Security Scenario Generator (SecGen): A Framework for Generating Randomly Vulnerable Rich-scenario VMs for Learning Computer Security and Hosting CTF Events

Z. Cliffe Schreuders, Thomas Shaw, Mohammad Shan-A-Khuda, Gajendra Ravichandran, and Jason Keighley, *Leeds Beckett University*
Mihai Ordean, *University of Birmingham*

Abstract

Computer security students benefit from hands-on experience applying security tools and techniques to attack and defend vulnerable systems. Virtual machines (VMs) provide an effective way of sharing targets for hacking. However, developing these hacking challenges is time consuming, and once created, essentially static. That is, once the challenge has been "solved" there is no remaining challenge for the student, and if the challenge is created for a competition or assessment, the challenge cannot be reused without risking plagiarism, and collusion.

Security Scenario Generator (SecGen) can build complex VMs based on randomised scenarios, with a number of diverse use-cases, including: building networks of VMs with randomised services and in-the-wild vulnerabilities and with themed content, which can form the basis of penetration testing activities; VMs for educational lab use; and VMs with randomised CTF challenges. SecGen has a modular architecture which can dynamically generate challenges by nesting modules, and a hints generation system, which is designed to provide scaffolding for novice security students to make progress on complex challenges. SecGen has been used for teaching at universities, and hosting a recent UK-wide CTF event.

1. Introduction

Computer security students benefit from hands-on experience applying security tools and techniques to attack and defend vulnerable systems. Practical lab work and pre-configured hacking challenges are common practice both in security education and also as a pastime for security-minded individuals. Competitive hacking challenges, such as Capture the Flag (CTF) competitions have become a mainstay at industry conferences and are the focus of large online communities. CTF activities have been used in education as an effective way of providing and assessing engaging hands-on security challenges, and is often the focus of student hacking society activity (see e.g. [1]–[3]). Virtual machines (VMs) provide an effective way of sharing targets for hacking, and can be designed in order to test the skills of the attacker. Websites such as Vulnhub [4] host pre-configured hacking challenge VMs and are a valuable resource for those learning and advancing their skills in computer security. However, developing these hacking challenges is time consuming, and once created, essentially static. That is, once the challenge has been "solved" there is no remaining challenge for the student, and if the challenge is created for a competition or assessment, the challenge cannot be reused without risking plagiarism, and collusion.

Delivering hacking scenarios to students involves a number of existing challenges, which we aim to overcome: existing pre-configured hacking challenges (such as Metasploitable and those on VulnHub) are typically static and therefore they suffer from limited re-play and reuse, since they only need to be solved once before a solution/write-up is available; and, as a consequence, academic or competitive assessment via pre-developed scenarios is fraught with the risk of hard to detect or prevent plagiarism and collusion.

The typical attempted solution to these issues is the time-consuming process of manually configuring hacking scenarios as vulnerable learning scenarios are required, typically on an event-by-event basis, accepting that each student has the same challenge and the same CTF flags to find.

This is not practical at scale: the network infrastructure and staff costs of running a single two day event is large (see e.g. [5]) and it can be argued that providing a whole cohort of students with appropriate and randomised assessment tasks, across a 12 week course is not practical using traditional methods.

Recently, there has been some related work to randomise security challenges or flags (such as [2], [6], [7]); however, these approaches are focussed on adding randomness to specific challenges or generating random flags that are inserted into static challenges.

We have created Security Scenario Generator (SecGen)[1] which provides a robust framework that can build complex VMs based on randomised scenarios, with a number of diverse use-cases, including: building networks of complex VMs with randomised services and in-the-wild vulnerabilities and with themed content such as business names, employees, and so on, which can form the basis of penetration testing activities; VMs for educational lab use; and VMs with randomised CTF challenges, with randomised (yet meaningful) challenges including real-word vulnerabilities. SecGen has a number of unique features, including a modular architecture which can dynamically generate challenges by nesting modules, and a hints generation system, which is designed to provide scaffolding for novice security students to make progress on complex challenges.

In this paper we describe our aims, present the SecGen framework, including its architecture, configuration language, and use cases, and present evaluation based on using the system for teaching at universities, and hosting a recent UK-wide CTF event.

2. Related Literature

Capture The Flag (CTF) competitions have been popular in the computer security community since the 90s, including the first DEFCON CTF [8]. Other popular annual CTF competitions include those that target university students, such as CSAW CTF [9], [10] and RuCTF [11], those that target high schools, such as PicoCTF [7], [12], others include Ghost in the Shellcode [13], Codegate [14], and UCSB iCTF [15]. The website ctftime.org [16] tracks these CTF events and many more, and lists thousands of teams that take part in competitions on an almost weekly basis. Many CTF events are conducted entirely online, such as DEFCON CTF Qualifiers, and online CTFs often feature a write-up submission, while others are conducted in-person, such as DEFCON CTF, and typically include a live leaderboard.

The most common style of CTF is based on *jeopardy challenges,* where competitors are typically presented with a board of independant challenges, typically with downloads of files for each challenge. Other styles of CTF include attack-defence, where the focus is on attacking or defending systems from attack while keeping services available [17]. In some cases, such as CCDC [18], competing student teams focus entirely on defence, while in other cases, such as RuCTFE [11], teams both patch and defend their systems while attacking others. Attack-defence CTFs often distribute vulnerable systems in the form of VMs. Various forms of games-based learning and gamification (such as leveling-up and leaderboards) have been applied to security education [19], [20]. Gondree *et. al* [21] emphasise diversity of the variety of approaches taken and describe security games as being on a continuum based on task variety and adversarial dynamicity (such as whether teams interact with each other); while acknowledging that this is an over-simplification and that many other game attributes are important.

Capture The Flag (CTF) competitions are a popular means of engaging students with cyber security. The pedagogical benefits of CTF competitions have been widely reported. Efforts to incorporate CTF in higher education (HE) include engaging students in out-of-class CTF activity to cultivate “informal learning spaces” [1], [22], delivering the lab work exercises in the form of CTF-style challenges [2], where flags are revealed where tasks are completed or challenges are solved, and Class Capture-the-Flag Exercises (CCTFs) [23], where teams play-off against each other in regular in-class competitions.

Challenges in running CTF events include the effort required to design and test challenges for quality and appropriate difficulty level, especially where the aim is to ensure accessibility for beginners [9]. The effort required to create challenges and attack scenarios (whether CTF-style or other vulnerable scenarios such as Metasploitable or VMs posted to Vulnhub) is substantial, and time consuming, and as stated earlier, essentially static, making reuse problematic.

Various frameworks for hosting CTFs have been published, such as Facebook CTF (FBCTF) [24], CTFd [25], HackTheArch [26], Mellivora [27], NightShade [28], and picoCTF-Platform 2 [29]. These frameworks typically present jeopardy challenges and scoreboards, and provide administrators a web interface for managing challenges. The iCTF Framework [15] can be used to host attack-defence CTFs, and can generate VirtualBox VMs for each team using setup scripts and vulnerable services that are manually created for each event. The InCTF Framework [17] builds on iCTF Framework to deploy CTFs and teams’ exploits on Docker containers. While these various frameworks lower the barrier for hosting CTF events, the challenges are typically static, and as a result challenges are often not publicly published, and as such each new CTF event involves manually creating new challenges. Furthermore, most frameworks are geared towards jeopardy-style CTFs with either one or no hints, and no existing framework mets our aims, as discussed in the following section.

CyTrONE is a framework that aims to automate environment setup tasks for security education [30]. CyTrONE has a management UI, integrates with LVEs, and YAML specifications state software to install and run, and can include questions and answers to present to

1 SecGen is free and open source software (FOSS) available at http://github.com/cliffe/SecGen

users.

Previous work to provide randomisation to security challenges includes PicoCTF-Platform 2, which includes automatic problem generation (APG), where permutations of challenges can be generated on a per-team basis (or allocated from polls of instances for improved scalability), and served to teams via the web interface [7]. This approach could be used to generate dynamic challenge content (as APG been applied in other disciplines [31]); however, PicoCTF 2014 solely used APG to detect and prevent cheating by generating different flags per-team [7]. Attempts to share flags were detected 1081 times (0.84%). Chothia et. al [2] devised a CTF system to prevent flag sharing by automatically generating separate flags (based on public key cryptography) per-student in a single VM that is distributed to students. Feng developed MetaCTF [6], which provides polymorphic and metamorphic reverse engineering challenges so that students are given unique challenges, and which is designed around jeopardy-style CTF challenges for a HE curriculum, with a scaffolded progression of exercises. Other randomisation of reverse engineering security challenges includes Tigress, which provides dynamic obfuscation of C code [32].

The authors are not aware of any other projects that provide randomisation at the system/VM level for generating randomly vulnerable systems.

3. Aims and Methods

3.1. Overall Aim

The overall aim for for this work was to provide a randomizable, flexible, and general purpose method for specifying and generating VMs for security education and training purposes.

3.2. Use Cases

The educational use-cases include:

- simulations of organisations with a mix of secure and insecure services; with desktop and servers; for simulated security audits;
- security lab exercises; and,
- challenges for CTF events or CTF-style lab work.

3.3. Rich-Scenarios

To achieve these ambitious use-cases, rather than focus on generating standalone *jeopardy challenges* in the form of individual files, the aim of this work is to output a set of VMs, representing *rich-scenarios*. Each rich-scenario can include:

- One or more systems (VMs)
- Complete operating systems, including server and desktop systems
- Networked configuration, including multiple network segments
- Network services (such as FTP, IRC, HTTP, NFS)
- System configuration (such as users and accounts, and software installed)
- Files representing thematic content, such as themed websites
- Software vulnerabilities (including in-the-wild software vulnerabilities, and randomly generated vulnerabilities in protocols or software/websites)
- Configuration vulnerabilities (including misconfigured access controls and services, weak passwords, and so on)
- Data interpretation challenges including steganography, encryption and encoding
- “Loot”, such as flags or simulated sensitive data
- CTF-style challenges (where solving challenges or compromising vulnerabilities such as any of the above leads leads directly to the discovery of flags)

3.4. Randomisation

Randomisation and modular reuse of the above elements is a primary goal. Our aim is to randomise the following:

- *Selection*: randomised selection of the above elements. For example, randomly choosing operating system(s), network configuration(s), service(s), system configuration(s), including user accounts and passwords, with random selection of in-the-wild vulnerabilities or security challenges.
- *Parameterisation*: all of the elements should be able to be configured (for example, the ports services should use, strength of passwords, theme of the scenario), and this configuration will be randomizable.
- *Nesting*: data generation (such as the generation of random flags) and interpretation challenges (such as encoding) should be able to be combined/nested in randomised ways. For example, a flag can be randomly generated, and then encoded in some random way before being leaked via a random software vulnerability.

3.5. Specification Language and Constrained Randomisation

A further aim is to design and implement a scenario specification language, that will randomly generate rich-scenarios for these use cases. Given the significant diversity in potential randomisation implied by our randomisation aims, it is important that the specification language can specify the inclusion of elements and constrain randomisation to meaningful and context appropriate selection, parameterisation, and nesting.

The specification language will be capable of representing the generation of unique security scenarios based on a configurable set of optional constraints: for example, a network of servers, with specific kinds of services (such as a Web server and a file server) with specific kinds of software or misconfiguration vulnerabilities (such as remote code execution and local privilege escalation vulnerabilities). Vulnerabilities and services will be randomly selected and installed on VMs, as specified.

3.6. Student Engagement

The project aimed to engage students, both in development, and in using and evaluating the VMs and learning environments that were generated. We aimed to use our framework to provide rich-scenarios for penetration testing exercises, and to introduce new university student hacking teams to CTFs and as a stepping-stone to taking part in international competitions.

3.7. Development Methodology

Software design and development was led by the primary-author, with a cross-institutional software development team that over time included 10 undergraduate students (6 employed, others working on sub-projects), 1 postgraduate MRes student, and 1 postdoctoral researcher. Additionally, a team of students were employed to develop a range of CTF-style challenges, and adapt CTF challenges from existing security labs (such as [2]).

The software was developed open source using a relaxed Scrum methodology, with a backlog, regular sprint meetings, and task assignment. The current version was always available via Github, and typically members of the team tested each other's code before committing to the master branch.

This approach was designed to engage our students in developing their skills beyond their taught courses, giving them experience in software development, and developing learning materials.

4. Security Scenario Generator (SecGen)

4.1. Introducing SecGen

Here we present Security Scenario Generator (SecGen), which is designed to achieve all of the aims described in Section 3.

SecGen is a Ruby application, with an XML configuration language. SecGen reads its configuration, including the available vulnerabilities, services, networks, users, and content, reads the definition of the requested scenario, applies logic for randomising the scenario, and leverages Puppet and Vagrant to provision the required VMs.

SecGen generates randomised vulnerable VMs that are created based on a scenario specification, which describes the constraints and properties of the VMs to be created. For example, a scenario could specify the creation of a system with a remotely exploitable vulnerability that would result in user-level compromise, and a locally exploitable flaw that would result in root-level compromise. This would require the attacker to discover and exploit both randomly selected vulnerabilities in order to obtain root access to the system. Alternatively, the scenario that is defined can be more specific, specifying certain kinds of services (such as FTP or SMB) or even exact vulnerabilities (by CVE).

This work builds on an early prototype implementation that demonstrated the feasibility of the combination of technologies [33]. The system was re-architected and advanced features were implemented to achieve our ambitious set of aims, and which are described in the following sections.

4.2. Architecture and Modularity

SecGen leverages a number of virtualisation and automation technologies, including Vagrant and Puppet. Vagrant, which is typically used by developers to manage development environments [34], is used to provision VMs, Puppet, which is typically used to manage large scale deployments of servers [35], is used to configure the VMs, and Librarian-puppet is used to manage the deployment of the selected puppet modules. The final output currently includes VirtualBox VMs.

SecGen is designed to be highly modular, with a directory structure and general design philosophy loosely inspired by Metasploit's modular structure. For example, the `modules/vulnerabilities/` directory includes modules representing various vulnerabilities, sometimes directly relating to Metasploit Framework's corresponding `modules/exploits/` modules.

The underlying structure of SecGen is that of a number of "system" objects, which represent VMs (with a Vagrant basebox that is selected based on specified attributes), and each is associated with a list of SecGen "module" objects which are primarily selected based on specified attributes.

Each module has a type (such as vulnerability, service, utility, generator, or encoder), module path, and a associative array of attributes (such as CVE number, difficulty level, CVSS, and so on). Modules can receive data into named parameters (such as port_number or strings_to_leak), either from the output from another module or from data stored in a datastore (variable). Modules can output data, which can be directed at the input of another module's parameters or into a datastore. Modules can include Puppet code which is deployed to and executed on the VMs (as in the case for vulnerability, service, and utility modules), or local code which provides randomisation or transformation of data (as with encoder and generator modules). Furthermore, modules can have default inputs, and dependencies on or conflicts with other modules.

Note that this modular structure is further explained with examples in the following sections.

There are two stages to running SecGen:

Stage 1) building the project output.

Stage 2) building VMs based on the project output.

At Stage 1, all available modules are read, and the scenario definition is also read. The scenario definition is used to select the modules to include for each system. In some cases modules will automatically add other modules to the scenario: either due to a dependency or as a default input to a parameter.

All randomisation happens at Stage 1. Modules that have local code are run to produce output, which is then fed into other modules' parameters.

Librarian-puppet is then used to deploy all of the puppet modules corresponding to the SecGen modules that have been selected into the project output directory. A Vagrantfile is created, which makes reference to all the generated data and puppet modules. Other outputs include files describing the generated scenario, including an XML file listing flags with corresponding hints.

Stage 2 simply involves invoking "vagrant up", which leverages Vagrant to generate and provision the VMs.

4.3. SecGen Modules

The types of SecGen modules are:

- base: a SecGen module that defines the OS platform (VM template) used to build the VM
- vulnerability: a SecGen module that adds an insecure, hackable, state (including realistic software vulnerabilities known to be in the wild or fabricated hacking challenges)
- service: a SecGen module that adds a (relatively secure) network service
- utility: a SecGen module that adds (relatively secure) software or configuration changes
- network: a virtual network card
- generator: generates output, such as random text
- encoder: receives input, such as text, performs operations on that to produce output (such as, encoding/encryption/selection)

The root of a module's directory always contains a secgen_metadata.xml file (illustrated in Figure 1), which defines the attributes of the module. In the case of vulnerability modules, this file contains information about the vulnerability, including CVE, privilege level the successful attacker gains, access level required in order to attack (remote vs local), any metasploit module that can be used to exploit the vulnerability, CVSS score and vector string, difficulty level, and description. This information can be used to filter module selection for scenarios, and also used to specify modules that conflict with each other or to satisfy dependencies between modules.

```xml
<?xml version="1.0"?>
<vulnerability [snip]>
  <name>DistCC Daemon Command Execution</name>
  <author>Lewis Ardern</author>
  <module_license>MIT</module_license>
  <description>Distcc has a documented security weakness
    that enables remote code execution.</description>
  <type>distcc</type>
  <privilege>user_rwx</privilege>
  <access>remote</access>
  <platform>unix</platform>
  <!--module inputs-->
  <read_fact>strings_to_leak</read_fact>
  <read_fact>leaked_filenames</read_fact>
  <default_input into="strings_to_leak">
    <generator type="message_generator"/>
  </default_input>
  <default_input into="leaked_filenames">
    <generator type="filename_generator"/>
  </default_input>
  <!--optional vulnerability details-->
  <difficulty>medium</difficulty>
  <cve>CVE-2004-2687</cve>
  <cvss_base_score>9.3</cvss_base_score>
  <cvss_vector>AV:N/AC:M/Au:N/C:C/I:C/A:C
  </cvss_vector>
  <reference>https://www.rapid7.com/db/modules/
  exploit/unix/misc/distcc_exec</reference>
  <reference>OSVDB-13378</reference>
  <software_name>distcc</software_name>
  <software_license>GPLv2</software_license>
  <!--optional hints-->
  <msf_module>exploit/unix/misc/distcc_exec
  </msf_module>
  <hint>On a non-standard port</hint>
  <solution>Distcc is vulnerable, and on a high port
    number.</solution>
  <!--Cannot co-exist with other installations-->
  <conflict>
    <software_name>distcc</software_name>
  </conflict>
</vulnerability>
```

Figure 1: secgen_metadata.xml

4.4. Scenario Specification

The selection logic for choosing the modules to fulfill the specified constraints can filter on any of the attributes in each module's secgen_metadata.xml file (for example, difficulty level and/or CVE), and any ambiguity results in a random selection from the remaining matching options (for example, any vulnerability matching a specified difficulty level). The filters specified are regular expression (regexp) matches.

As illustrated in Figure 2, the default scenario defines a scenario with a remotely exploitable vulnerability that grants access to a user account, and a locally exploitable root-level privilege escalation vulnerability.

```
<?xml version="1.0"?>
<scenario [snip]>
  <!-- an example remote storage system, with a
    remotely exploitable vulnerability that can then
    be escalated to root -->
  <system>
   <system_name>storage_server</system_name>
    <base platform="linux"/>
    <vulnerability privilege="user_rwx"
      access="remote"/>
    <vulnerability privilege="root_rwx"
      access="local"/>
    <service/>
    <network type="private_network" range="dhcp"/>
  </system>
</scenario>
```

Figure 2: default_scenario.xml

```
<?xml version="1.0"?>
<scenario [snip]>
  <system>
    <system_name>file_server</system_name>
    <base platform="linux"/>
    <vulnerability module_path=".*nfs.*">
    <input into="strings_to_leak">
      <value>Leak this text and a flag</value>
      <generator type="flag_generator"/>
    </input>
    </vulnerability>
    <network range="dhcp"/>
  </system>
</scenario>
```

Figure 3: Module parameterisation

Parameterisation enables modules to be fed input. For example, a vulnerability can be fed information to leak as output. And modules can be nested, so that the output from nested modules are passed into the input for the parent modules. SecGen module parameters are analogous to named and (always) optional parameters. For example, Figure 3 shows a system with a NFS share that will host a publicly exported file containing leaked text, including a generated flag.

Figure 4 illustrates how the flag generator can be nested within an encoder to first encode the flag before it is leaked.

Generators and encoders will always produce/return an (unnamed) array of strings, which can be directed to input parameters for other modules (by parameter name into modules they are nested under, as illustrated in Figure 4). All string encoders will accept and process the "strings_to_encode" parameter, so it's safe to pass input into any randomly selected encoder. It is also possible to direct the output from multiple modules to input to the same module parameter, by nesting multiple modules under an `<input>` element. In which case each of the nested inputs to that same parameter are concatenated into the same array of strings.

```
[snip]
  <vulnerability module_path=".*nfs.*">
    <input into="strings_to_leak">
      <encoder name="BASE64 Encoder">
        <input into="strings_to_encode">
          <value>Leak this text</value>
          <generator type="flag_generator"/>
        </input>
      </encoder>
    </input>
  </vulnerability>
[snip]
```

Figure 4: Nesting encoders

Note that module definitions can specify a set of (potentially nested) modules that should be selected for input to a parameter, if an input is not specified in the scenario. This is illustrated in Figure 1, where strings_to_leak has a generated message as it's default value.

Other advanced features include methods for ensuring modules selected are unique, and using datastores (variables) to hold values for reuse. Datastores are similar to variables in other languages. However, a datastore always holds an array of strings, and writing to the datastore concatenates to the array of strings. Datastores can be used to store generated information for complex scenarios, such as the organisation's name, employees, etc, which can then be fed through to websites, and services, user accounts, and so on.

This specification language has proven to be a powerful method for generating meaningful challenges and systems. However, through our experience with collaborative software development we concede it has a steep learning curve to development.

Access to existing scenarios makes SecGen's barrier for entry low. This removes the requirement for end users of the framework to understand SecGen's configuration specification. Scenarios can be found in the `scenarios/` directory. Developed scenarios include a set of VMs for a randomly generated fictional organisation, with a desktop system, webserver, and intranet server, ready for a security audit; and a set of VMs for hosting a CTF competition; and many other example scenarios.

4.5. Implemented Functionality

Over 100 modules have been implemented to date, which provides functionality that makes the SecGen framework practically useful. 11 service modules provide a range of secure services including NFS, IRC, NTP, SMB, FTP, database, and web servers. 11 utility modules provide various system configurations such as user accounts, firewalls, and desktop environment configuration. 24 vulnerability modules provide a range of vulnerable services, such as vulnerable NFS, IRC, SMB, FTP, SSH, web servers and web apps, vulnerable desktop configurations, access control and system configurations, the majority of which can be deployed either as CTF challenges or to provide open-ended simulations. 45 generator modules can provide content, such as business and user names, addresses and email addresses, messages, filenames and directories, images, ssh keys, passwords, and CTF flags. 13 encoder modules provide various forms of encryption, conversion between data formats, and encoding methods. Network modules provide network cards for scenarios with multiple network segments. The focus has been on deploying Linux systems; however, we have had success testing Windows functionality, which is in development.

4.6. Front End: CTF Website and Hints

A website has been developed to provide a front end to SecGen generated VMs for CTF events. The website provides a scoreboard, timer, flag submission, progress indication, and hints.

SecGen automatically generates a `marker.xml` file, listing all the flags, and for each flag a list of corresponding hints, based on the metadata for the module. Hints range from general hints, such as trying port scans, to progressively more specific hints all the way through to the description of a solution. The approach taken for hints was to penalise points for each hint taken, although the penalties for hints will never exceed the reward for submitting the flag. Where multiple flags are behind the same challenge (for example, differently encoded flags behind the same vulnerability), submitting any of those flags unlocks repeated hints (such as how to exploit the vulnerability).

5. Evaluation

5.1. Rich-scenarios and randomisation

SecGen provides a platform that uniquely and demonstrably achieves the aims described in Sections 3.1 to 3.5. The framework can demonstrably generate highly-randomised VMs based on rich-scenarios.

5.2. Experience Teaching Using SecGen

SecGen has been applied in HE to provide security exercises, from small-scale exploitation exercises through to open-ended audits of a complex set of VMs. Recently a rich-scenario was developed which was used to create targets for team-based security audit projects. The scenario includes a web server, intranet server, and desktop system. The attacker (Kali Linux) VM was placed on the same network segment as the webserver (ie. sharing the same virtual network card), which in turn was connected to the intranet and desktop systems. The students were required to breach the webserver before pivoting attacks through to the other systems. The scenario includes a generated business name, manager, and employees, and involves a random selection of secure and vulnerable services and configurations. A security audit remit was also generated for each team. Student teams followed a security audit methodology and completed a writeup. The output from SecGen was used to assist marking.

5.3. CTF Using SecGen

SecGen was used to generate a set of VMs for use in hosting a UK-wide full-day in-person CTF event. 59 students from 10 universities competed. 3 VMs were generated for the event using SecGen, including one with random decoding challenges, one with a random set of vulnerabilities and image steganography, and another with a root-level privilege escalation. At the end of the event SecGen was presented to participants.

An evaluation survey was run to gauge success of the framework and the event. The response rate was 21 of the 59 participants from 8 of the 10 universities that took part. 52% were postgraduate students, 43% undergraduate (1 reported "N/A"). Many were completing the first year of their degree (38%).

Satisfaction of the event was good, with only one participant responding negatively on the scale of satisfaction. A multiple linear regression analysis was conducted to understand whether the level of satisfaction with the event was impacted by the level of study (not applicable/undergraduate/postgraduate), year of current course (not applicable/first year/mid-course/final year), whether they had taken part in a Capture The Flag (CTF) or other hacking challenge before (yes/no), level of knowledge and understanding of cybersecurity, and sex of the participants (male/female/prefer not to say). All assumptions such as independence of residuals, evidence of multicollinearity, and assumptions of normality were met. Examining all of the independent variables, the overall model that was found to have best fit of the data has $F(3,17) = 3.313$, $p < 0.05$, $R^2 = 0.369$, two of the independent variables (stage of study and whether they had taken part CTF before) have statistically significant contribution in explaining variation (nearly 37%) of the dependent variable (satisfaction with the event) with $p < 0.05$. Table 1 (below) represents regression coefficients with standard errors. The model suggests that in general the participants who are at a later year of study were less satisfied with the event compared to participants at earlier stages of study. The result might support the view expressed in previous research that there is a need to preserve balance between difficulty and ease for designing security competitions with respect to the target audience [5]. The higher satisfaction amongst those that had participated in CTF previously perhaps supports the findings from qualitative data that indicated an appreciation for the uniqueness of the event such as the "attack-format" and use of attack tools, which could be appreciated more by the participants with past hacking challenge experience.

Table 1: Summary of Multiple Regression Analysis on Satisfaction

Variable	**B**	**SE_B**	**β**
Intercept	5.432	0.656	
Level of study	-0.322	0.302	-0.213
Year of study	-0.425	0.169	-0.503*
CTF experience	-0.825	0.355	-0.460*

Note: *p<.05; B=unstandardized regression coefficient; SE_B=Standard error of the coefficient; β=standardized coefficient

81% (n=18) reported that their level of knowledge and understanding of cyber security increased as a result of participating in the event. 81% also expressed an interest in competing in similar events in the future (on a 1-5 Likert scale M=4.43, SD=1.12), with positive but slightly lower interest in online team competitions (M=4.14, SD=1.10), online individual (M=4.10, SD=1.13), and offline (M=3.67, SD=1.35).

The difficulty level was good. During the one day event no team completed every SecGen flag (min=1, max=18, out of 21 possible flags). On a 5 point Likert scale of too easy to too hard, 67% (n=14) selected '3' (not too easy or too hard) (M=3.10, SD=0.7).

The hints system received a mixed response, with participants largely divided over how hints should be implemented in a CTF event. 19% thought the best approach to hints was to have multiple hints per flag - at a penalty (as with the SecGen VMs), another 19% prefered having one hint per flag with no penalty, 19% prefer to have free hints from organisers directly, 14% to have one hint per flag at a penalty, and 29% "Other" with various comments, including an indication that teams avoiding making use of the hints, or that they found the hints unhelpful or too helpful.

Significantly, a large number of those who participated responded that they were interested in making use of the SecGen framework in the future. 86% (n=19) would compete in similar CTF events using SecGen (1 answered "No", 1 other "Not sure"), 72% responded they were interested in browsing the source code to understand the challenges, 63% would use SecGen to generate VMs as personal challenges, 59% were interested in hosting their own CTF events using the framework, and 55% were interested in contributing to SecGen development.

Qualitative data also indicates a positive experience. Multiple participants noted the uniqueness of the "attack-format", and use of attack tools, which was compared to the usual jeopardy format.

Negative comments were focussed on the networking issues that some teams faced, when configuring the VMs that were distributed to teams' own laptops.

Following the event the authors received significant interest in using SecGen to run further CTF events for universities and schools.

6. Future Work

SecGen benefits from the development of further modules to add functionality, such as more vulnerabilities, generated content, encoding methods, and CTF challenges. The authors are developing further SecGen modules and still in the process of converting CTF challenges that have been developed.

Work is in progress to incorporate further digital forensics challenges, and output to forensic disk images, such as E01 files. Related work includes incorporating Microsoft Windows baseboxes and vulnerabilities into SecGen. Work is also ongoing to add cloud deployment of SecGen VMs, specifically to an oVirt-based lab infrastructure. Work is also ongoing to further integrate lab sheet based lab exercises, with randomised worksheets. The platform will be extended with further gamification and immersive scenarios.

7. Conclusion

SecGen provides a flexible and highly modular framework that generates VMs based on scenario definitions that can include randomisation of vulnerabilities (from in-the-wild software vulnerabilities and misconfiguration, to randomised CTF-style challenges), secure services and configuration, and content that can be generated and encoded to provide meaningful *rich-scenario* style challenges. SecGen has been successfully used to enhance security education, by providing randomised targets for lab exercises, large team project security audits, and for generating CTF competition VMs. SecGen can be used to overcome the challenges of generating unique security challenges (and the issues inherent when not randomising tasks given to students), and is free and open source software (FOSS), ready for use in security education. The authors have clear plans for continued development and future work.

Acknowledgements

This project is supported by a Higher Education Academy (HEA) learning and teaching in cyber security grant (2015-2017). Tom Chothia managed University of Birmingham's contributions to the project.

References

[1] A. Mansurov, "A CTF-Based Approach in Information Security Education: An Extracurricular Activity in Teaching Students at Altai State University, Russia," *Modern Applied Science*, vol. 10, no. 11, p. 159, Aug. 2016.

[2] T. Chothia and C. Novakovic, "An Offline Capture The Flag-Style Virtual Machine and an Assessment of Its Value for Cybersecurity Education," in *2015 USENIX Summit on Gaming, Games, and Gamification in Security Education (3GSE 15)*, Washington, D.C., 2015.

[3] C. Eagle and J. L. Clark, "Capture-the-Flag: Learning Computer Security Under Fire," Jul. 2004.

[4] "Vulnerable By Design ~ VulnHub." [Online]. Available: https://www.vulnhub.com/. [Accessed: 05-May-2017].

[5] N. Childers *et al.*, "Organizing Large Scale Hacking Competitions," in *Proceedings of the 7th International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, Berlin, Heidelberg, 2010, pp. 132–152.

[6] W. Feng, "A Scaffolded, Metamorphic CTF for Reverse Engineering," in *2015 USENIX Summit on Gaming, Games, and Gamification in Security Education (3GSE 15)*, Washington, D.C., 2015.

[7] J. Burket, P. Chapman, T. Becker, C. Ganas, and D. Brumley, "Automatic Problem Generation for Capture-the-Flag Competitions," in *2015 USENIX Summit on Gaming, Games, and Gamification in Security Education (3GSE 15)*, Washington, D.C., 2015.

[8] DEF CON Communications, Inc., "DEF CON Hacking Conference - Capture the Flag Archive," *https://www.defcon.org/html/links/dc-ctf.html*, 2013. [Online]. Available: https://www.defcon.org/html/links/dc-ctf.html. [Accessed: 17-Dec-2013].

[9] K. Chung and J. Cohen, "Learning Obstacles in the Capture The Flag Model," in *2014 USENIX Summit on Gaming, Games, and Gamification in Security Education (3GSE 14)*, San Diego, CA, 2014.

[10] E. Gavas, N. Memon, and D. Britton, "Winning Cybersecurity One Challenge at a Time," *IEEE Security Privacy*, vol. 10, no. 4, pp. 75–79, Jul. 2012.

[11] "RuCTF." [Online]. Available: https://ructf.org/index.en.html. [Accessed: 05-May-2017].

[12] P. Chapman, J. Burket, and D. Brumley, "PicoCTF: A Game-Based Computer Security Competition for High School Students," in *2014 USENIX Summit on Gaming, Games, and Gamification in Security Education (3GSE 14)*, San Diego, CA, 2014.

[13] "Ghost in the Shellcode." [Online]. Available: http://ghostintheshellcode.com/. [Accessed: 05-May-2017].

[14] "Codegate CTF." [Online]. Available: http://ctf.codegate.org. [Accessed: 05-May-2017].

[15] G. Vigna *et al.*, "Ten Years of iCTF: The Good, The Bad, and The Ugly," in *2014 USENIX Summit on Gaming, Games, and Gamification in Security Education (3GSE 14)*, San Diego, CA, 2014.

[16] "CTFtime.org / All about CTF (Capture The Flag)." [Online]. Available: https://ctftime.org/. [Accessed: 05-May-2017].

[17] A. S. Raj, B. Alangot, S. Prabhu, and K. Achuthan, "Scalable and Lightweight CTF Infrastructures Using Application Containers (Pre-recorded Presentation)," in *2016 USENIX Workshop on Advances in Security Education (ASE 16)*, Austin, TX, 2016.

[18] NCCDC, "Collegiate Cyber Defense Competition (CCDC)? About." [Online]. Available: http://www.nationalccdc.org/index.php/competition/about-ccdc. [Accessed: 08-May-2017].

[19] J. A. Amorim, M. Hendrix, S. F. Andler, and P. M. Gustavsson, "Gamified Training for Cyber Defence: Methods and Automated Tools for Situation and Threat Assessment," in *NATO Modelling and Simulation Group (MSG) Annual Conference 2013 (MSG-111)*, 2013.

[20] Z. C. Schreuders and E. Butterfield, "Gamification for Teaching and Learning Computer Security in Higher Education," in *2016 USENIX Workshop on Advances in Security Education (ASE 16)*, Austin, TX, 2016.

[21] Mark Gondree, Zachary N J Peterson, and Portia Pusey, "Talking about Talking about Cybersecurity Games," *;login:*, vol. 41, no. 1, 2016.

[22] A. R. Schrock, "Education in Disguise: Culture of a Hacker and Maker Space," *InterActions: UCLA Journal of Education and Information Studies*, vol. 10, no. 1, 2014.

[23] J. Mirkovic and P. A. H. Peterson, "Class Capture-

the-Flag Exercises," in *2014 USENIX Summit on Gaming, Games, and Gamification in Security Education (3GSE 14)*, San Diego, CA, 2014.

[24] "Facebook CTF is Now Open Source!" [Online]. Available: https://www.facebook.com/notes/facebook-ctf/facebook-ctf-is-now-open-source/525464774322241/. [Accessed: 08-May-2017].

[25] "CTFd/CTFd," *CTFd - CTFs as you need them (GitHub)*. [Online]. Available: https://github.com/CTFd/CTFd. [Accessed: 08-May-2017].

[26] "mcpa-stlouis/hack-the-arch," *HackTheArch: A free open source scoring server for cyber Capture the Flag competitions (GitHub)*. [Online]. Available: https://github.com/mcpa-stlouis/hack-the-arch. [Accessed: 08-May-2017].

[27] "Nakiami/mellivora," *Mellivora is a CTF engine written in PHP (GitHub)*. [Online]. Available: https://github.com/Nakiami/mellivora. [Accessed: 08-May-2017].

[28] "UnrealAkama/NightShade," *NightShade - A simple capture the flag framework (GitHub)*. [Online]. Available: https://github.com/UnrealAkama/NightShade. [Accessed: 08-May-2017].

[29] "picoCTF/picoCTF-Platform-2," *PicoCTF-Platform-2: A genericized version of picoCTF 2014 that can be easily adapted to host CTF or programming competitions (GitHub)*. [Online]. Available: https://github.com/picoCTF/picoCTF-Platform-2. [Accessed: 08-May-2017].

[30] Razvan Beuran, Cuong Pham, Dat Thanh Tang, Ken-ichi Chinen, Yasuo Tan, and Yoichi Shinoda, "CyTrONE: An Integrated Cybersecurity Training Framework," presented at the 3rd International Conference on Information Systems Security and Privacy (ICISSP 2017), Porto, Portugal, 2017, pp. 157–166.

[31] D. Sadigh, S. A. Seshia, and M. Gupta, "Automating Exercise Generation: A Step Towards Meeting the MOOC Challenge for Embedded Systems," in *Proceedings of the Workshop on Embedded and Cyber-Physical Systems Education*, New York, NY, USA, 2013, p. 2:1–2:8.

[32] Christian Collberg, "The Tigress C Diversifier/Obfuscator." [Online]. Available: http://tigress.cs.arizona.edu/. [Accessed: 05-May-2017].

[33] Z. C. Schreuders and L. Ardern, "Generating randomised virtualised scenarios for ethical hacking and computer security education: SecGen implementation and deployment," in *1st UK Workshop on Cybersecurity Training & Education (VIBRANT 2015)*, Liverpool, UK.

[34] M. Hashimoto, *Vagrant: Up and Running*. .

[35] S. Walberg, "Automate System Administration Tasks with Puppet," *Linux Journal*, vol. 2008, no. 176, Dec. 2008.