistakes.

After the introduction, we let the students start diving into the game levels. We found that a small number of volunteers (about 5 volunteers for 30 students) wandering around the room to help with questions helped significantly with progress and understanding. In some cases, the students needed clarifications on what certain words meant or more guidance about how a protocol worked. In other cases, they just needed affirmation that they were on the right track or encouragement that it was okay to experiment with the construction and timing of packets: that there is no correct or incorrect way to explore how network protocols work.

The volunteers in our case study sessions were all graduate students in computer security; however, we are currently compiling a set of companion documents for high school teachers that do not have a computer science or security background to educate and assist them in teaching the game concepts and helping with student questions. Additionally, there were some questions and simple clarifications that could be answered quickly with tooltips that contain definitions, explanations, and examples integrated into the game. We are currently consulting with educators of both technical and non-technical backgrounds to decide which terms should be renamed and what tooltips should be added to keep the immersive feel of the game while providing extra information and reducing the reliance on classroom volunteers.

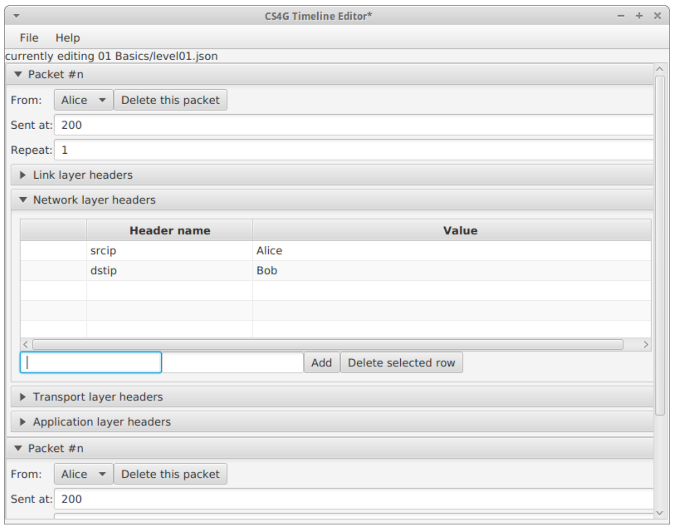
The lessons we learned can be grouped into four main categories: changes to the UI that simplified concepts and made game-play easier, tweaks to game mechanics that immersed the students in the game and gave positive feedback on progress, technical bottlenecks in terms of efficiency and the ease of setting up and distributing the game, and general comparisons between our activity and other computer science teaching tools.

3.1 UI improvements

We noticed shortcomings in the first few iterations of our tool almost immediately into the case study sessions. The largest change we made was to remove any unnecessary memorization of terminology and exact reproduction of key terms. These pitfalls are also seen in the teaching of programming languages to introductory students. It is difficult for students to remember the keywords of various languages and reproduce them perfectly every time. Frequently mistakes such as capitalizations, misspellings, or the misuse of whitespace result in frustration from the student and a lot of work on behalf of the teacher or volunteer to travel around the room debugging student work. Furthermore, the memorization of terms was not a learning objective for this tool; rather, we aimed to teach students how to think about the security of systems and creatively detect potential security problems in existing designs.

In Figure 3, we compare the packet crafting interface of the old and new versions of our game. In the old version, we required students to correctly reproduce both the header name and the value of fields in each layer (e.g., they had to type in “srcip” exactly, as well as the sender’s IP address). This led to confusion, frustration, and mistakes. In our current version, we fix the header name of each field, as these remain constant across levels. This also more accurately reflects reality, in which protocols do not specify the names of fields in protocol headers, but rather rely on byte ordering and header size to craft packets.

Our other main UI improvement was to combine both the packet editor and the simulator into the same interface. In the previous version, players had to switch between two different windows, and save the contents of one window before using them in another. This resulted in students forgetting to save their progress, and



currently editing 01 Basics/level01.json

Packet #n

From: Alice

Sent at: 200

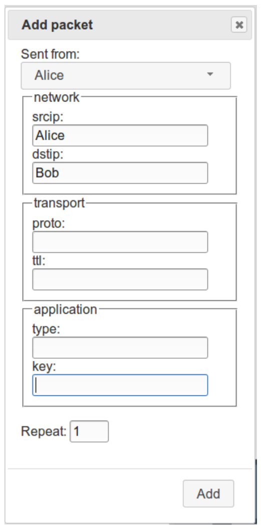
Repeat: 1

Header name	Value
srcip	Alice
dstip	Bob

Packet #n

From: Alice

Sent at: 200



Sent from: Alice

srcip: Alice

dstip: Bob

proto:

ttl:

type:

key:

Repeat: 1

(a) The old packet crafting interface (b) Our new packet crafting interface

Figure 3: A comparison of the old and new interfaces for crafting and editing new packets. We simplified the interface and reduced the information that the player needs to enter to craft a fully functioning packet.

re-running their simulation, only to notice after some time had elapsed that their newly crafted packets were missing.

3.2 Game mechanics

The integration of the packet editor and the timed simulator into the same window and interface also changed the mechanics of our game in a positive way. During the crafting of a packet in the old version, the player had to specify the time in the simulation at which the packet would be released into the network. Many attacks require sending packets at specific times relative to other packets. This led to a lot of unnecessary work on behalf of the students, who had to use trial and error to determine when pre-existing packets were sent. In the new version, the student fills in all the details for a packet, restarts the simulation, and then presses a button while the simulation is running to send the packet. This immerses the student more fully in the game and cuts out unnecessary work in performing timing-dependent attacks.

One of the first changes we made was to add a flashy, positive message (such as fireworks) at the completion of each level. Much in the same way that a security student in university feels that moment of elation when they first drop into a root shell, this sudden message of success provoked a lot of positive emotions in the players of our game. Exclamations of joy and relief, followed by increased determination to tackle the next level, infected not only the successful player but everyone around them. This made the game more of a community experience.

3.3 Technical obstacles

From teachers who wished to use the game and administrators responsible for setting it up on school computers, we received a lot of feedback on how to make the game more deployable and dependable. In our first iteration, we wrote the game in Java. While this provided a nice interface, the new Java version we used did not match the version currently installed on the school computers. Linking to the newer version without also breaking existing programs required a lot of setup time that occasionally ate into the time students could spend playing the game. Additionally, the computers we used had a networked file system. This allowed the students to switch computers in the event of technical difficulties or changes in classroom structure, but also slowed down the execution of desktop programs. We also found deployment of desktop programs to be difficult; we had to compile different versions for different operating systems and address the installation difficulties of each user that depended on the specifics of their setup.

As a result of these challenges, we decided to implement the current version of our game as a web application. It can be played simply by visiting a website, and educators can either use our free hosted copy or host it on their own webservers. The efficiency problems we encountered before with a locally run program on a networked file system are non-existent in this setup. Instead, only a steady Internet connection is needed. For sufficiently large classes, the server can be configured to handle more incoming connections more easily than changing the specifications of each individual computer.

3.4 General lessons

We noted several distinct differences in the reactions of both students and teachers to our sessions, compared to other popular introductions (e.g., beginner programming languages [1, 2]) to computer science or programming. The majority of these differences stemmed from the exploration component of our tool and the positive feedback that players receive at the end of each level. In traditional introductions to programming, students are taught that there is a *correct* and *incorrect* way of designing and writing programs. This mentality is re-enforced by the largely negative feedback of compiler warnings, errors, or unintended behaviour. At the end of these exercises, the student has either succeeded in building a program that matches their expectations or has failed. Beginner-friendly programming languages such as Alice [1] or Scratch [2] gamify this process by letting students visually see the impact of their decisions and correct their mistakes, but the emphasis on correctness rather than exploration remains.

In our game, the main goal is to learn about how packets are sent between machines, how basic protocols work, and how unintended behaviour can be exploited. While each level has a specific success trigger, the crafting of a packet that does not trigger the end of the level is not a failure. It is simply a learning step that gives the player more information about how the Internet works. Furthermore, positive feedback in the form of a successful completion of a level is doled out frequently throughout the game. This differs from traditional programming in which success is evaluated at the end of the class, and mistakes made early in the exercises may adversely affect the student in future steps.

We noticed a marked difference in how students interacted with each other and helped their neighbours progress through levels of the game. Rather than the prescriptive advice a more seasoned student gives in response to compiler warnings in traditional exercises, students of all levels were sharing cool or unusual behaviours they managed to produce by crafting different kinds of packets. The goal was not so much to produce a correct result, as it was to show off unexpected ways to "break" their networks.

4 Future work

We are continuing to develop Netsim into the future, and have a plan in place for its continued development. Many of the improvements are based on feedback and experience from our previous workshop sessions. For example, in our session with educators, they expressed an interest in being able to create new levels. While this can already be done fairly easily by editing JSON files, we intend to add a graphical level editor. We also learn more about how the user interface can be tweaked to be more intuitive every time we run the workshop.

Perhaps most importantly, we are working with the University of Waterloo's Centre for Education in Mathematics and Computing[4] to create a package of teaching materials for non-technologist teachers to run this workshop with. This will include introduction slides, extended background information for teachers on the concepts involved in the game, and consultations with high school teachers from across Ontario.

The hosted version will remain in place as long as there is interest from people in playing it or running workshops. At the time of this writing, more than 1200 people have registered accounts to play the online version. We are also working on a version that does not use PHP at all; this precludes us from tracking players' progress through the levels, but also makes it trivial to install on free hosting services.

Finally, we also have a list of potential future levels to implement. The simulator engine already makes it easy to demonstrate protocols such as DHCP, DNS, and email. These can be used to illustrate concepts such as Internet privacy, and common network utilities such as traceroute. We also envision adding wireless packets to demonstrate concepts like "coffee shop" attacks. This also gives us a place to expand on the concepts we abstracted away from the game, such as numeric IP addresses.

5 Availability

The source code of Netsim is free under the MIT license, and is available on Github at:

`https://github.com/errorinn/netsim`

Educators can download the code and host it on their own webserver, or can use our free hosted copy at:

`https://netsim.erinn.io/`

References

[1] Cooper, S., Dann, W., and Pausch, R. Alice: A 3-D Tool for Introductory Programming Concepts. *J. Comput. Sci. Coll. 15*, 5 (April 2000), 107–116.

[2] Maloney, J., Resnick, M., Rusk, N., Silverman, B., and Eastmond, E. The Scratch Programming Language and Environment. *Trans. Comput. Educ. 10*, 4 (November 2010), 16:1–16:15.

[4]http://cemc.uwaterloo.ca/