Subject:
Date ۱۳۹۵/۹/۲۱

جلسه ۱

امنیت سیستم‌های نرم‌افزاری

Secure Software Development → مهندسی نرم‌افزار در دیدگاه امنیت سیستم
Software Systems Security

Retrofit vs. Built-in Security

↓ ↓

بعد از توسعه نرم‌افزار — امنیت در دل توسعه نرم‌افزار

Wish → Feasibility → Requirements → refinement / specification → e.g. Use Case → Design → Implementation and Test

Requirements → Security Requirements

Use Case → Misuse Use Case

where is auditing and review?

- معمولاً در نرم‌افزار، امنیت = غیر Nonfunctional است
  ↳ با حداقل‌ها کار می‌کنیم.

- به همین دلیل امنیت، مراحل و روش‌های خاصی برای توسعه نرم‌افزار نیاز دارد.

Course Sections:

1 – Software Security → مقدمه

- Security Problems, security attributes, Vulnerabilities and Attacks, pillars of Software Security, Secured and Safe programming, Security Standards

  ↳ against known attacks
  ↳ against unknown "Hacks"

2 – Security Development Lifecycle → چرخه حیات امنیت

- Security Requirement Analysis, Threat modeling and Risk Analysis, Security Design, Secured and Safe Coding, Security Review of Auditing, Generating the Executables, Security Testing, Secured Deployment, Security Remediations, Security Documentation

PAPCO

Subject:
Date

3 – Security Testing

– Vulnerability Assessment, Security Test Plans, Code Coverage Tools, (Best Practices)

Security Test Cases, Testing Methods and Best Practices, Penetration Testing, Fuzz Testing, Fault Injection

4 – Language-based Security

5 – Security in Mobile Applications

6 – Security in Web-Facing Applications

Textbook: (Recommended Readings)

– Architecting Secure Software, Talukder, charitnya, CRC Press, 2009.

– Software Security: Building Security In., Gary McGraw, Addison Wesley, 2006.

– The Art of Software Security Assessment, Identifying Vulnerabilities ...
Dowd, McDonald, Schuh, Addison Wesley, 2006.

– Threat Modeling: Designing for Security, Shustack, Wiley, 2014.

PQPCO

Subject امنیت نرم‌افزار
Date ۹۵/۶/۲۱

- 24 Deadly sins of Software Security, Howard, LeBlanc, Viega, McGraw Hill, Press 2009.

- Writing Secure Code, Howard, LeBlanc, Microsoft Press, 2004.

- Secure and Resilient Software Development, Merkow, Raghavan, CRC Press, 2010.

Grading:

Midterm 20%
Final 50%
Homework 10%
Projects 10%
Lectures 10%

Informal Assignment:

The Art of ...

Preliminaries:

Vulnerability:

Vulnerabilities are specific flaws or oversights in a piece of software that allow attackers to do something malicious to expose or alter sensitive information, disrupt or destroy a system, or take the control of a computer system or program.

Subject:
Date

آیا باگ‌های امنیتی هم bug هستند؟

Bugs are errors, mistakes, or oversights in programs that result in unexpected and typically undesirable behavior.

تعریف هم باگ‌های امنیتی و هم بعضی از bug‌های غیر امنیتی را پوشش می‌دهد.

چون ممکن است خواسته و سیاست‌های نرم‌افزار ما، فراتر از ویژگی‌های عملکردی را شامل شود.

(Security Policy و Software Requirements)

Security vs. Reliability:

- Security is a subset of reliability. (in general)

- Reliability is continuity of correct service.

تداوم درستی سرویس‌هاست.

یک برنامه reliable است از نظر نرم‌افزاری نباشد یعنی با ورودی‌های Fault هم درست کار کند.

یعنی نرم‌افزار defensively نوشته شده باشد. exception‌ها را هندل کند.

یک برنامه Secure شاید برنامه reliable، robust نباشد. یک برنامه امن باید این‌ها باشد.

با ورودی‌هایی که می‌خواهد نرود تا نرود و شکایت نکند.

Subject: SSS
Date ۹۵/۴/۲۲

Dependability :

دارای ۷ خصوصیت زیر است :

Availability, Reliability, Safety, Integrity, Maintability, and Security.

چیزی که مرتبط است با Dependability and Security است.

- The process of attacking a vulnerability in a program is called exploiting.

"Exploit a vulnerability"

یکی از راه‌ها، روشی خلاقانه برنامه را تغییر دهد تا کنترل برنامه خودش را داشته باشد، اجرای آن (هدف خاصی برنامه را اجرا کند)

خوش شانسی: گاهی اوقات از آسیب‌پذیری استفاده می‌شود.

دستورالعملی که برنامه خاصی برای خود استفاده کند، آن exploit script یا exploit نامیده می‌شود.

Security Policy :

خط‌مشی است که دربرگیرنده رفتارهای مجاز و غیرمجاز را تعریف می‌کند.

مجموعه‌ای از اجراهاست (a set of executions) که در آن خط‌مشی‌ها تعریف می‌شود.

نقض امنیت، به معنای نقض خط‌مشی است.

PAPCO

۵/

Subject: SSS
Date 90/4/12

security automaton, logic, algebra, ...

a formal written document with clauses ← Formal Specification

an informal slightly ambigious collection of people's expectations of reasonable program security behavior (Common Sense)

- Security Expectations: (CIA)

1) Confidentiality → محرمانگی
2) Integrity → صحت
3) Availability → قابلیت دسترسی

عدم دسترسی اطلاعات محرمانه توسط افراد غیرمجاز

Privacy نوعی از محرمانگی است.

یعنی اطلاعات محرمانه از جنس اطلاعات شخصی است.

عدم تغییر غیرمجاز نرم‌افزارها

سرویس‌های ارائه‌شده توسط نرم‌افزار

- Auditing:

ثبت رویدادهای مهم

- Assurance

Subject: SSS
Date 90/6/12

امنیت نرم افزار و روشهای کنترل

۷

راهکارهایی برای auditing:

- Automated Code Analysis
- Security Unit Testing
- Manual Code Audits

(غیری، خارجی)

تعریف Auditing:

Auditing an application is the process of analyzing application code (in source or binary form) to uncover vulnerabilities that attackers might exploit.

روش ها:

- توسعه دهنده قبل از تحویل محصول بازبینی کند.
- شرکت سازنده پیش از انتشار محصول بازبینی کند.
- توسط شخص ثالث وظیفه بازبینی را انجام دهد و گزارش محصول را ارائه کند.
- توسط شخص ثالث بدون بازبینی وارد عمل انجام شود.
- به کاربر نهایی مستقل

دارد Review با Auditing چه رابطه‌ای دارد؟ Auditing نوعی از Review است.

Subject:
Date

- Auditing vs. Black Box Testing:

Black box testing is a method of evaluating a software system by manipulating its exposed interfaces.
(only)

* Automated black box testing is fuzz testing.

fuzz tester → dumb
→ [illegible]

[illegible] malformed [illegible]

[illegible]

Example -

```
struct keyval {
    char *keys;
    char *value;
};
int handle_query_string (char *query_string){
    struct keyval *qstring_value, *cnt;
    char buff[1024];
    if (!query_string) return 0;
    qstring_values = split_keyvalue_pairs (query_string);
```

Subject: SSS
Date ۹۵/۹/۲۲

```
if ((ent = find_entry(qstring_values, "mode")) != null) {
    sprintf(buf, "MODE=%S", ent->value);
    putenv(buf);
    ...
```

environment value

! BufferOverFlow

Code Auditing and System Development Life Cycle (SDLC):

Wish → Need → Requirement

1) Feasibility study →

Subject:
Date ۹۵/۴/۲۸

ازروی نیازمندی ها موردهای تست را بسازیم.

2. Requirements Definition

↳ اهداف کلی
نیازمندی های کلی ← ظاهر شخص در وقت ها

3. Design

↳ طراحی سیستم
مشخص شدن اجزا و ارتباطات بین آن ها

4. Implementation

↳ پیاده سازی
توسعه و نگهداری

5. Integration and Testing

↳

Quality Assurance واسه

اینجا فاز کامل و دقیق انجام شود، قرار نشود محصول بیاد بیرون.

6. Operation and Maintenance

↳ محصول در Deploy

Revision, Updates, User Feedbacks

Vulnerability Class تعریف:

A vulnerability class is a set of vulnerabilities that share some unifying commonality, pattern, or concept that isolates a specific feature shared by several different software flaws.

Subject: SSS

Date ۹۵/۴/۱۸

برای شناسایی آسیب پذیری می‌توانند از سطح ارتباط نرم‌افزاری و فرآیندهای سیستم کمک گرفت.

سه کلاس از آسیب پذیری سطح نرم‌افزار:

1- Design Vulnerabilities (SDLC 1-3)

آسیب پذیری‌های مربوط به فازهای تحلیل، طراحی و توسعه نرم‌افزار

2- Implementation Vulnerabilities (SDLC 4-5)

نقص‌های مربوط به فازهای چهارم و پنجم

3- Operational Vulnerabilities (SDLC 6)

آسیب پذیری‌های ناشی از محیط‌های توصیف نرم‌افزار است.

نقص در

آسیب پذیری‌های ناشی از نقص‌های سطح پایین در ساخت واقعی برنامه (کد) Code

آسیب پذیری‌های عملیاتی از استقرار و مدیریت نرم‌افزار در یک محیط خاص ناشی می‌شوند.

A design vulnerability is a problem that rises from a fundamental mistake or oversight in the software design.

Moto: Design Flaw means software is not secure because of its design!

چون نقص در طراحی باعث می‌شود نرم‌افزار ایمن نباشد، آسیب پذیری ایجاد می‌شود.

محصول ناشی از نقص‌های طراحی نرم‌افزار

طراحی شده به طوری که نباید کار کند!

ارتباط محیط سیستم

ارتباط با سیستم‌های دیگر

ارتباط با نرم‌افزار

11/

Validation vs. Verification

Subject:
Date

↓ stakeholder

, architectural flaws ، high-level vulnerabilities

problems with requirements or constraints

requirement ⟶ design specification

architecture diagrams ;
logic diagrams
process flow chart
interface
class hierarchy

unencrypted ← telnet (مثال

→ SSH

Subject: SSS

Date ۹۵,۴,۲۸

(زبان Safe برای Abstraction خوش حرکت کند)

XSS

Injection

BOF

بسیاری از آسیب پذیری ها از این ضعف ناشی می شوند

دسته اول آسیب پذیری ها: سوء پیاده سازی

In an implementation vulnerability, the code is generally doing what it should, but there is a security problem in the way the operation is carried out.

در این نوع آسیب پذیری عمل درست است اما نحوه انجام آن دارای مشکل امنیتی است.

این ضعف زمانی رخ می دهد که پیاده سازی از مواردی غافل می شود که در طراحی به آن توجه شده است.

(Implementation , Abstraction نامناسب)

دسته دوم آسیب پذیری عملیاتی:

Operational vulnerabilities are security problems that arise through the operational procedures and general use of a piece of software in a specific environment.

ناشی از روش تعامل نرم افزار با محیط و دیگر نرم افزارها.

- نیاز به تعامل نرم افزار با محیط

- وابستگی به دیگر نرم افزارها و سخت افزارها

- نیاز به دخالت کاربران در برخی از مراحل

Theft , Social Engineering -

مثال: در روش های ارتباط امن مانند telnet ، پسورد به صورت رمز نشده ارسال می شود

Subject:
Date

(Gray Areas)

باگ‌هایی هستند که ابزارهای خودکار مثل BOF یا Injection را خوب پیدا می‌کنند.

پس دسته‌بندی ارائه‌شده افرادی که هم‌پوشانی خواهد داشت.

guidelineهایی برای تفکیک دسته‌های تهیه‌شده:

- وقتی درباره آسیب‌پذیری‌ای صحبت می‌کنیم، موارد مثل کتابخانه‌ها، فریم‌ورک‌ها و الگوریتم‌ها هم باید بررسی شوند.

در برنامه‌نویسی، مواردی هستند که ابزارها نمی‌توانند تشخیص دهند مثل race condition و ...

(logic vulns.)

در عملیاتی، مواردی مربوط به استقرار و پیکربندی نادرست مدیریت نادرست لاگ‌ها و ... است.

Common Threads

در بررسی آسیب‌پذیری‌های مختلف متوجه شده‌ایم که رشته‌های یکسانی داریم که به آن Common thread گفته می‌شود.

(رشته مشترک)

- Input and Data Flow e.g. BOF, XSS, Injection

کاربر به‌نحوی داده‌هایی را به شکلی وارد می‌کند که آسیب‌پذیری exploit شود.

برای بررسی این‌گونه موارد، برای کنترل جریان‌ها مثل Data Flow Diagram، Control Flow Graph، Data FG و همچنین System Dep. Graph، Program Dependence Graph

استفاده می‌شود.

↓

چون این‌ها static هستند، Conservative عمل می‌کنند.

در Web App‌ها، Sanitize کردن اهمیت زیادی دارد. ابزارهای کشف آسیب‌پذیری هم از Sanitizer برای ... استفاده می‌کنند.

Subject: SSS
Date ۹۵/۲/۴

- Trust Relationships

وقتی برنامه‌ای که بعضی از مؤلفه‌ها به دیگری اعتماد دارد و اطلاعات آن اهمیت دارد.

داده‌ها اعتماد باید مورد توجه باشد. نحوه ارتباط مؤلفه‌ها هم نیز اهمیت دارد.

- Assumptions and misplaced Trusts

توجه به attackers, environment, supporting programs و ... دارد.

مثلا تصور می‌شود که نام name در فرم حجم دارد و بنابراین به عبارت logical نیست!

- Environmental Attacks

بعضی از حملات ناشی از دستکاری محیط نرم‌افزار است.

مثلا در محیط UNIX و /tmp race حمله‌ای ← در حوزه امنیت است.

- حتی scheduler هم تاثیر دارد.

- Exceptional Conditions

سوء استفاده از نحوه هندل کردن exception ها

مثل Sigpipe در UNIX

این‌ها به thread ها guideline هایی برای نوشتن برنامه‌ها می‌دهد.

P4PCO

۱۵/

Subject:
Date

known Attacks Unknown Attacks

↑ ↑

Secured and Safe Programming موضوع

در برنامه نویسی امن (Secured) یعنی برنامه طوری نوشته شود که در برابر حملات و آسیب‌پذیری‌ها محافظت شود.

در برنامه نویسی امن یعنی برنامه‌ها به منابع (resource) ها دسترسی غیرمجاز نداشته باشند. (Safe)

Safe Prog. ← جلوگیری از Buffer overflow

در Secured Prog. علاوه بر جلوگیری از آسیب‌پذیری‌ها، مسائل امنیتی مثل رمزنگاری، صحت و ... (سرویس‌ها) هم باید پیاده‌سازی شود.

برنامه نویسی Secured یا Safe نیازمند Secure Coding (Programming) است.

Security Standards

موسسات مختلف برای استانداردهای متفاوتی تعریف کرده‌اند.

ITU-T X.509 → Public-key infrastructure (PKI) استانداردی برای

↳ RFC 3280

ITU وابسته به

NIST → (AES) استاندارد برای رمزنگاری

TLS, SSL (RFC2246)

Subject: SSS
Date 90/7/14

سازمان‌های پیشرو CERT در دانشگاه CMU مجموعه‌ای از بهترین روش‌ها و نحوه اصولی آن توسعه می‌دهند و توصیه می‌کنند.

CERT's Secure Coding initiative

↓

چه خطاهای برنامه‌نویسی متداولی وجود دارد.

استانداردهایی برای Secure Coding توصیه می‌شود.

توسعه‌دهندگان نرم‌افزار را آموزش می‌دهند چگونه از این خطاها جلوگیری کنند.

BSI (Build Security In) پروژه

ارائه دستورالعمل‌ها، tools، best practices، ...

OWASP (Open Web Application Security Project) موسسه

ارائه کارگاه‌های آموزشی

تمرکز روی برنامه‌های کاربردی وب

پیشنهاد روش‌های حفاظتی و guideline برای وب

OASIS (organization for the Advancement of Structured Information Standards) موسسه

استانداردهایی روی XML، SGML، CSS و ...

ISO 17799 → ISMS (Information Security Managment systems) استاندارد

(ادامه بخش اول و مقدمه بود)

Subject: امنیت سیستم های کامپیوتری و نرم افزاری
Date

مرور ← مروری بر 24 deadly sins of Software Security

Architecting Secure Software:

(از اینجا به بعد تا آخر کتاب فصل دوم خواهیم بود)

یک استراتژی برای طراحی نرم افزار امن.

چهار دسته کلی از حملات وجود دارد:

1) Unauthorized release of privileged information
افشای اطلاعات محرمانه ←

2) Unauthorized access to resources
دسترسی غیرمجاز ←

3) Unauthorized Modification of privileged information
تغییر غیرمجاز اطلاعات ←

4) Denial-of-Service

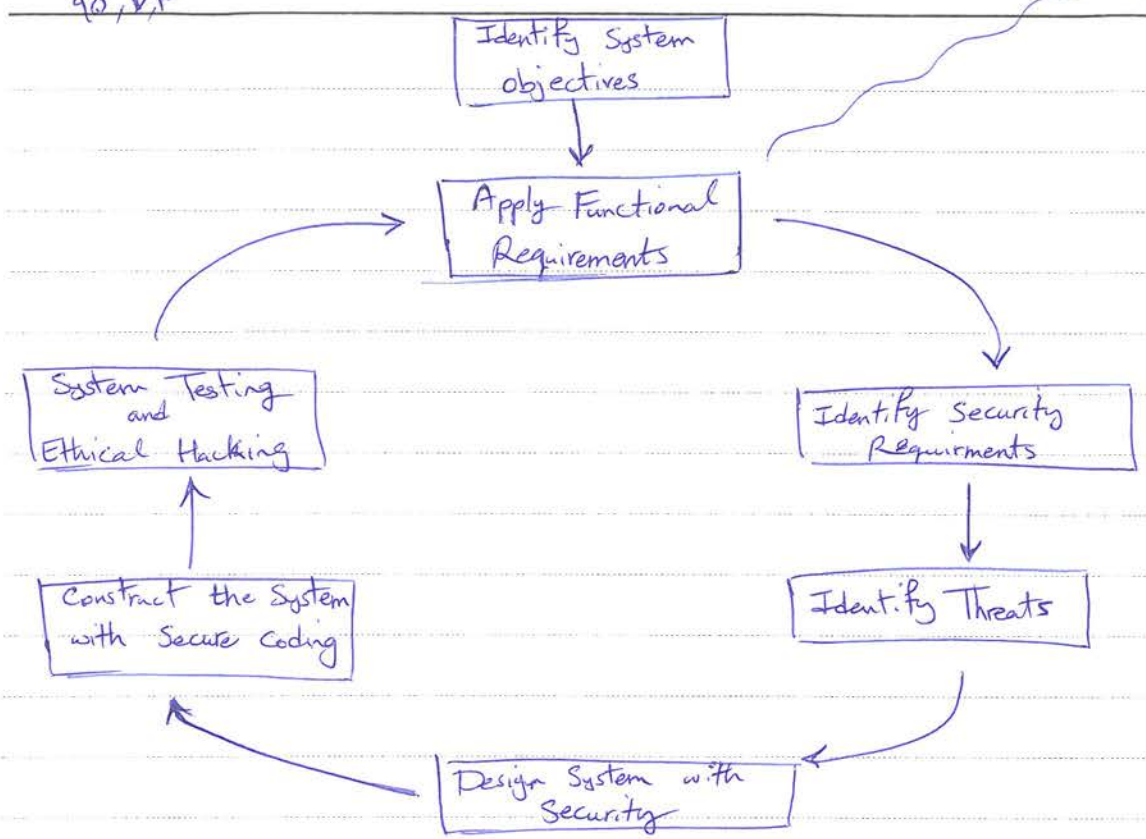
Security Development Lifecycle (SDL):

روشی از مجموعه اقداماتی که توسط تیم نرم افزاری شرکت مایکروسافت مورد توجه قرار می گیرد.

نظر Amoroso از چهار چیزی که در این روش می آید این است که امنیت باید built-in باشد نه after-thought.

اما چهار چیزی که مطرح می شود این است.

Subject: SSS
Date
۹۵/۷/۴



اصل دوم سیستم ایمنی

شاید جزو هیئت های دیگر نرم افزار، کار با شناسایی اهداف سیستم شروع میشود. (این SDL)

باید در گام اول، Security objectives تعیین شود.

Security Expert ندارد. ← من میتوانم

این موارد مشخص میشود:
(کلیه داده ها)

- Identity
  هویت کاربر، اطلاعات شخصی او، برای کدام کاربران
  مورد توجه باشد؟ با چه سطحی؟

- Financial
  ریسک های مالی سازمان در صورتی که اطلاعات
  امنیت نشود، چه مقدار؟

Subject:
Date

- Proprietary and Sensitive data
داده‌های حساس و دارایی‌های شخصی که از آن‌ها در برابر سوءاستفاده محافظت می‌کنیم؟

- Property and Life
چه کاری می‌تواند به دارایی یا زندگی افراد آسیب بزند؟

- Reputation
میزان اعتماد و اطمینان به یک شرکت یا سازمان

- Privacy and Regulatory

- Availability guarantees

- Regulatory
قوانین و دستورالعمل‌ها و آیین‌نامه‌های ایالت‌ها یا کشورهای مختلف

{ Functional → What?
{ Non-Functional → How?

non-functional reqs. است از دسته‌ی چه نوع؟

حال این خواسته‌های non-functional را در خواسته‌های functional کجا می‌توان گنجاند؟

Subject: SSS
Date
۹۵/۷/۴

در مبحث تحلیل، ابتدا باید نیازمندی‌های امنیتی را مشخص کنیم:

Security Requirements Analysis:

Functional vs. Non-functional Requirements

نیازمندی‌های نرم‌افزار به دو دسته کلی تقسیم می‌شوند و ما باید آن‌ها را مشخص کنیم. این دسته‌ها عبارتند از:

functional

آیا دارای تست می‌شود؟

non-functional ← تعریف

functional ← تعریف

Functional Requirement:

A software requirement that specifies a function that a software or software component must be capable of performing.

Non functional Requirement:

A software requirement that describes not what the software will do but how the software will do it. For example, software performance requirments, software design constraints, software quality attributes.

Subject:
Date

- در مرحله refinement، حوزه‌ها به non-functionals و non-functional تقسیم می‌شوند.

حوزه Subjective است.

- بر اساس IEEE تعریف‌شده است. Non func. req. یکی از func. req. ها است.

- یک تعریف جامع از کل آن است که فقط از شامل همه نیازمندی‌ها اعم از functionality، constraints، interfaces، ... .

- است (از دید نرم‌افزار تعریف شده) non-func-req. است.

- یکی از Best Practice های functional ها برای یکی نیازمندی‌ها، یکی Use-case است.

Use-Case Analysis

آن / ن / Func. req. را توان توصیف ساختاری داستان‌گونه.

روی اتفاقات یک سیستم، از دید یک کاربر آن ← که تعامل داستان‌گونه یک کاربر است

↓

آن کاربر، اشخاص

↳ Actor نامیده می‌شود.

↳ e.g. End user, human, program, device, ...

داستان، Scenario نامیده می‌شود.

Subject: SSS
Date 90,V,4

- Use cases capture "who" (actor) does "what" (interaction) with the system for "what purpose" (goal).

- Use case diagrams ، UML در Use-Case ابزار مناسب برای کشیدن

این نمودارها UC هم کنش UC مورد نظر را با یک abstraction بالا می‌برد.

سه نوع رابطه بین UC ها وجود دارد:

Relationships:

include, extends, generalization

A و B نام UC ها را می‌نویسیم.

شماره رابطه‌ها برای راحتی کار

تغییری در UC ایجاد می‌کند.

(optional)

- مثال از جایی که باید از misuse case استفاده کنیم: hacker سیستم، چه خرابی‌هایی در سیستم می‌تواند ایجاد کند؟

misuse case ← این چیزی است که Use Case برای است هم داریم.

برای مطالعه بیشتر Sindre and Opdahl

Subject:
Date

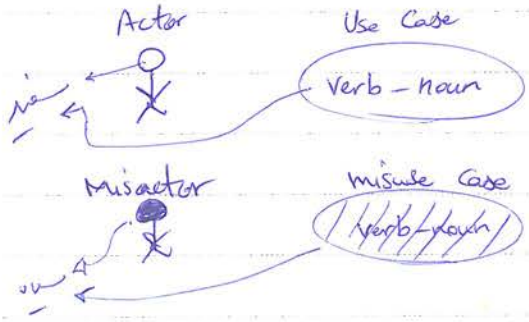
Misuse Case :

توقف نرم‌افزار چه خرابی‌هایی ایجاد شود؟ چه‌کسی می‌تواند؟

↓

misacter

A misuse case is a special kind of use case describing behavior that the system owner does not want to occur.



برای یک توقف کامل باید، دو کیس سناریوی Co-represent و nonfunc-، func. شود.

مثل e-commerce application

کسی می‌تواند از این سایت خرید کند، یک کاربر نرم‌افزار ثبت‌نام کرده، شهر، آدرس

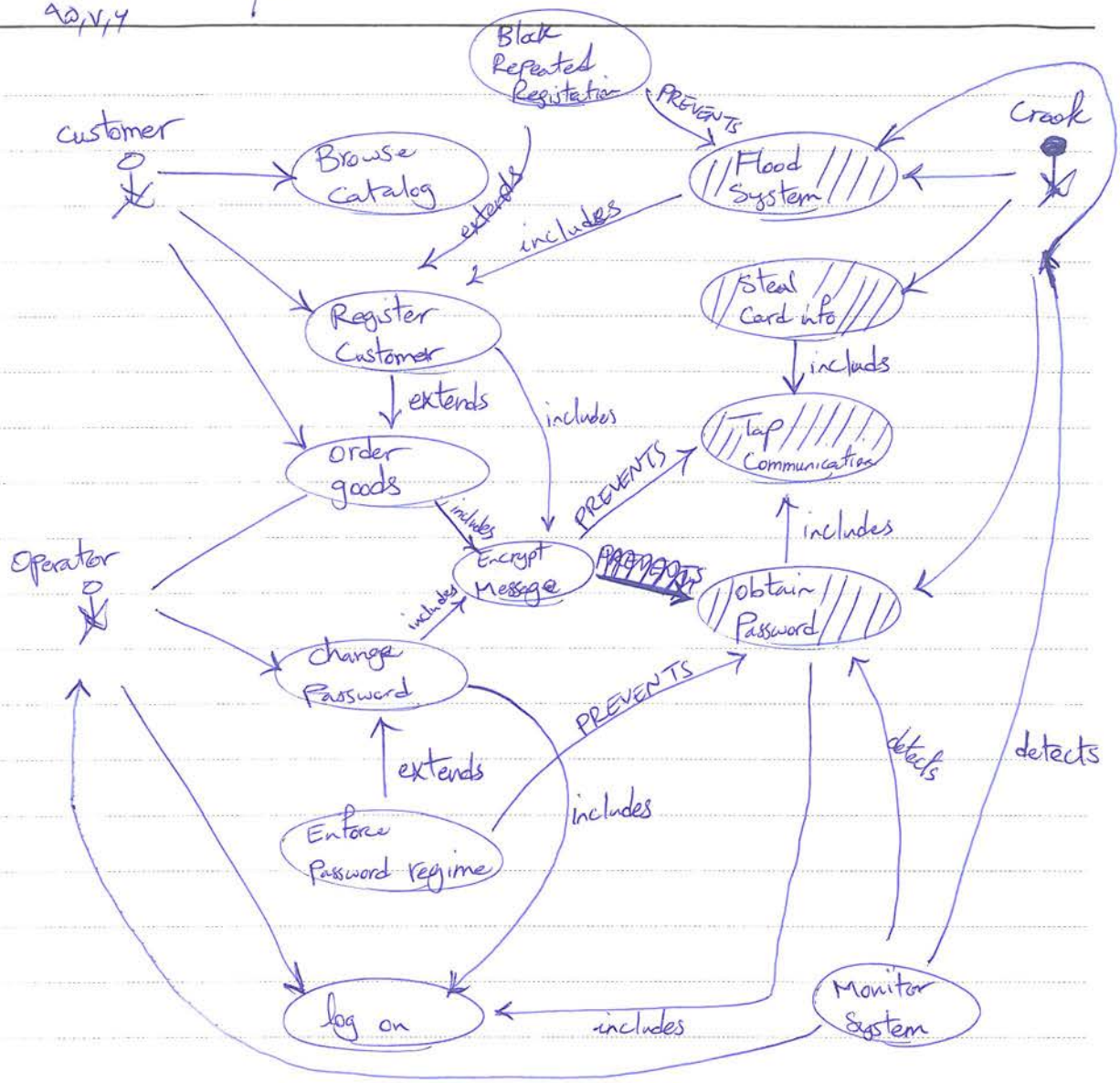
کند، کاتالوگ را ببیند، سفارش می‌دهد که برای آن اطلاعات کارت دارد نرم‌افزار، برای کارت‌خوان، نرم‌افزار

ثبت لاگین کند، ایرادی داریم که کارهایی مثل ثبت توقف حالا را می‌تواند انجام دهد. سایت سرور است و باید مراقبه

تصدیق اصالتی انجام شود.

Subject: SSS

Date 90/V/4



Step 1. First, concentrate on functional requirments through normal actors and the main use cases requested by the actor.

Subject:
Date

Step 2. Look at security-related misuse case. Consider all likely threats for your system.

↳ Threat Modeling استفاده از

شخص کردن تهدیدات

به Misactor ها اضافه کنیم

Step 3. Investigate potential relations between use cases and misuse cases, in terms of "includes" relation.

بررسی کنیم که آیا بین هر کدام از این موارد استفاده می‌شود.

Step 4. Look at security-related non-functional requirements as functional requirments. Introduce new use cases that are necessary to create with the purpose of "detecting" or "preventing" misuse cases.

برای هر misuse case یک use case اضافه کنیم که هدف آن شناسایی یا جلوگیری از آن misuse case باشد.

تهدیدهای خود را بررسی کنید!

Consider all... به معنای در نظر گرفتن همه تهدیدها است.

همه تهدیدها را نمی‌توان شناسایی کرد، به همین دلیل

هم شبیه خود را به guideline های مختلف بررسی کنید، همچنین تهدیدهای دیگر

Subject: SDL

Date
۹۵/۷/۱۴

Iteration علاوه بر اینکه در مدل فرم مثبت عمل دارد، در چرخش و تکرار هم کاربرد دارد.

Identity + Security Req. + Functional Req. → Threats

(SDL در)

Analysis Security Requirement نیز می‌شود.

در فاز دوم مراحل مدل، در نظر گرفتن تهدیدهای امنیتی اهمیت زیادی دارد. همچنین خود این iteration‌های زیاد

بین سه box نوشته (Req. Analysis) تکمیل شده است

اما در این جا Threat Modeling می‌گوییم: (مدل‌سازی تهدید) Threat Modeling (برای اطلاعات بیشتر)
Sehostack آقای

- You use threat modeling to determine security threats. It is a process that helps you to identify, analyze, document and possibly rate the system's vulnerabilities. It is useful for designers to countermeasures.
? protrize

این Threat Modeling در چه کسی است یا طراحی؟ طبق ذات این کار باید سیستم در حال طراحی باشد.

فرض ها هم باعث می‌شود حمله راحت‌تر انجام شود. طبق نظر دشمن، کار است در کجا کمبود سازه‌ای داریم.

من می‌دانم که فرض‌ها را در نظر گرفت.

هیچ اصطلاح واحد خاصی در مورد مدل‌سازی تهدید‌ها وجود ندارد.

PqPCO

۲۷/

Subject:
Date

- Assets : منابع دارای ارزش که نیاز به محافظت دارند

محتوا: tangible مثل پول، سخت‌افزار و ... | intangible مثل اطلاعات حساس، شهرت، اعتبار و ...

- Vulnerabilities : ضعف‌های سیستم

- Threats : رخدادهای بالقوه که می‌توانند به دارایی‌ها آسیب بزنند

(A threat involves one or more vulns)

- Exploit or Attacks : استفاده از یک آسیب‌پذیری برای نفوذ به سیستم

- Countermeasure : اقدامات حفاظتی برای کاهش یا رفع آسیب‌پذیری‌ها

روش‌های حمله:

STRIDE

دسته‌بندی برای تهدیدات امنیتی

stands for:

Spoofing, Tampering, Repudiation, Information Disclosure, DoS, Elevation of Privilege

Spoofing: جعل هویت کاربر یا سیستم دیگر

Tampering: تغییر غیرمجاز داده‌ها

Repudiation: انکار انجام عمل

Information Disclosure: افشای اطلاعات محرمانه

DoS: از کار انداختن سرویس

Elevation of Privilege: به دست آوردن سطح دسترسی بالاتر از حد مجاز

PgPCO

Subject: SSS
Date
۹۵, ۷, ۱۵

Attack Tree:

attack paths ... threat zone ...

(arms race) ...

DREAD

Damage Potential, Reproducibility, Exploitability, Affected Users,

Discoverability:

... loss ... quantify ...

Attack Surface:

STRIDE: ... (noun) a long step you make while you are walking.

DREAD: ... (verb) to walk quickly with long steps)

STRIDE ...

verb: to feel anxious or worried about something that is going to happen or may happen.

noun: a strong fear of something that is going to happen or may happen.

۲۹

Subject:
Date

برای شناسایی و کنترل تهدیدهای این بخش، مدل STRIDE را بررسی می‌کنیم:

1) Spoofing Identity → جلوگیری از جعل هویت توسط adversary

⇓

Countermeasure: Authentication

2) Tampering with data → دستکاری داده‌ها

hash ⇐ راهکار: استفاده از توابع درهم‌ساز

3) Repudiation → انکار انجام عمل توسط dishonest از genuine

⇓

Signature, Auditing

4) Information Disclosure → افشای داده‌ها

⇓

Encryption

5) Denial of Service → اختلال در سرویس‌دهی

⇓

حل با الگوریتم‌های کنترل ازدحام

6) Elevation of privilege → دریافت غیرمجاز مجوزهای بالاتر

⇓

Strong Authorization

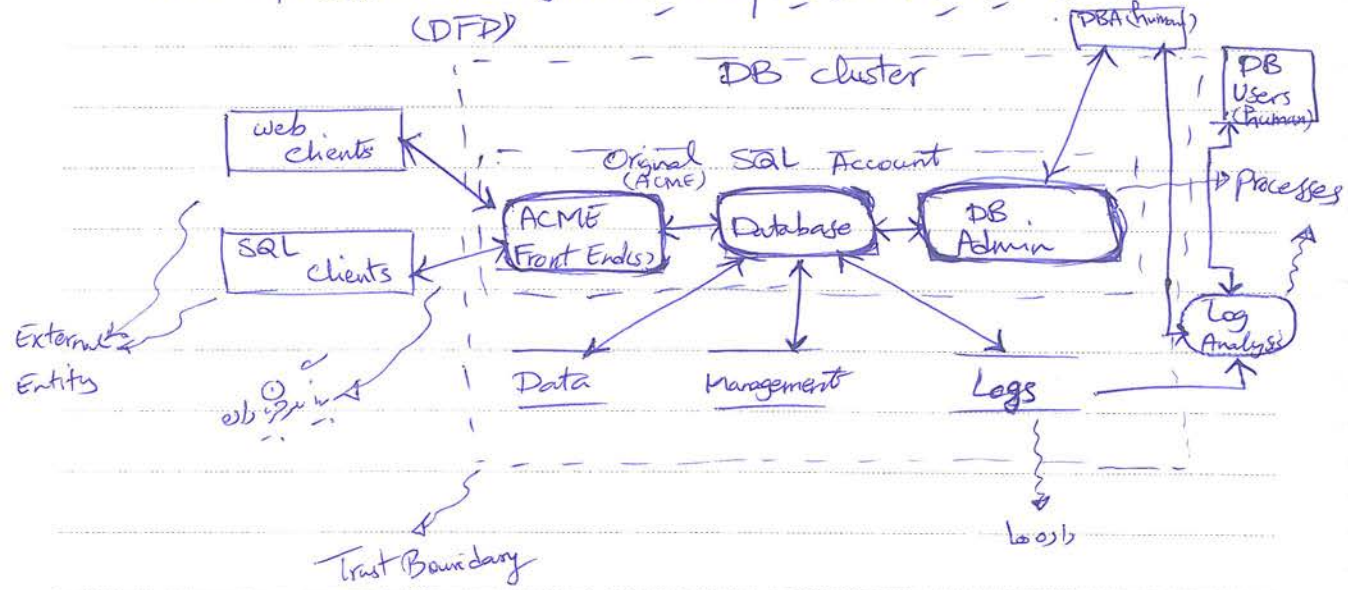
(ریپوزیتوری: Shostack)

Subject: SSS
Date ۹۵/۷/۱۱

IPSec ، ACL ، SSL ، HTTPS مثلاً ← (بزن)

مثال ACME/SQL

می‌توان برای کلیه سرویس‌ها از دیاگرام‌های زیر به صورت Data Flow Diagram (DFD) هم استفاده کرد.



از STRIDE برای مدل‌سازی تهدید استفاده می‌شود.

Spoofing:

Web clients / SQL clients

با دزدیدن هویت آن‌ها می‌توان اطلاعات را تغییر داد.

Front End: باید جلوی Spoof شدن گرفته شود و هویت‌ها را بررسی کند.

Data, Management, Logs: باید جلوی خواندن داده‌ها توسط افراد غیرمجاز گرفته شود.

Tampering: داده‌های موجود در دیتابیس تغییر نکند.

Repudiation: Web client کاری را انجام داده و بعداً نتواند انکار کند.

Information Disclosure: اطلاعات هویت‌ها در دیتابیس نباید فاش شود.

Denial of Service: برای Front End جلوگیری شود.

eavesdropping

Subject: SSS
Date
۹۵/۷/۱۳

Elevation of Privilege: ... not authorized ...

... Command ...

(... ۷۷ ...)

... STRIDE ... variant ... DFD ...

... (STRIDE per element) ... (STRIDE per Interaction)

Attack Trees

... Component ...

... attack Path

... OR , AND ... Composition ...

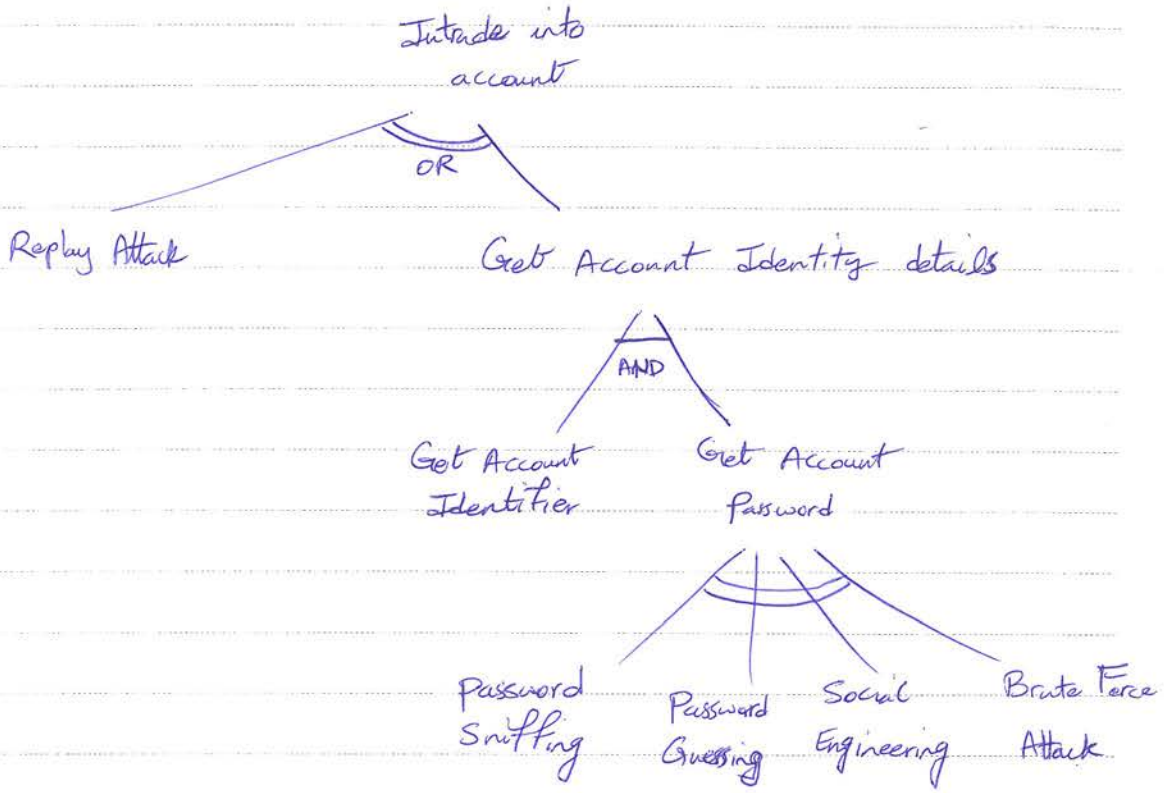
P4PCO

Subject: SSS

Date ۹۵/۷/۱۴

مثل) به روتر دسترسی root (پوشش داده) وقتی وارد شبکه بی‌سیم شدیم احراز هویت انجام شد (SSL VPN -

One_time_password : مکانیزم



Attack Paths:

(Replay Attack)

(Account_identifier, Sniff Password)

(Account_identifier, guess password)

⋮

Subject:
Date ۹۵/۷/۱۸

فصل چهارم - اطلاعات مقدماتی درباره چگونگی روش‌ها

چند نکته درباره تکنیک این روش‌ها هست:

1) decide on representation
(AND-tree, OR-tree, AND-OR-tree)

۲) Completeness اهمیت داره که مطمئن بشیم چیزی رو هم جا نذاشتیم.

۳) روش هرس کردن: اگر در یکی از شاخه‌ها نیازی نبود، بریم جلو و mark کنیم.

۴) زیرگره‌ها تا جای نیاز ادامه پیدا کنند.

۵) ارزیابی روش را در پایان حتما.

(مرجع سوم - روش SSL mind map)

- هدف کلیدی در این مرحله آنه Threat ها رو address کنیم و برای Testing برنامه.

- استاندارد ما STRIDE داره می‌گه این‌ها Abstraction سطح بالایی دارن و باید دقیق‌تر بشن. یعنی

Stage 2 (Threat Modeling) همین الان شروع شد! :)

پایه‌ی این استاک، attack libraries هاشون.

Subject: 885
Date ۹۵/۷/۱۸

Attack Libraries :

- Audience ⟶ چه کسی می‌خواهد استفاده کند.
- Detail vs. Abstraction ⟶ جزئیات در سطح بالا انتزاعی‌تر است.
- Scope ⟶ حوزه‌ای که کتابخانه‌ها پوشش می‌دهند.

Ontology ⟵ تعریف مفاهیم و روابط ⟵ برای کاربرد روش‌های Taxonomy و چگونگی آن

Knowledge Representation

Library of Malware Traffic Patterns (مثال)

Audience : Intrusion Detection Tools

Scope : Web Applications حوزه کتابخانه‌ها : OWASP (مثال)

- تعداد libraryها کم است و ساختن آنها کار سختی است.

شاید بتوان از حرکت در جهت نمایش برداری Vector استفاده کرد و آن را ذخیره کرد.

Subject:
Date

← این اسامی نشان دهنده Taxonomy هستند.

انواع حملات: Library

CAPEC:

(Common Attack Pattern Enumeration and Classification)

مجموعه کامل Structured از ۴۷۶ الگوی حمله در ۱۵ گروه مرتب شده است. مثل:

Data leakage, Attacks Resource Depletion, Injection, Spoofing,

Time and State Attacks, Exploitation of Authentication,

Exploitation of Privilege or Trust, Network Reconnaissance,

Social Engineering Attacks, ...

هر کدام دارای description کامل و Completeness خوب است که نحوه حمله را توضیح می‌دهد و

همچنین دارای مثال و ... است که به ما کمک می‌کند تا

حمله‌ها را بهتر بشناسیم.

(مثال در مورد گروه Injection)

OWASP:

لیست بهترین‌هایی است که راه‌حل‌های امنیتی، ابزارهای تست، آموزش‌ها و ... را برای توسعه‌دهندگان وب ارائه می‌دهد.

این لیست Top 10 دارد و هر کدام را بر اساس impact مرتب کرده است.

Subject: SSS
Date
۹۵/۷/۱۸

ریسک سنجی سرویس ها این صفحه:

Injection, Broken Authentication and Session Management,
Cross-site Scripting, Insecure Direct Objects References,
Security Misconfiguration, Sensitive Data Exposure,
Missing Function Level Access Control, Cross-Site Request Forgery,
Components with known Vulnerabilities, Invalidated Requests and
Forwards.

ریسک سنجی خواهد بود. سپس ریسک حریم خصوصی

آیا از Texas at Austin در schematickov استفاده می کنیم؟

خیر. در این قسمت، ابزارهای حریم خصوصی

Privacy Tools:

آیا در این Threat Modeling ها privacy ابزارها، قسمتی خواهد بود؟

از سه اصل حریم خصوصی می توان از CIA است

Availability

Confidentiality Integrity

PqPCO

۴۷

Subject: Software Systems Security

Date ۹۵/۷/۱۸

چند جای دیگه در مورد حریم خصوصی صحبت شده و تعاریفی ارائه شده که گفتیم مشکل دارند و ...

مروری بر حریم خصوصی چیست و انواع تعاریف و ... که حتماً باید ببینید.

روش‌هایی برای دسته‌بندی حریم خصوصی:

Solove's Taxonomy of Privacy: (Soulouv)

Understanding Privacy

هدف و آسیب‌هایی که impact می‌گذارند و منجر به harm می‌شوند.

- (Identifier Creation) ~~> Solove، Shostak
- Information Collection: Surveillance, interrogation (بررسی بیشتر شود)
- Information Processing: aggregation, identification, insecurity, secondary use, exclusion (عدم اطمینان از امنیت کافی)
- Invasion: intrusion, decisional interference

(فصل سوم Shostack)

DREAD: (مدل تهدید برای ارزیابی ریسک و اولویت‌بندی)

مدل‌های مختلفی برای تحلیل ریسک‌ها وجود دارد که در این جلسه یکی از آن‌ها را بررسی می‌کنیم.

is a method (methodology)

Subject: SSS

Date ۹۵/۷/۲۵

1) Damage 2) Reproducibility 3) Exploitability 4) Affected Users 5) Discoverability

از این ۵ مورد برای اولویت بندی باگ‌ها استفاده می‌شه.

آلوری به اسمای می‌گوید این است ۵ راه حرکت از باگ‌های بیج ۵، عددی بین صفر تا ده می‌دهد و سپس جمع آنها را تقسیم بر ۵ می‌کند. این اعداد از بین از آن باگ‌ها assets ها تاثیر می‌گذارد به باگ را حذری کنیم.

- Damage Potential → از این باگ exploit شود، چقدر خرابکاری می‌شه.

0 → nothing

10 → Complete system or data structure

5 → individual user data compromise or affected

- Reproducibility → چقدر راحت می‌شود باگ را exploit کرد.

هرچه عدد بیشتر، یعنی تکرار راحت‌تر آن

(تکرار سناریوی حمله)

۰ تا ۱۰ بعضی سناریوی خیلی دور کمتر است.

- Exploitability → چه اندازه برای exploit کردن باگ نیاز است.

مثلاً: دقت باگ نیز اهمیت می‌دهیم.

این داشتن سرمایه یا ابزاری برای باگ نیاز دارد، یا نیاز هزینه‌ای می‌خواهد.

↓ Ten 10 ↓ Zero ∅

PqPCO

(تعداد ابزار و تکنیک‌های حمله)

۳۹

Subject:
Date

- Affected Users → چه تعداد کاربر تحت تاثیر این آسیب‌پذیری قرار می‌گیرند؟

0 → none 10 → All users

- Discoverability → چقدر راحت می‌توان این آسیب‌پذیری را کشف کرد؟

خیلی سخت → 0 نیازمند دانش خاص → 5 اطلاعات آسیب‌پذیری در دسترس عموم است → 10

* وقتی از این روش استفاده می‌کنیم که Security Review انجام می‌دهیم و برای آسیب‌پذیری‌ها discoverability تعیین می‌کنیم.

این روش مناسب نیست.

* می‌توان وزن‌های دلخواه برای هر بخش در نظر گرفت.

Attack Surface

(سطح حمله)

بخشی از سیستم که با هکر expose شده است و می‌تواند در آسیب‌پذیری‌ها اهمیت داشته باشد.

The attack surface of an application is the union of code interfaces, services, protocols, and practices exposed to a user (or attacke alike).

مثلا: اگر به کاربر یک سیستم مدیریتی اجازه آپلود فایل در یک WebApp بدهیم

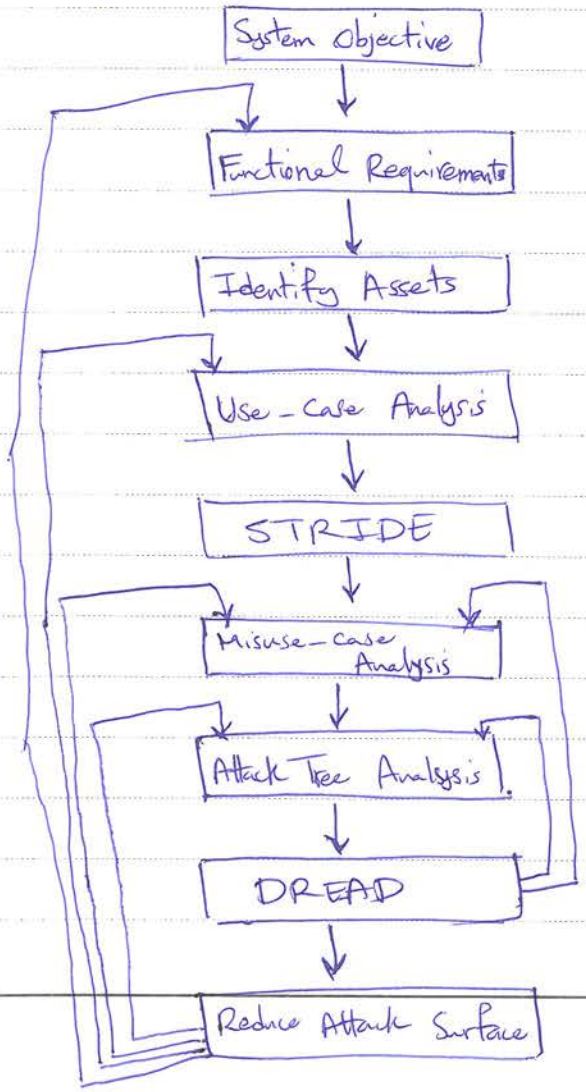
تحلیل attack surface یعنی بررسی functionality ها و بخش‌هایی که به functionality دسترسی دارند.

Subject: SSS
Date
۹۵/۷/۲۵

- آیا این feature مورد نیاز است؟ کاربر چه کاری می‌خواهد؟
- آیا این feature جزء وظایف اصلی کاربر است؟
- چه فرض‌هایی در مورد feature انجام شده؟
- آیا feature های interface مناسب هستند؟

Security Requirements Analysis Lifecycle:

* همان فرآیندهای مهندسی نیازمندی را برای Security Requirement Analysis انجام می‌دهیم.



Subject:
Date

- این refinement سلسله مراتبی (Repetative) یا تکراری (Recursive) است.

- در اولین گام objective های سیستم از صحبت با stakeholder ها به دست می‌آید.

- استفاده از ابزارهای modeling در گام دوم

- در گام سوم، Security objective ها لیست می‌شوند ← CIAAAA یا STRIDE یا ...
  Conf. Integ. Avail. Authentic. Authoriz. Accaunt.

- لیست کردن asset ها و ریسک ها در گام بعدی

- استفاده از درخت‌های حمله (یا سناریوی حمله) برای درک بهتر misuse_case ها

- سپس اولویت‌بندی حمله‌ها را داریم.

- ممکن است چند طرح هزینه‌های جزئی از حمله، یک حمله را در نظر بگیریم.

- هشت سطح حمله وابسته مرار و چگونگی های مختلف در این گونه حملات کنترل سازی آن.

که همه این گزینه‌ها در مرحله کنترل و ارزیابی یک طراحی نرم‌افزار.
Design

حال می‌رویم روی طراحی سیستم‌های نرم‌افزاری صحبت کنیم.

Subject: SSS
Date
۹۵/۲/۲۵

Design:

(Security Design) (در بعضی از منابع این قسمت را جدای از کشن مطرح می‌کنند.)

Best Practices → اصول و راهکارها و بهترین‌ها

(Touch point)

Patterns and Anti-Patterns

Gang of Four!

الگوهایی برای خوبی‌ها

Spagetti Code ⇐ دوری از maintenance

این الگوها نتیجه تجربیات است به وجود آمده است و برای طراحی نرم‌افزار با کیفیت بالا به کار می‌رود.

Security Design Patterns ⇒ الگوهای امنیتی برای این بحث

↳ 1998 dl.
Barkalov, Yuder!!

(gang of 4 الگوهای معروف در طراحی نرم‌افزار مطرح است: (بعضی از الگوهای امنیتی)

(Architecting... الگوهای امنیتی)

pattern در الگوهای امنیتی (الگوهای امنیتی)

Applicability

Structure -

Graphical Representation, الگوهای امنیتی ← Pattern

Classification, نام دیگر -

Also Known as -

Context -

Intent -

Motivation - ← مثال‌هایی در الگوهای امنیتی

Subject:
Date ۹۵/۷/۲۴

- Participants ← لیستی از کلاس ها و کلاس های مورد استفاده
- Collaboration ← تعامل بین عناصر بالا
- Consequences ← trade-off ها و پیامدهای الگو
- Implementation ← نکته های پیاده سازی الگو
- Sample Code
- Known Uses ← استفاده های واقعی الگو
- Related Patterns ← الگوهای مرتبط

accidental ها و monitoring ها را مشکل می کند

و نیاز است بررسی انجام شود. برای مقایسه با الگوهای مطرح شده و best practice های موجود

CERT @ CMU Software Eng. Institute مقاله مرتبط در این حوزه

تعدادی از الگوهای طراحی مطرح شده توسط Yuder, Barcalow را بررسی می کنیم:

1- Single Access Point
2- Check Point
3- Roles
4- Session
5- Full view with error
6- Limited view
7- Secure Access Layer

Subject: SSS
Date
۹۵/۷/۲۴

پترن‌های دیگری هم هستند که در هیچ‌کدام از دسته‌بندی‌های اصلی نمی‌گنجند:

- Least Privilege
- Journaling
- Exit gracefully

پترن‌هایی در راستای هریک از الگوها:

- The single access point pattern recommends that there should be only one point of entry into the system.

مثل پترن Singleton در کلاس‌هایی که فقط یک شیء از آن‌ها ایجاد می‌شود.

- The check point pattern suggests that the user of a system should be validated.

یعنی کاری مثل authorization, authentication و ... که در validation انجام می‌شود.

- The roles pattern

استفاده از نقش‌ها برای کاربران به جای اعطای امتیازات به کاربران که معمولاً کار سختی است.

در این حالت به راحتی نقش‌ها عوض می‌شوند.

مثل Role-based Access Control

Subject:
Date

A

- session remembers the context of a transaction;
It remembers where the user is at any point in time with respect to transaction.

کاری که کاربر باید Session تعریف شود - از ابتدای هر تراکنش.
شروع و پایان مشخص و دقیق تراکنش.

- The Full view with error pattern →
هم کاربر باید بتواند همه گزینه ها را درخواست نشده ببیند و هم می تواند نبیند، بعضا خطا نمایش داده شود.
بعضی طراحی می تواند اشتباهی باشد. ← هر چه می تواند با تنگ شدن اطلاعات شود پس باید گزینه ها را بیشتر کند.

- The limited view pattern → کاربر فقط گزینه ای را می بیند که مجاز هست ببیند
و به چیز دیگری. بعضی طراحی می تواند اشتباهی هم باشد.

پس شاید این دو الگوی اخیر با هدف نقض باشد، ولی منظور این است که یکی از این دو باید حتما در
از نظر طراحی این مشکل مشخص دارد می شود.

- The secure access layer pattern →
نوعی تجزیه و سیستم عامل و ورودی که خروجی سرور را به لایه های بیرونی تحویل این گزینه ها متناسب
مجبور است که با تغییر محیط اجرا، فقط آن لایه تغییر کند و الباقی ایمن باشد.

PqPCO

Subject: SSS
Date ۹۵/۷/۲۶

- Least Privilege (need-to-know)

هیچ جزئی از سیستم نباید از آن چیزی بداند، نباید در اختیارش قرار گیرد.

Any computer program must always remain in a least-privileged state.

اگر یک process ای خواست کاری انجام بدهد در هیچ امتیازات بیشتری می‌خواهد، فقط دقیقاً همان امتیازاتی که لازم است.

این یادمان باشد حتی اصلی امنیتی ساده است!

- Journaling

All usage details must be recorded. (through journal files)

خود همین فایل‌های ژورنال باید محرمانه بماند.

- Exit gracefully

سیستم‌ها باید به صورت خوشایندی شکست بخورند.

آیا exception یا no یا yes به isAuthorized چه تفاوتی می‌کند؟

این اصل می‌گوید (مثلاً در) Exception به حالت no نرو.

(fail secure, fail safe, fail silent, fail operational)

اگر fail کرد، هیچ کاری نکند.

اگر fail کرد به حالت امن برود.

سیستم از چه راهی بتواند از خرابی خود بازیابی کند؟ (Fault-tolerance)

Subject: SSS
Date 25/2/14

* Secure and Resilient ... Merkow, ... (...)

Run with least privilege, Fail securely, Use a Positive Security model, Apply Defense-in-Depth, Default Secure, Infrastructure, Detect Intrusion, Keep Security Simple, Avoid Security by obscurity: CERT Program CMU/SEI

Secure Design Patterns:

reusable ...

- Architectural-level pattern

Focus on the high-level allocation of responsibilities between the different component of the systems and define the interaction between those high-level components.

- Distrustful Decomposition
- Privilege Separation (PrivSep)
- Defer to kernel

Implement. Design Arch.

* Distrustful decomposition:

Intent: The intent of distrustful decomposition secure design pattern is to move separate functions into mutually untrusting programs thereby reducing the

Subject: SSS
Date 93/7/14

- attack surface of the individual program that make up the system.

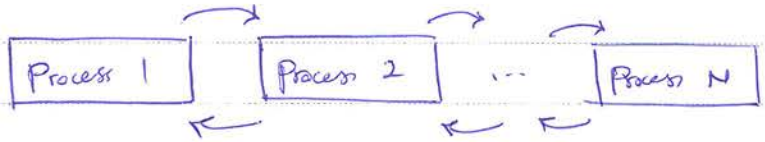
- Functionality and data exposed to an attacker if one of the mutually untrusting programs is compromised.

Also known as: Privilege Reduction

Motivation: ≡

Applicability: این الگو در سیستم‌هایی قابل استفاده است که نیاز دارند برنامه‌ها با سطوح مختلف privilege کنترل شوند.

Structure:



سیستم را به برنامه‌های مختلف تقسیم می‌کنیم و هر برنامه را در سطح امتیازات مختلف اجرا می‌کنیم.

هر برنامه از برنامه‌های دیگر functionality سیستم را دریافت می‌کند. ارتباط برنامه‌ها توسط RPC, Sockets, SOAP, Shared files

Remote Procedure call

یا WebService

Subject:
Date

Consequences: مزایا و معایب این الگو را بیان کنید.

Implementation:

(Process / Privilege model)

One-way: fork() / exec() (in UNIX / Linux)

Create_process() (in Windows Vista)

⋮

Two-way: TCP, SOAP, Shared files

Sample Code: ≡ (Look at the report!)

Builder Factory → Secure Builder Factory

Subject: SSS
Date
۹۵/۸/۲

مثال در Design-Level Patterns : (این الگوها روی سطح کلاس عمل می‌کنند)

Secure Factory:

Intent: to separate the security dependent logic involved in creating or selecting an object from the basic functionality of the created or selected object.

(Similar to Abstract Factory Pattern in Software Engineering)

در Secure Factory Implementation کلاس‌ها و اشیاء Caller باید بررسی شود که آیا دسترسی به کلاس‌ها یا اشیاء Security credential دارد یا خیر.

Caller نباید مستقیماً از کلاس‌ها و اشیاء استفاده کند و باید از طریق Secure Factory این کار را انجام دهد.

Motivation:

یک برنامه ممکن است از کلاسی استفاده کند که رفتار آن بستگی به سطح اعتماد دارد.

یک شیء ممکن است دارای ...

A secure application may make use of an object whose behavior is dependent on the level of trust the application places in a user or operating environment.

مثال: object ← اطلاعات مربوط به سطح اعتماد کاربر

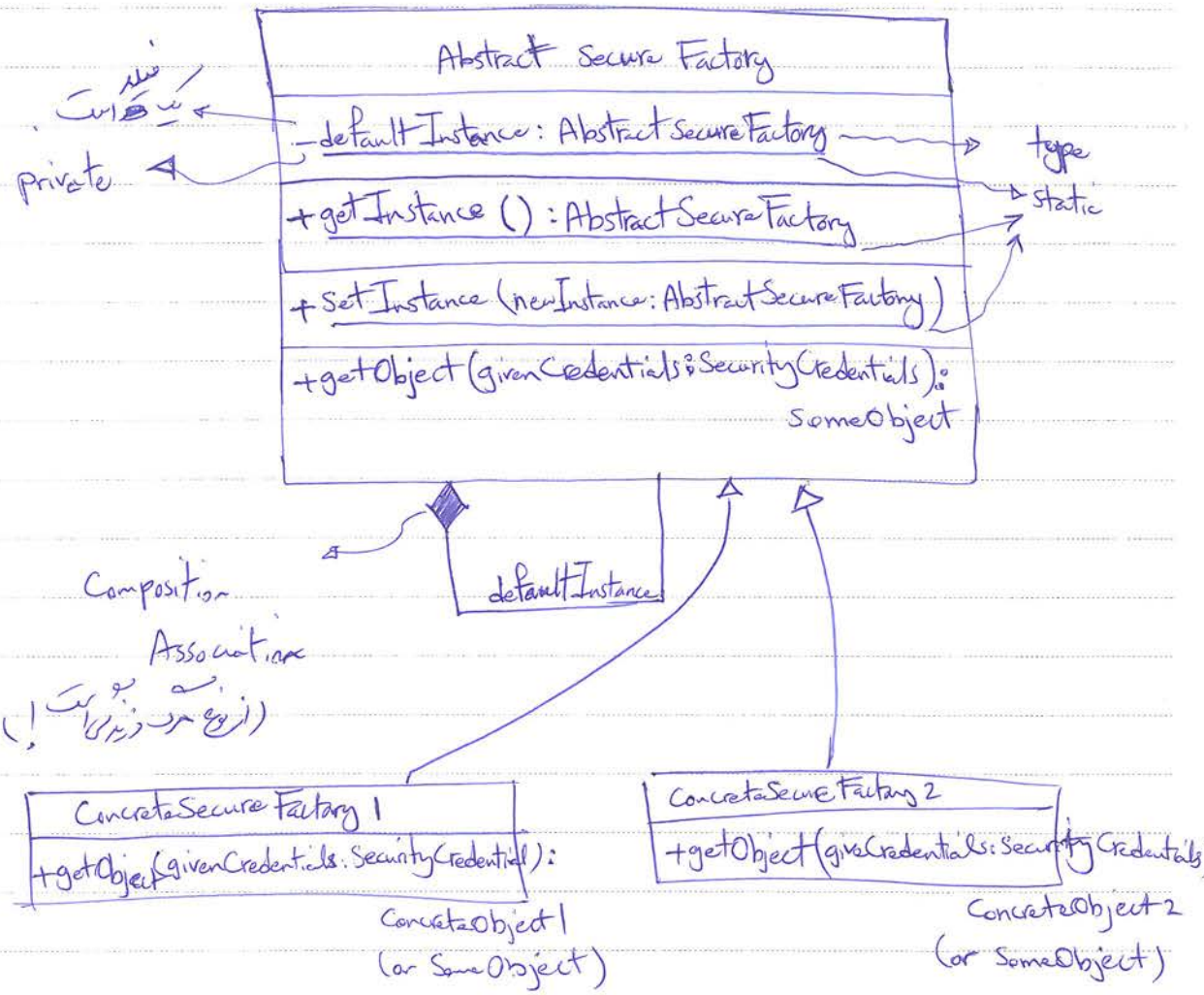
۵۱/

Subject:
Date

Subject: SSS
Date 90/07/2

- The available security credentials contain all of the information needed to select and construct the correct object.

Structure: (class Diagram)



Participants: - client →

- SecureFactory

Subject:
Date

(Abstract class : abstract class)
Interface : abstract

object ... Call ... get object ... Concrete Factory ...

- Security Credentials

representation ... :

- AbstractSecureFactory :

getInstance() ... ConcreteInstance

defaultInstance ... SetInstance()

getObject : Concrete class

- ConcreteSecureFactory N

1, 2, 3, ...

object ...

- SomeObject
- ConcreteObject N

Consequences : obj

obj ... hidden ... black box

PqPCO

Subject: SSS
Date
۹۵/۲/۸

Implementation:

۱) object ای را برای ساخت با این کلاس برگردان، تعیین کنید.

۲) کلاس های Concrete ای که باید برای object ها برگردانده شوند را تعیین کنید.
(ConcreteObject N پیاده سازی)

۳) کلاس AbstractSecureFactory را تعریف کنید.

۴) اطلاعاتی را که برای تعیین هویت کاربر لازم است، تعیین کنید.
(SecurityCredential)

۵) برای تبدیل SecurityCredential ها به ConcreteSecurityFactory ها، یک متد تعریف کنید.

۶) یک ConcreteSecurityFactory را به عنوان defaultSecurityFactory تعیین کنید.

Known Uses: از این الگو در سیستم Secure XML-RPC Server به عنوان یک Library استفاده شده است.

Sample Code: ≡ → ۵ خوانده شود

Subject: threat modeling, requirements و design interplay

Date

CERT ... مطالعه ۱۲ تهدید مطرح شده در

فصل ۱۲ کتاب ... را مطالعه کنید

(Appendix D، صفحه Shostack ...)

Case Study:

تهدید را کاملا درک کنید.

Shostack ... STRIDE ... تهدیدها را با ... بررسی کنید و در فصل ۱۲ ...

... را مطالعه کنید.

1- The Acme Database

2- Acme operational network

3- phones and one-time token authentication

iNTegrity ... تهدید

Acme Database → ... تهدیدات را ...

... Front-end - Sql Client ...

(شکل صفحه ۲۱ همین جزوه)

Security Requirements:

بعضی از نیازمندی‌های امنیتی عبارتند از:

- The product is not less secure than the typical competitor.

- The product can be certified for sales to the U.S. government.

- Non-requirement: protect against DBA.

... نیازمندی‌ها فقط برای ... نیستند و ...

Subject: SSS
Date
۹۵/۸/۳

برای Specific component های هم معماری نیازمندی خوبی که داریم.

سپس باید این ها را بررسی کنیم و تهدیدهای آن را پیدا کنیم.

استفاده از STRIDE برای پیدا کردن این ها

Front Ends: هم تهدیدها و نیازمندی ها را در این بخش خواهیم داشت.

Spoofing: نحوه تحلیل هم تهدیدهای این بخش:

باید authentication هم از Web Clients و هم از SQL clients صورت گیرد.

سپس باید در Operation Guide خبری از امنیت شود تا بتوانیم این را اصلاح کنیم. نوع رمزگذاری هم باید مشخص باشد.

Security

Tampering:

نیازمندی از نوع دسترسی به اطلاعات از طریق امنیتی به درستی مورد تحلیل قرار نگرفته است. باید از authentication

مورد تحلیل قرار گیرد.

Repudiation:

نیاز به logging در این بخش خواهیم داشت. این خبر از log هم هست در Frontend

باید این اطلاعات را به back-end ارسال شود.

Information disclosure:

نیاز به error message های debugging interface هم داریم.

باید این موارد را در Operation Guide بیاوریم.

PqPCO

۵۷

Subject:
Date

Denial of Service:

تکه‌های مربوط به Reliability بررسی کرده.

تکه‌های مربوط به Configuration در guideline ای برای نحوه بررسی کامل تعریف شده.

Elevation of Privilege:

این بخش از ابزار بررسی کرده که آیا برنامه‌ای مثل input validation را درست انجام داده یا نه.

(مثلاً: PIX4)

Connection to Front-Ends:

برای Web client (Front-end) اگر SSL انجام شود، Tampering و Info disclosure بررسی می‌شود.

برای SQL client ها upgrade می‌شود تا SSL اجرا شود. اگر روی SSL نباشد، SSL اجرا می‌شود.

به Special option در بررسی Set می‌شود تا از Acme client برای بررسی SSL استفاده شود. (Web or SQL)

بعضی موارد، برای option ها روی مقدار پیش‌فرض تست می‌شود.

Core Database:

=

(Traceability Matrix)

↓

تمرین سناریوها و راه‌های مواجهه

برای meet شدن.

PqPCO

Subject: SSS
Date
۹۵/۸/۱۴

(Security Design Cookbook : مرجع)

DB Admin :

Authentication Strength :

Account creation :

Creating new DBA accounts will require two administerators and all administrators will be notified.

Sensitive Data :

STRIDE :

Spoofing :

Web Portal uses SSL.

Subject:
Date

DB Admin ها را استفاده از Single Factor Auth. وصل می‌کرد. پس نه از چنین دیتابیس‌ها استفاده کرد.

Tampering:

The DBA can tamper. ← هیچ راهی نیست که!

جون دیتابیس چنین است دیگه.

Repudiation: ادمین می‌تواند لاگ‌ها را تغییر دهد.

← این نیز ربطی به دیتابیس ندارد.

(کاربر)

Elevation of Privilege: راه‌های اضافه!

phone OTT برای کاربر.

← فقط نیست Info disclosure نیز.

Shostak — STRIDE روش iNTegrity

(روش)

A simple file integrity checking tool

ابزاری برای تغییرات غیرمجاز

آیا نیاز یا چیزی از روش‌های مختلف تغییر کرده است؟ checksum علامت است؟

Administrator Console, host component : در نمودار دارد

PgPCO

SSS19413086.pdf ترسیم use case و کلاس‌ها را مطابق توضیحات روی این سند انجام دهید.

Subject: SSS
Date ۹۵/۸/۱۱

کتاب Writing Secure Code از microsoft ، chapters مهم:
کتاب The Art of Security Assesment (Part 2)

هدف این فصل ها از کتب Memory Corruption که در ارتباط است بحث خواهیم کرد. بعضی از آسیب پذیری ها
(Buffer Overrun)
و این فصل ها بر اساس Safe Programming و قواعد آن Secured prog. برای جلوگیری از این آسیب پذیری ها بحث می کنیم برای
فقط جلوگیری نشده، همچنین اطلاعات داده شده.

مقاله Smashing the stack for fun and Profit از Aleph One

Memory Corruption:

از دیدگاه Code review آسیب پذیری ها exploitable یعنی می توان، اما همه قابل بهره برداری نیست. کمی
exploit بودن بحث بعدی از این موضوع است.

Buffer Overflow is a software bug in which data copied to a location in memory exceeds the size of the reserved destination area.

برای کپی حافظه بیشتر از مقدار مشخص شده. اگر سیستم آسیب پذیر باشد قابلیت بهره برداری کامل وجود دارد.

Process Memory Layout:

در فرایندها حافظه به سه بخش تقسیم می شود:

- Program Code ⟵ حاوی دستورالعمل های اجرای برنامه (machine Code) همچنین اشاره گر به دستور بعدی (Instruction Pointer) را نگهداری می کند.
- Shared libraries همچنین اشاره گر به کتابخانه های مرتبط و پویا (Compiled Code) می تواند در این بخش جای گیرد.

Subject: SSS
Date ۹۵/۸/۱۱

- Program Data → (static, global) برای متغیرهای غیرمحلی
(Program Heap)
در این حافظه نگهداری می‌شوند.
داده‌های پویا هستند.

- Program Stack → فضای ذخیره توابع در حال اجرا و ذخیره توابع فراخوانی شده

انباشت. dynamic است.
↓
Stack discipline

روشهای مختلف با کمک این ساختمان داده حافظه مدیریت می‌شود. در این جا نیم Program Stack است.

Stack Overflows:

BOF‌های مشابه باز در Program Stack

Abstract Data Type یعنی stack و operation‌های تعریف شده روی آن

خواص پیاده‌سازی کنیم.

این سه عبارت مختلف اصطلاح می‌شوند: Call stack، program stack، runtime stack
به stack می‌گویند.

activation records هم گفته می‌شود.

activation record (stack frame) که از اجرای هر تابع فراخوانی می‌شود ساخته می‌شود و پس از return پاک می‌شود.

همانند راحل مستقل سکو مدیریت می‌شود و Portable است.

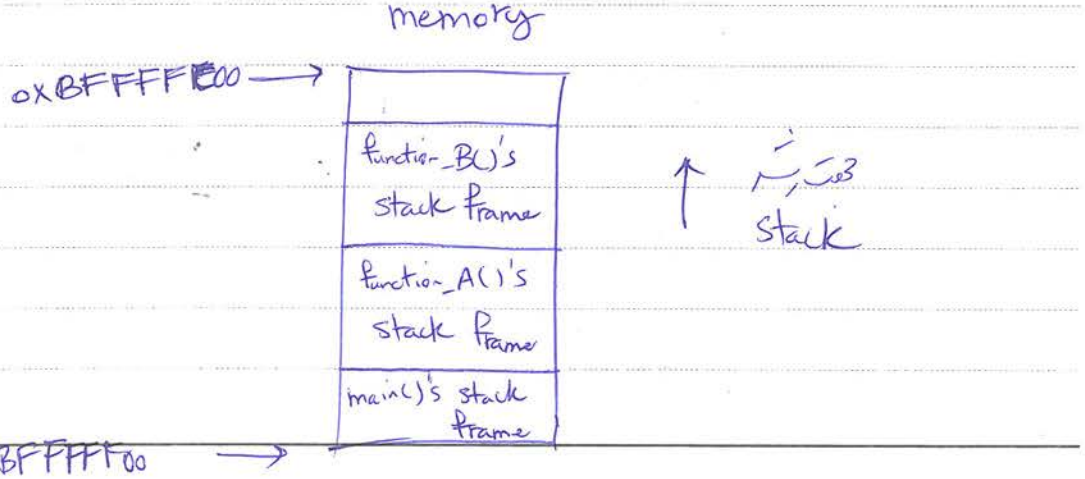
PdPCO

Subject: SSS

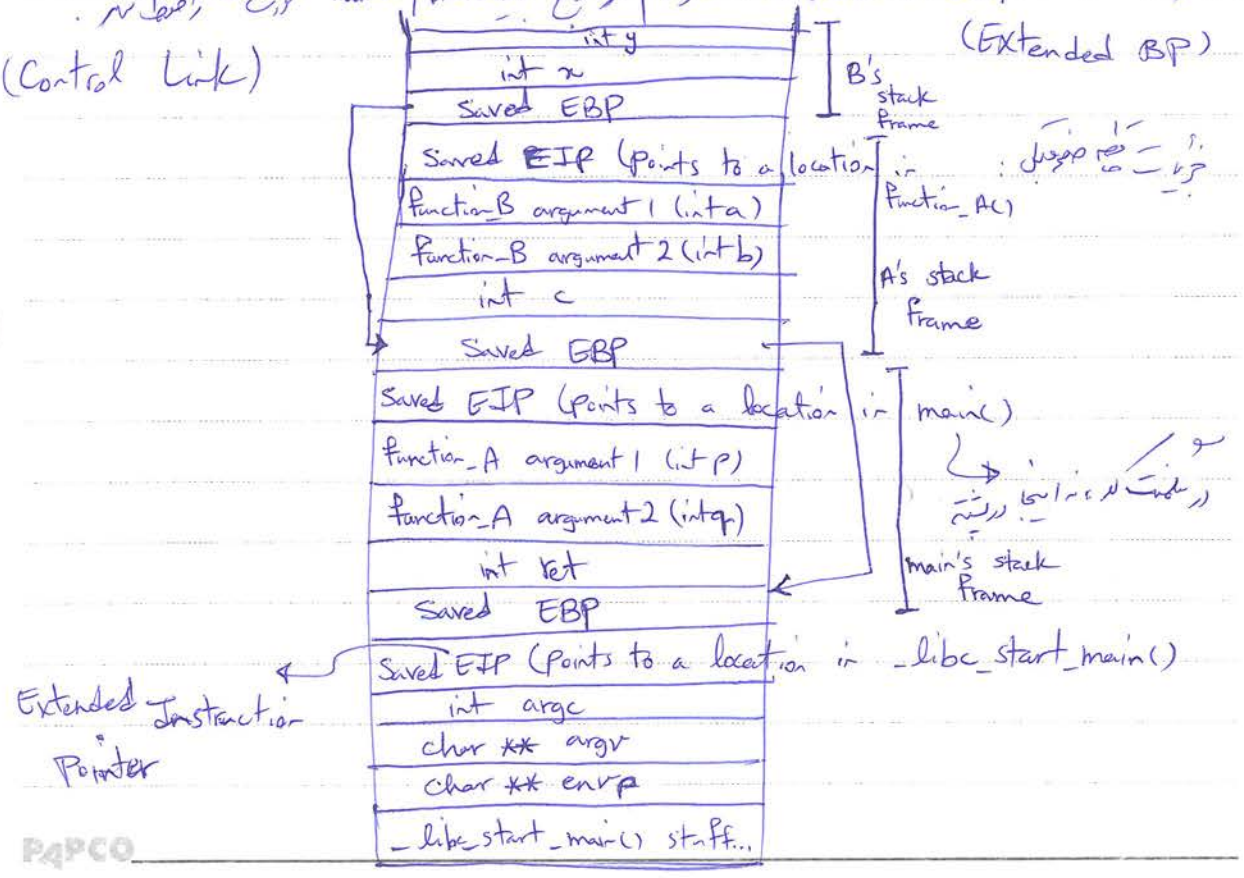
Date ۹۵,۸,۱۱

ESP (Extended Stack Pointer) X86 CPU

```
int function_B (int a, int b) {
    int x,y;
    x = a*a;
    y = b*b;
    return x+y;
}
int function_A (int p, int q) {
    int c;
    c = p*q*function_B(p,p);
    return c;
}
int main (int argc, char **argv, char **envp) {
    int ret;
    ret = function_A(1,2);
    return ret;
}
```



Subject:
Date



Subject: SSS
Date
۹۵/۸/۱۱

هر تابع در ابتدا frame ایجاد می‌کند تا متغیرهای محلی را در آن ذخیره کند:

```
push ebp
mov ebp, esp
```

حال برای سرریز Stack Overflow ؛

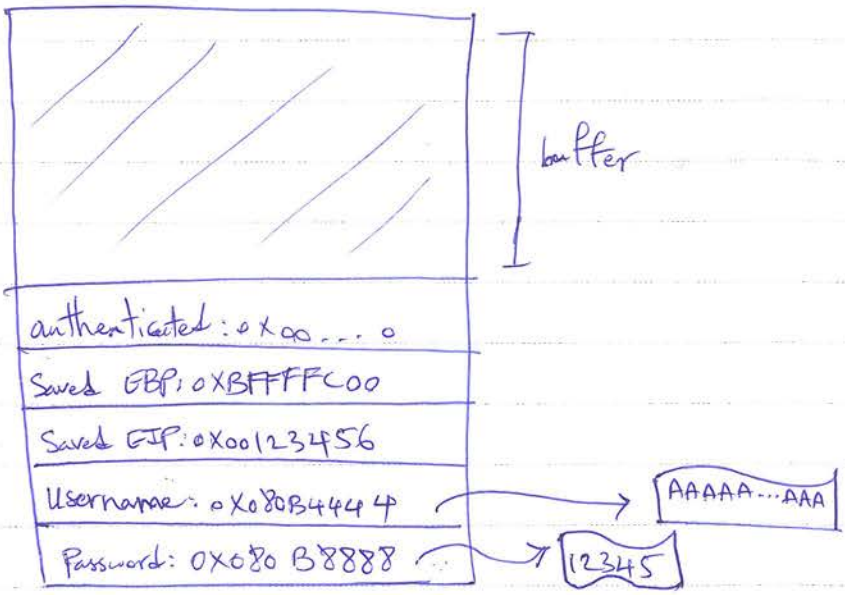
متغیرهای محلی نیز در داخل Stack Frame هر تابع ذخیره می‌شوند.

بنابراین اگر یک بافر در پشته سرریز شود، اطلاعات مهم ذخیره شده روی Stack Buffer نیز تحت تأثیر قرار می‌گیرند.

داده‌ها روی متغیرهای محلی (مثل متغیر Saved EBP ؛ Saved EIP در State) نوشته می‌شوند.

این کار، کنترل جریان اجرای برنامه را تغییر می‌دهد.

```
int authenticate (char *username, char *password) {
    int authenticated;
    char buffer[1024];
    authenticated = verify_password (username, password);
    if (authenticated == 0) {
        sprintf (buffer, "password is incorrect for user %s \n",
                                    username);
        log ("%s", buffer);
    }
    return authenticated;
}
```


buffer
authenticated : 0x00...0
Saved EBP: 0XBFFFFC00
Saved EIP: 0X00123456
Username: 0X080B4444
Password: 0X080 B8888
AAAAA...AAA
12345

Subject: SSS
Date 99/8/14

user-controlled data seeded with executable code

SEH Attacks:

(Structured Exception Handling)

SEH

Exception Handler

Stack

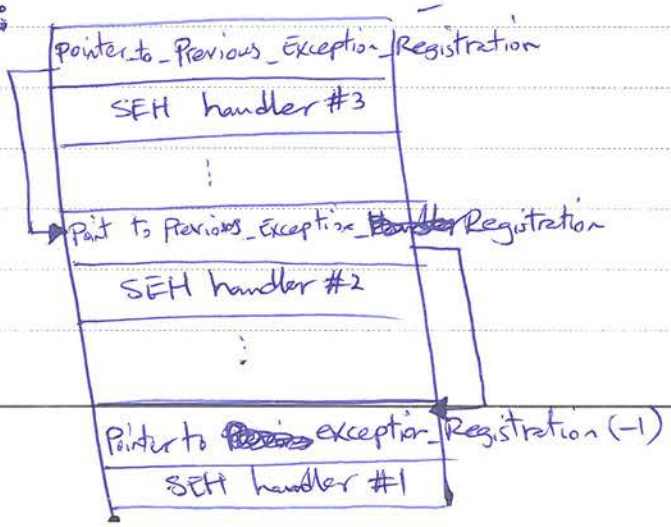
Exception Handler

SEH

Exception handler

(unhandled exception filter).

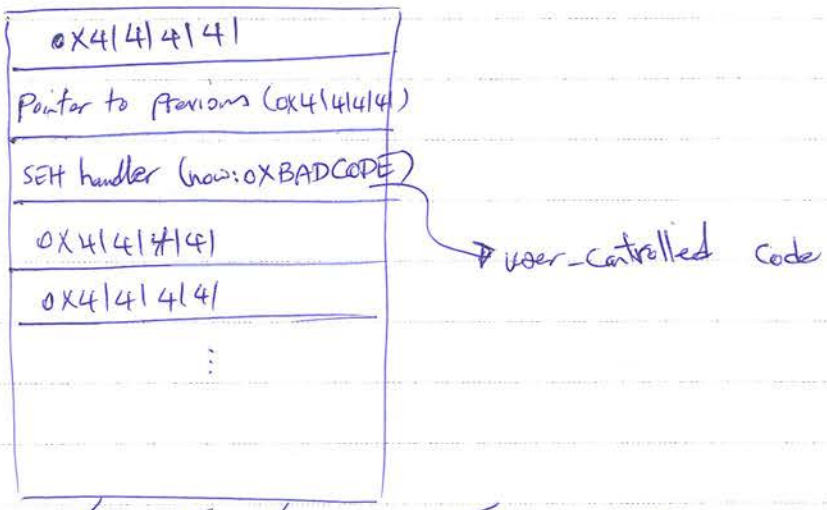
Windows SEH layout:



Subject:
Date

حالا حمله این طوری میشود که هاجم باید یک exception ایجاد کند ... چون shellcode در دسته است ، میتواند هاجم روی آن بنویسد.

SEH Exploit



پس حالا exception بوجود آمد و برای حل آن ، میرود به هاجم را اجرا میکند !

off-by-one Errors :

حالات مختلف حافظه از جمله حلقه ها را اشتباه کرده

پس برای دستیابی به برنامه این باید طول ورودی را از حدی بیشتر نشود validate شود.

```
void Process_String (char *src) {
    char dest[32];
    For (i=0; src[i] && (i <= sizeof(dest)); i++)
        dest[i] = src[i];
...
```

برنامه نویس خوب کدی را اضافه کرده تا حافظه بیش از حد نوشته نشود.

Subject: SSS
Date 95/8/14

(زبان از Abstraction خودش حفاظت نمی‌کند ← زبان برنامه‌نویسی Safe e.g. Java counterexample: C)

خطای برنامه‌نویس همین است که وارد متن حفاظ می‌شود. یعنی دارد حفره BOF ایجاد می‌کند.

می‌تواند بیشتر از حافظه/element بنویسد dest

این از خانه‌های سربرج می‌شود ! sizeofArray

مثال:

```
int get_user(char *user) {

    char buf[1024];
    if (strlen(user) > sizeof(buf))
        die("error: user string too long");
    strcpy(buf, user);
    ...
}
```

Stringها خانه اضافی null (10^{zero}) دارند.

یعنی strlen تا null می‌شمارد اما strcpy تا null کپی می‌کند

اگر String بیشتر از 1024 بایت طول داشته باشد، حافظه خراب می‌شود

1024 بایت از حافظه را اشغال می‌کند و می‌رود.

Shell code:

BOF به هکر اجازه می‌دهد که کنترل حافظه‌ای که داده‌های برنامه در آن هستند را در دست بگیرد و بعد خودش کد دلخواه خود را اجرا کند.

Shell code

کدی که هکر اجرا می‌کند و داده‌های دیگر آن در حافظه هستند را shell می‌نامند، چون معمولا هکر shell را launch می‌کند.

49

Subject:
Date

برای shellcode:

```
char *args[] = {"/bin/sh", NULL}
```

execve("/bin/sh", args, Null) → دستور اجرای برنامه

envp

که مربوط باید ← فراخوانی از library ، باید اول که برنامه‌ها locate کرد.

اینجا این شکل است. چون execve خود برنامه خودش را فراخوانی می‌کند، دیگر

و قسمت‌های برنامه که مربوط را انجام دهیم. این چند خط اسمبلی مربوطه هستند که باید:

```
xor1  %eax, %eax  → صفر کردن
mov1  %eax, %edx  → Edx = envp = NULL
                    ↳ environment parameters
mov1  $address_of_shell_string, %ebx  → Ebx = Path parameter
mov1  $address_of_argv, %ecx  → ECX = argv
movb  $0x0b  → syscall number for execve
int   $0x80  → interrupt for syscalls
```

حالا این را چطور در shellcode بیاوریم.

حالا این را در memory چطور بیاوریم؟

چطور این کار را کرد:

```
jmp end
code:
... shellcode ...
end:
call code
.string "/bin/sh"
```

Subject: SSS
Date 98/8/19

- SEI CERT C Coding Standard
- PREfast Analysis Tool (MSDN) → حواله شد.

برخی از این Call ها در حالت عادی برای Push کردن (این ... string) و

Shellcode این از /bin/sh را دارد! Pop کرده و به آن

* همین مقاله یعنی Smashing the stack for Fun and Profit از Aleph One

برای Shellcode باید از offset های Call و jmp استفاده می‌شد.

Call و jmp آن را position independent می‌کرد و از همان دستورات

استفاده می‌کرد. ← روش‌های حافظه‌ی پشته را دستکاری می‌کرد و آدرس‌ها را relative می‌کرد.

نکته‌ی مهم در این است که نحوه‌ی وارد کردن داده در برنامه‌ها، چگونه

خودم بتوانم دستکاری کنم و کد اجرایی را تزریق کنم. Code مهم خودم بتوانم. مهم واقعی این‌ها

در اختیار دارد. یک مهاجم می‌تواند با این روش‌ها تغییر کند. تمام این‌ها در آدرس بازگشت

برای shellcode است.

Shellcode با privilege چه سطحی، caller نیز main دارد که

admin یا root است.

حال برای این مشکل چه راه‌حل‌هایی وجود دارد.

Subject:
Date ۹۵/۸/۱۳

راه‌کارها:

۵- از توابعی استفاده کنید که این boundary ها را رعایت می‌کنند، مثل safety

۴- از کامپایلرهای جدید استفاده کنید. (البته با مقاومت‌هایی که در ادامه مشاهده می‌کنید)

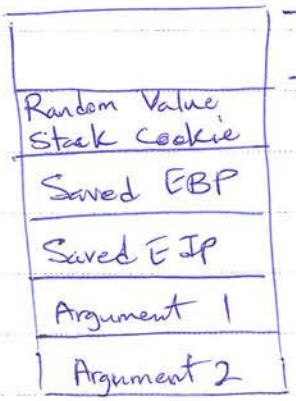
۱- Stack Cookies (Stack Canaries) (Compile-time)

قرار دادن یک مقدار رندوم بین local variable و return address

(۳۲ بیت یا ۶۴ بیت)

قرار دادن یک مقدار رندوم.

هنگام برگشت از تابع با بررسی این مقدار، صحت آن بررسی می‌شود ← integrity check



بعضی از کامپایلرها از null در انتهای cookie استفاده می‌کنند چون رشته‌ای مثل strcpy در

(Terminating Canaries) هنگام رسیدن به null متوقف می‌شود.

این راه‌حل جلوی Buffer Over Flow (نوشتن روی متغیرهای محلی و ...) را می‌گیرد اما جلوی

کپی شدن برنامه‌های مخرب روی Argument ها را نمی‌گیرد. این دسته از حملات را جلوگیری نمی‌کند.

همچنین SEH نیز دور زدن این روش را ممکن می‌کند.

Subject: SSS

Date 95/8/18

این راه‌حل، Compile-time است. چیزی از کرنل نداریم که LoApplication را re-compile کنیم.

۲- Non-Executable Stack : در این سیستم‌ها می‌توانیم کنترل کنیم که پشته قابلیت اجرا شود یا خیر.
(NX) (W⊕X)

write یا execute

فقط اجازه داریم یکی از این دو را داشته باشیم و دیگری را نداشته باشیم.

سیستم‌ها از سال ۲۰۰۴ به این طرف توانایی mark کردن صفحه‌های حافظه را دارند.

اطرافی بین این کش‌های حافظه مربوط به ثبات را برای اجرا mark می‌کنیم.

ROP: در این راه‌حل چیزی که قرار است bypass شود سیستم‌عامل است، مثلاً return

نقطه ضعف: این سیستم‌عامل‌ها معمولاً در CodeSegment استفاده می‌کنند. معمولاً توابع مربوط به کرنل را ندارند.

یا اینکه Sys call داریم، قطعه‌ای unprotected داریم. (مثل API های ویندوز)

۳- Address Space Layout Randomization (ASLR) : هدف این است که آدرس‌ها را به‌صورت تصادفی در حافظه قرار دهیم.

حافظه، جاهای مختلف می‌گذاریم و دائم آدرس متغیرها را عوض می‌کنیم (مثلاً آدرس base را عوض می‌کنیم) و دیگر حمله‌ای نمی‌توانند

(Security by obscurity : ایده) اینکه حمله نکنند، نمی‌دانیم چرا نمی‌کنند.

چون ایده، از یک اجرا به اجرای دیگر متفاوت است، نمی‌توانیم از آن استفاده کنیم.

Subject:
Date

کدهایی که این را انجام میدهند: به جای آدرس ثابت static همیشه ثابت است. مثل

PEB و بعد مثل Specialized data Procedures مثل Base Executable

linux در loader ، VSyscall

process ها است. از همین جا باهم کار این است که میدهد.

مثل brute-force است. هر چیزی را باید امتحان کرد تا درست شود. مثل

A&LR همه چیز ها تصادفی نیست مثلا 256 تا برای stack محدود است. طراحی هم برای کار شد

child به صورت process randomization طراحی شد.

این را درست کردند که بعد از ارجاع اصطکاک را از بین ببرد، واقعا همین کار را کرد.

۴- SEH : Safe از ویژگی های XP service pack 2 به بعد استفاده شده، برای

exception هایی که در حین اجرا برنامه از برنامه رد می شود، با جدول exception های کار می کند

جدول، به جدول می رود از آدرس handler که هست، بعد اجرا می کند.

دزدیدن جدول exception ها کار بسیار راحتی است برای کار بدی که بعدا می خواهد انجام شود.

۵- Function Pointer Obfuscation : اسم بازی است از جایی که می خواهد. مثلا با یک xor کردن

shell code ها، برای اسکن DoS و ... این کار را می کند Deobfuscate می کند. این کار را می کند

(Decrypt)

Subject: SSS
Date 95/8/18

- مقاله ASLR
- مقاله Catal Flow Integrity

- مقاله: Aleph One

Code Segment

Program Building Blocks:

(The Art of Security Assessment
Chapter 7 (7.3)

- Variable Use:

- Understanding variables

- Their relationship to each other

- How an application can be affected adversely by unexpected manipulation of these relationships.

- Variable Relationships

Control Flow Graph, Data Flow Graph

Subject:
Date

در داده مهم در تغییر داده یا تغییر State برنامه اهمیت می‌یابد.

مثال) متغیری که قبل از شرط استفاده در حلقه طول بقیه حلقه ثابت است.

از متغیرها با تغییر در ارتباط نباشد، با هم در lockstep نباشند.

این دو متغیر ها ارتباط نباشد و de Synchronized باشد، از نظر ایمنی مشکل دارد.

Wrong Judgment ↵

قرارداد از نظر Syntax برای این ها را حفظ کنیم. چون داریم در این مورد بیش از اندازه فکر می‌کنیم.

مثال) mod_dav Apache module
↓
distributed authoring and versioning

CDATA XML elements → عناصری که داده دارند
Character Data → داده‌ای که markup نیست

```
cdata = s = apr_palloc(pool, len+1);
for (scan = elem -> first_cdata.first; scan != NULL; scan = scan->next)
{
    tlen = strlen(scan->text);
    memcopy(s, scan->text, tlen);
    s += tlen;
}
for (child = elem -> first_child; child != NULL; child = child->next)
{
    for (scan = child -> following_cdata.first; scan != NULL; scan = scan->next)
    {
```

Subject: 885
Date
۹۵/۸/۲۲

```
tlen = strlen (scan->text);
memcpy (s, scan->text, tlen);
s += tlen;
}
}
*s = '\0';
```

قبل از اول بار (len) موقعیت مناسب را بالا هم مرتب نگه داریم. اول کار برسیم به آخر

بار از trim می‌کنند. یعنی whitespace ها را حذف می‌کنند.
(پایانی)

```
if (strip_white){
    while(apr_isspace(*cdata))
        ++cdata;
    while (len > 0 && apr_isspace(cdata[len]))
        continue;
    cdata[len+1] = '\0';
}
return cdata;
}
```

cdata , len ... مرتب

len نمی‌شود !! , cdata اولش

این دستور desynch است.

Subject:
Date

```
char *trimwhitespace(char *str)
{
    char *end;
    while (isspace((unsigned char) *str)) str++;
    if (*str == 0) return str;
    end = str + strlen(str) - 1;

    while (end > str && isspace((unsigned char) *end)) end--;
    *(end + 1) = 0;
    return str;
}
```

Tip: Determine what each variable in the definition means and how each variable relates to the others. After you understand the relationships, check the member functions or interface functions to determine wether inconsistencies could occur in identify variable relationships. To do this, identify code paths in which one variable is updated and the other one is not.

- Variable Initialization :

(Initialize_before_use)

Subject: SSS
Date
۹۵/۸/۲۴

```
int login (char *login_string){

    char *user, *style, *ptr;
    ptr = strchr(login_string, ':');
    if (ptr) {
        *ptr = '\0';
        user = strdup(login_string);
        style = strdup(ptr+1);
        *ptr = ':';
    }
    else
        user = strdup(login_string);

    ≡

    if (style) {
        ≡
    }
```

style نوع authentication است.
مثلا: biometric

این style در صورتی است else (یعنی کاربر : ندارد در login) متغیر user اولیه

شده است !!

Subject: SSS
Date 98/A/11

(تمرین ۱۲ از ۱۲ تمرین — تمرین ۷ — تمرین ۳ — تمرین ۲ — تمرین ۱ — Prefast — تمرین ۱ از ۱۲)

جلسه گذشته در مورد building blocks (۱) صحبت کردیم که شامل Variable Use (۲)، Control Flow (۳) و Functions (۴) بود.

The Art of Software Security Assesment — مرجع اصلی درس — Memory Management (۴)

کتاب تمرینات.

مثال: Constructor، Object oriented، initialization.

```
class netobj {
private:
    char *data;
    size_t datalen;
public:
    netobj() {datalen=0;}  ~~> Constructor
    ~netobj() {free(data);}  ~~> Deconstructor
    getdata() {return data;}
    int setdata { ... memcpy(data, din); ... }
}
=
int get_object(int fd) {
    char buf[1024];
    netobj obj;  ⟵ تعریف شیء obj از کلاس netobj
    int n;
    if ((n = read(fd, buf, sizeof(buf))) < 0) return -1;
    obj.setdata(buf, n);
    =
    return 0;
}
```

در خط آخر return، obj از بین می‌رود و باید data را deallocate کند. پس باید از طریق تابع free این کار انجام شود.

Subject: SSS
Date 90/8/20

این تیپ برای Auditor و Developer است.

Tip: در انتهای هر تابع از clean-up Prolog استفاده کنید.

صورت تابع که منابعی را آزاد می‌کند یا کنترل خروجی دارد باید از goto استفاده کند.

- Arithmetic Boundaries:

خطاهای مربوط به overflow و underflow در محاسبات عدد صحیح اتفاق می‌افتد.

برای کشف این خطاها از روش زیر استفاده کنید:

1- Discover operation that, if a boundary condition could be triggered, would have security-related consequences.
(Problem Area)
حوزه مسئله

2- Determine a set of values for each operand that trigger the relevant arithmetic boundary wrap. (Problem Domain)

3- Determine wether this code path can be reached with values within the set determined in step 2.
(Reachable code path) (Validated Domain)

برای اولین عملوند عملیات (size allocation) و Copy و ... آنالیز انجام دهید.

برای هر عملوند باید یک boundary wrap ایجاد کنید.

Subject: 32-bit unsigned value
Date

if (length + 32 > sizeof(buffer))

0xFFFFFFFF ، 0xFFFFFFE0 ... length ...

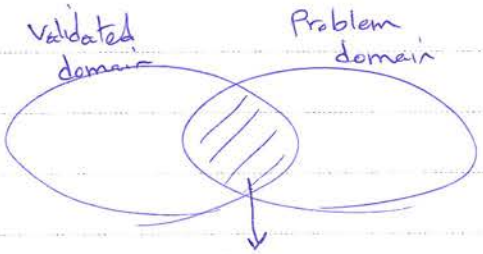
... no error ...

Problem domain

if (length1 + length2 > sizeof(buffer))

... Problem domain ...

Symbolic Execution ... SAT Solver ...



if the intersection is not empty, so be careful!

- Identify the data type of the variable involved

e.g. Problem domain : 0xFFFFFFE0 - 0xFFFFFFFF

if 16-bit unsigned integer ⟶ It is safe.

Subject: 885
Date
۹۵/۸/۲۶

size_t نباید باشد به عنوان نمونه متغیری که از ورودی می‌آید. (؟)

- Determine the points at which the variable is assigned value.

از هیچ یک از متغیرها و ورودی‌ها استفاده نشده است. Problem domain

- Determine the constraints on the variable from assignment until the vulnerable operation.

- Determine supporting code path constraints.

sizeof(buf) مثال

```
#define BLOB_MAX 1024                                    (مثال
unsigned char *readblob(unsigned char *blob, size_t pktlen){
    int bloblen;
    unsigned char *buffer;
    bloblen = ntohl(blob);
              ↓
    network_to_host → تبدیل داده‌ها از ترتیب شبکه
```

```
    if (bloblen + sizeof(long) > pktlen ||
                                          or
                bloblen > BLOB_MAX)        → Step 2
        return NULL;

    buffer = alloc(bloblen);                → Step 1
    if (!buffer) return NULL;
    memcpy(buffer, blob+4, bloblen);
    return buffer;
```

PAPCO

Subject:
Date

حمله در اغلب شرایط وقتی رخ میدهد که برنامه نویس فرض میکند که شرایط خاصی برقرار است و این فرض را در کد خود لحاظ میکند.

برنامه نویس باید به این شرایط توجه کند و از bypass شدن آنها جلوگیری کند.

حل این نوع مشکل نیز بستگی به Problem domain دارد.

در اغلب موارد، unsigned بهتر از signed است و blob len باید از نوع size of (long) باشد.

Problem domain:

0xFFFFFFFC - 0xFFFFFFFF

(Signed: -4 to -1)

حال اگر mode برابر (Signed) -4 و size of (long) برابر 4 باشد، شرط ما برقرار خواهد بود.

اگر شرط ما برقرار نباشد، Problem domain ما برقرار نیست و return NULL میشود.

```
int process_packet (unsigned char *pkt, size_t pkt_len) {

    unsigned int length = 0;
    int type = 0;
    unsigned char * data;
    type = pkt[0];
```

PAPCO

Subject: SSS
Date: ۹۵/۸/۲۵

```
switch (type) {
    case TYPE_KEY:
        length = ntohl (&pkt[1]);
        if (length != RSA_KEY_SIZE) return -1;
        data = read_blob (&pkt[1], pktlen);
    ...
    case TYPE_USER:
        data = read_blob (&pkt[1], pktlen);
    ...
    default:
        return -1;
```

← این قسمت در محدوده Problem domain نیست. چون مقایسه حذف کرده است

برابر با ۱۰۲۴ یا ۲۰۴۸ بیت.

این قسمت می‌تواند هر مقداری باشد. پس

validated domain اشتراکی با Problem domain ندارد.

⇓

آسیب‌پذیری اینجا هست پس!

مثلاً ۴ در validated domain هست؟

معیارهای supporting چیه؟

به ازای type = TYPE_USER برابر با ۶ مقدار pktlen و بیشتر

&pkt[1] طوری باشد که آسیب‌پذیری exploit شود.

Program Analysis
Nielsen

Subject:
Date

Prefast → Annotation
Inference tool to make annotations automatically
{ Walk Through Analysis / Code Analysis
- Visual studio , C/C++ }

- Type Confusion → to type عوض کردن

- Lists and Tables →
(linked list)

Control Flow : (Auditing Control Flow)

Data Flow , Control Flow : انواع flow در برنامه ها مطرح میشود
جریان داده — جریان کنترل

با بررسی کدهای برنامه هرکدام از این جریان ها مشخص میشوند.

در زبان های imperative : جریان های کنترل با دستوراتی که نوشته میشود، مشخص میشوند.

برنامه ها استفاده از goto میکنند، unstructured هستند.

Control Flow آنالیز یا بررسی کدهایی که یک برنامه برای اجرا دنبال میکند.

تعدادی از construct های زبان مشخص کننده مسیر برنامه برای اجرا هستند: مثلا از Subroutine ها

goto, loop, while, if, switch, ... → Control Structures

- Looping Constructs:
از مرسوم ترین ساختارها هستند. برای انجام کارهای تکراری

تکراری ← خواندن و نوشتن چند از مجموعه داده ها (فعلی تمام شده و منتظر داده بعدی است.)

PqPCO

Date : 885
Subject : ۹۵/۹/۲

Page : ( )

* طراحی این نرم افزار باید به صورت Safe انجام شود.

- The terminating conditions do not account for destination buffer sizes or do not correctly account for destination sizes.

شرط‌های خاتمه

- The loop is post-test when it should be pre-test.

- 'break' or 'continue' statement is missing or incorrectly placed.

- ~~misp~~ Some misplaced punctuation ...

* هر یک از این‌ها را می‌توان به صورت Information Flaw در نظر گرفت.

* شرط‌های خاتمه برای حلقه‌ها:

- طراحی صحیح این حلقه‌ها باعث جلوگیری از ایجاد ارور می‌شود.

- تعاریف برای هندل کردن خطاها.

- این شرط‌ها را به صورت Boundary Check در نظر بگیرید.

مثال) Network Time Protocol این پروتکل از چه نوع خطاهایی جلوگیری می‌کند؟

اما باید توجه کرد که این خطاها مثل off-by-one error

Page : (/)

Date : ۹۵/۲/۲
Subject : نسبت سنجی های کاربردی

داده های مربوط به متد POST سرور ارسال و کاهش می‌یابد.

Apache mod_php (مثل

non-terminating buffer vulnerability

در کنار این کد، شرطی هم بگذارید و خطا بگیرید error ای که آسان می‌شود از حمله جلوگیری کنید.

```
if ( SG(read_post_bytes) > SG(post_max_size)) {
    error...
}
return;
```

در کنار هایی که در انتهای بافر null قرار می‌دهند terminating صورت می‌گیرد که از سرریز شدن بافر جلوگیری می‌کند.

- Posttest vs. Pretest Loops:

do-while

for, while

انتهای حلقه بررسی می‌شود

حلقه حداقل یک بار اجرا می‌شود.

```
char* cp = get_user_data();
do {
    ++cp;
} while(*cp && *cp != ',');
```

(مثل

اگر null باشد، خارج می‌شود.

اگر null نباشد، cp یک واحد جلو می‌رود

اگر null نباشد.

Date : 555
Subject : ۹۵/۹/۲

Page : ( )

- هر حال باید وقت داشت تابع باید body مناسب را اجرا کند و pretest و posttest را
استفاده کرد.
- از initialization بعد از pretest و بعد از error و return
- پس هیچ وقت از خطا خارج نشد، تمام راه‌های ورودی باید posttest استفاده کنیم.

- Flow Transfer Statements :

'break' ⟵ از حلقه داخلی می‌آید بیرون

همان است که باید در یک حلقه‌ای از همه دستورات این حلقه‌ها بیاید تا این باعث از کار افتادن کد.

'Continue' ⟵ از این iteration رد شو، برو بعدی

'Switch-case' ⟵ باید از هر case یک break بگذاریم، یعنی حالت‌های case کم.

Auditing Functions :

- function‌ها باید وظایف خود را به درستی انجام دهند. اصل Abstraction در برنامه‌نویسی از همه مهم‌تر است.
Universal Typing از این تعریف بالاتر است. (؟)
- تمام کار developer، تمام این auditor خوب است.

Page : ( )

Date :
Subject :

- Return values are misinterpreted or ignored.
- Arguments supplied are incorrectly formatted in some way.
- Arguments get updated in an unexpected fashion.
- Some unexpected global program state change occurs because of the function call.

Function audit logs :

- function name (prototype) (signature)
- Description
- Location (line of code which)
- Cross-reference

Date : 885
Subject : ۹۵/۹/۲

Page : ( )

- Return_Value type

- Return_Value meaning → مثلا اگه از تابع خطا دریافت، معنی خطا چیست. (مثل int main)

- Error Conditions

- Erroneous return values

باید تمام اطلاعات لازم از عملکرد تابع در return value موجود باشد. نیاز به خواندن کدهای تابع نباشد و تابع black box باشد!

مثال)

```
char *buf = (char *) malloc(len);
memcpy(buf, src, len);
```

اگر malloc خروجی NULL داد، buf مقدار نامعتبر خواهد گرفت!

اگر تابع void باشد، یعنی هیچ خطایی ندارد (چون تابع void) و نیاز به کنترل خطا بعد از فراخوانی تابع نیست.

موفقیت یا شکست آن از فراخوانی تابع چک میشود.

Page : ( )
Date
Subject :

```
int read_data (int sockfd, char **buffer, int *length){

    char *data;
    int n, size = MAX_SIZE;
    if ( !(data = (char *) calloc (MAX_SIZE, sizeof (char))))
        return 1;
    if ( (n = read (sockfd, data, size)) <=0)
        return 1;

    *length = n;
    *buffer = data;
    return 0;
}

int process_request (int sockfd) {
    char *request;
    int len, reqtype;
    read_data (sockfd, &request, &len);
    reqtype = get_token (request, len);
    =
```

(Secure Programming with Static Analysis)

Static Analysis :

Any tool that analyzed code without executing it is performing static analysis.

(Semantic analyzer in Compiler → بخشی از کامپایلر که Context-Free Grammer ها را بررسی می‌کند)

هر جزیی که در Compile_time کامپایلر بود، کامپایلر آنها را بررسی می‌کند.

- Dynamic Analysis → در زمانی که برنامه در حال اجرا می‌شود
- Runtime Analysis → در زمان اجرای واقعی، به بررسی برنامه می‌پردازیم.
- (Runtime Verification)

Program code بهترین Specification است. چون تمام اجزای کامل برنامه را در اختیار داریم.

یک کد، مدلی است از هم اتفاقات در زمان اجرا، پس همه آن کامپایلر وضعیت می‌شود.

یکی از کاربردهای Static Analysis ها Spell checker ها است:

۱) همه اشتباهات را نمی‌گیرد. مثل mute ، most ← کامپایلر هم همین خطاها را نمی‌گیرد.

۲) برای Speller ها که نرم‌افزار است ولی برای کسی که زبان خیلی خوب است، همه واژه‌ها را خوب می‌بیند.

پس کامپایلر برای کد برنامه‌نویس خوب کار می‌کند.

۳) کامپایلر اشتباه کد که از اسمبلی تبدیل شده است، نمی‌گیرد.

ویژگی‌های کامپایلر:

- Static analysis tools check thoroughly and consistently.

کد با احتیاط، بدون bias، همه چیز را چک می‌کند.

- By examining the code itself, static analysis tools can often point to the root cause of a security problem, not just one of its symptoms.

Auditor کامپایلر عکس وقت‌ها از اشتباه‌ها را نمی‌بیند. مثلاً اگر A در کد باشد، برنامه خراب می‌شود. یک راه حل این است که هیچ‌گاه در کد A نباشد! این روش حل مشکل BOP است.

- static analysis tools can find errors early in development, even before the program is run for the first time.

مثال) خودروی در میلیون‌ها خط در زمان اجرا می‌کند؛ که همه‌هم کلیل است طول بکشد، خطا دارد.

وظیفه کلیل است صحت‌سنجی از تجربیات و دانش قبلی‌هم داریم استفاده می‌کنیم.

- روی‌هم‌رفته کلیل است برای پیدا کردن خطاهای ساده و پرتکرار خوب است، مثل Enforceable Security Policies، ...

- شکل اصلی کلیل است ← ایجاد noise

False Positives ← نتایج مثبت کاذب، خطاهایی که واقعاً خطا نیستند
False Alarms

- اگر FP ها خیلی زیاد باشند، ممکن است auditor تمام خروجی‌ها را نادیده بگیرد.

- چون FP مزاحم است، False Negative را ترجیح می‌دهیم؛ یعنی ترجیح می‌دهیم از نظر امنیتی مطمئن نباشیم. چون auditor حس خوبی دارد.

ابزار کلیل است باید توازنی بین این دو برقرار کند.

سخت‌گیری زیاد و محافظه‌کارانه
برای اینکه FN نداشته باشد، کلی FP برمی‌گرداند.

اگر کمتر باشد (غیرامنی)، بعضی از FN ها را نادیده می‌گیرد.

تکنیک‌های مختلفی برای کلیل است وجود دارد:

- Type Checking
- Style checking
- Program Understanding
- Program Verification
- Property checking
- Bug Finding
- Security Review

روش‌ها هرکدام از ویژگی‌های قبلی صحبت خواهیم کرد.

(روش‌ها مستقل از هم نیستند، مثلاً خروجی یکی از این‌ها می‌تواند ورودی دیگری باشد)

Type Checking:

Semantic analyzer

(Types and Programming Languages, Pierce)

(Type Theory)

Java Program:

```
short S = 0;
int i = S;
short r = i;
```

```
Object[] objs = new string[1];
objs[0] = new Object();
```

ArrayStoreException

FN

False Negative

Security Type Checker

Casting

Type Safety

typable

get stuck

Type Checking

Style Checking:

Type Checker

rule

gcc

gcc -wall

(Warn All)

```
typedef enum {red, green, blue} color;
char *getColorString (Color c) {
    char *ret = NULL;
    switch (c) {
        case red :
            Printf("red");
    }
    return ret;
}
```

اگر gcc -Wall را بزنیم، می‌گوید برای Case مربوط به blue و green چیزی نیست!

چون برنامه‌ها style های مختلف دارند، پس محور flexible است.

ابزار Lint یک StyleChecker معروف است. ابزار PMD برای جاوا است.

ابزار ParaSoft برای Java, C/C++ یک style checker, bug finder است.

Program Understanding:

کد برنامه (Code Base) را می‌گوید. مثلاً IDE جاوا این را دارد.

مثلاً این تابع در کجاها استفاده شده، تعریف شده و ... ← در IDE خیلی راحت می‌شود.

ممکن است که بخواهیم Reverse Engineering کنیم ← در آن از روی کد برنامه، دیاگرام‌های UML مثل Class diagram را می‌سازد.

Design Pattern های استفاده شده را می‌دهد.

ابزار Fujaba می‌تواند از روی Code، UML Diagram بسازد.

ابزار CAST Systems

Program Verification and Property Checking:

اثبات اینکه برنامه با Specification طراحی شده مطابقت دارد یا نه.

equivalence checking برای Specification: UML، Z، جاوا و ... را می‌گوید.

Specification داده شده، برنامه را تولید می‌کند.

مثلاً در طراحی مدارها و چیپ‌ها، استفاده از این روش خیلی مرسوم است.

555
95.9.7

Specification چگونه می‌توان صحت یک برنامه را نسبت به یک

Equivalence Checking برای برنامه‌های بزرگ عملی نیست.

Partial Specification Property Checking Equivalence

Temporal Safety Properties

Individual Executions

Define Bad Prefixes → irremediable

Property Checker FP FN

Counterexample

```
1: inBuf = (char *) malloc(bufSz);
2: if (inBuf == NULL)
3:     return -1;
4: outBuf = (char *) malloc(bufSz);
5: if (outBuf == NULL)
6:     return -1;
```

Property:
"allocate memory should always be freed"

line 2: inBuf != NULL
line 5: outBuf == NULL
line 6: function returns (-1) without freeing inBuf

Sound with respect to the specification

if it will always report a problem if one exists.

FP FN

SSS
۹۵،۹،۲

ابزارهای Property Checker معمولاً برای یک Property خاص طراحی شده‌اند و FP کمتری دارند.

روش کار قبلی، خواندن کد برای پیدا کردن باگ‌ها بود:

line 2: inBuf == NULL
line 3: function return(-1) without freeing inBuf

پس این ابزار جزء Sound است ولی False Positive دارد.

Soundness یعنی هیچ False Negative وجود ندارد؛ یعنی برنامه سالم است و ابزار نباید هشدار دهد، اما ممکن است هشدار دهد (یعنی FP دارد) (Complete است)

اثبات Soundness یکی از طراحی‌های این ابزارهاست.

دو ابزار Property Checker وجود دارد: PolySpace, Grammatech (این‌ها تجاری هستند.)

Bug Finding:

هدف این دسته ابزارها Formatting نیست (این کار Style Checker است) و به دنبال پیدا کردن باگ‌ها هستند.

برای همین سرعت بیشتری نسبت به Property Checker دارند.

مجموعه‌ای از bug idiom ها تعریف شده است.

به دنبال پیدا کردن باگ‌هایی از این نوع هستند و سعی می‌کنند آن‌ها را پیدا کنند.

مثال‌هایی از آن‌ها را در ادامه می‌بینیم.

* FindBugs یکی از آن‌هاست.

↳ مثال:

```
if (this.fitz == null) {
    synchronized (this) {
        if (this.fitz == null) {
            this.fitz = new fitzer();
        }
    }
}
```

Double-checked locking

(؟) ↳ Google it!

نکته: در جاوا دستور synchronized وجود دارد.

double-check locking bug

۹۵۵، ۹۵،۹،۹

بعضی از bug Finder ها از الگوریتم‌های Property checker استفاده می‌کنند. اما هدف آن‌ها این است که FP کم شود حتی اگر FN زیاد شود.

چون خطاها را در بهینه‌ترین حالت پیدا می‌کنند.

ایده‌آل آن است که:

An ideal bug Finder is sound with respect to a counterexample, ...

به این معنا که هر آنچه گزارش می‌دهد، مثال نقضی است که واقعاً اتفاق می‌افتد. این را Completeness می‌گویند.

(feasible scenario)

یعنی از bug Finder هرچه گزارش شد، حتماً باید رفع شود. (کم کردن FP هم مهم است اما اولویت با کم کردن FN است)

چون دیدن برای خطاهایی که عمدتاً از FN ها هستند، از Bug Finder, Property checker استفاده می‌شود.

ابزارهایی مثل Coverity (برای C/C++) و حتی analyze در MS Visual Studio و همچنین PreFast (برای C/C++) و از Klocwork از جمله bug Finder ها هستند.

Security Reviews:

ابزارهای مذکور احتمالاً برای امنیت کاربرد دارند یا این‌ها از قابلیت‌های داخلی ابزارهای دیگر استفاده می‌کنند. مثلاً ابزارهای FN زیاد دارند. ابزارهای اولیه و ابتدایی برای امنیت مثل ITS4، RATS، FlawFinder هستند. همگی از "grep" استفاده می‌کنند. حال روی این لیست بعضی از توابع خطرناک را در برنامه پیدا می‌کنند. (مثلاً strcpy) پس تعداد زیادی FP دارند. اما ابزارهایی که الان هستند ترکیبی از Property checker و Bug Finder ها هستند.

بسیاری از Security Property ها را می‌توان در قالب Property بیان کرد. مثلاً BOF:

"the program does not access an address outside the bounds of allocated memory."

پس از برنامه‌ای خواهیم که اگر دارد (یعنی خودش خطا دارد)، احتمال Buffer Over Flow در آن برنامه تا چه حد exploit است.

(برای توضیحات از ابزارهای مذکور به cert.aut.ac.ir/formalSecurity)

- سطح بالاتری از abstraction در این ابزارها وجود دارد (intensional ، extensional)

بعضی شخصا روی یک مسئله خاص و با Property مشخص روی برنامه کار می‌کنند مثل race condition ، information flow.

- این ابزارهای امنیتی در واقع یک نوع reviewer هستند و بعضی بار کار reviewer را سبک‌تر می‌کنند.

- یکی از Fortify Software (بعدا بیشتر در مورد این)

A little Theory, A little Reality:

It is a fact that static analysis is a computationally undecidable problem.

این به این معناست که نمی‌توان برنامه‌ای نوشت که in general این مسئله را کامل انجام دهد و به این معناست که نمی‌توان با کامپایل کردن خروجی‌ها با رفلکس کردن شرایط FP, FN برنامه‌ای ساخت که همه برنامه‌ها را کامل انجام دهد.

(البته محدودیت سرعت و سرعت نور و ... هم هست!)

Rice's Theorem: هر خاصیت نحوه برنامه‌ها که بر روی آن تصمیم‌گیری می‌کند مسئله کامل است ← تعریف صوری دارد.

ثابت کردن هر کدام از این دو، مثل halting problem ... جزو سری‌هاست.

(diagonalization ایده)

فرض کنیم تابعی داریم is_safe() که برنامه می‌گیرد و می‌گوید این خاصیت را دارد یا نه.

(در این صورت تابع ما یک مسئله undecidable را حل می‌کند.)

حال چنین برنامه‌ای را می‌سازیم:

```
bother (function f)
    if (is_safe(f)){
        call unsafe();
    }

b = bother;
bother(b);
```

اگر این برود و کار نادرستی بکند ← اگر f ، این را کار نادرست انجام می‌دهد.

اگر این نرود و شرط تمام شود و کار را انجام دهد ← اگر f ، این را کار نادرست انجام نمی‌دهد.

Can you decide "bother"

888,98/9/9

Success Criteria: (پارامترهای موفقیت ابزار کلیل است)

- The ability of the tool to make sense of the program being analyzed.

درک درستی از برنامه‌ای که دارد کلیل می‌کند، داشته باشد.

- The trade-offs the tool makes between precision and scalability.

- The set of errors that the tool checks for.

- The length to which the tool's creators go in order to make the tool easy to use.

کلیک کنید MS Visual Studio در Prefast : Chapter 7 ← ابزار تحلیل ایستای آزمایشی

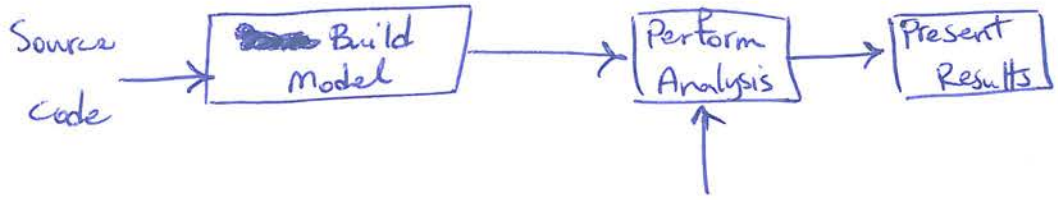
analyze

۹۵،۹،۲۸ تحلیل کردن : نسخه دوم

مسیرهای Walk Through دارد در ریپازیتوری

۹۵،۹،۱۴ SSS

ساختار داخلی ابزارهای تحلیل ایستا :



در Build Model ، در اولین گام ، همراه با تحلیل ایستا این کار انجام می‌شود ، از سورس کد ، یک Program Model ایجاد می‌شود که تحلیل‌ها روی آن انجام می‌شود. این مدل بستگی دارد به نوع تحلیلی که می‌خواهد انجام شود ، یا به پلتفرمی که هدف است. ممکن است چند مدل هم ساخته شود.

The Dragon Book

Tiger Books (Apple's Modern Compiler Implementation Series)

اولین گام در این بخش تحلیل Lexical است.

Syntax = Lexical + Phrase structure

tokenize کردن ورودی

کنار هم گذاشتن tokenها برای ساختن جمله

برنامه را به صورت یک جریان از کاراکترها می‌بیند

tokenها را از Stream می‌خواند و ایجاد می‌کند

Example — `if (ret) // Probably true`
`mat[x][y] = END_VAL;`

Lexical Analyzer Result : IF_KW LPAREN ID(ret) RPAREN ID(mat) LBRACKET ID(x) RBRACKET LBRACKET ID(y) RBRACKET EQUAL ID(END_VAL) SEMICOLON

در ابتدا Lexical را با Reg Ex پیاده می‌کنیم ، بعد به سمت Parser می‌رویم :

90/9/14 - SSS

RegEX	Token
if	IF_KW
(	LPAREN
)	RPAREN
⋮	⋮
/[\t\n]+/	ignore whitespace
/\/\/.*/	ignore Comment
/[a-zA-Z][a-zA-Z0-9]*/	ID

یه ابزار کامل آنالیز زبان می‌تونه کار بالا رو تمیز بزنه.
مثل RATS, ITS4, ... که حتی خطاهای رایج مثل strcpy رو هم پیدا می‌کنن.
کد خطا نیز داده شود.

Parsing: گرامری که توکن‌ها رو به Sentence تبدیل می‌کنه.

A context_free grammar از موارد زیر تشکیل شده:
nonterminal, terminal, Production rules

Parse Tree: درخت تجزیه

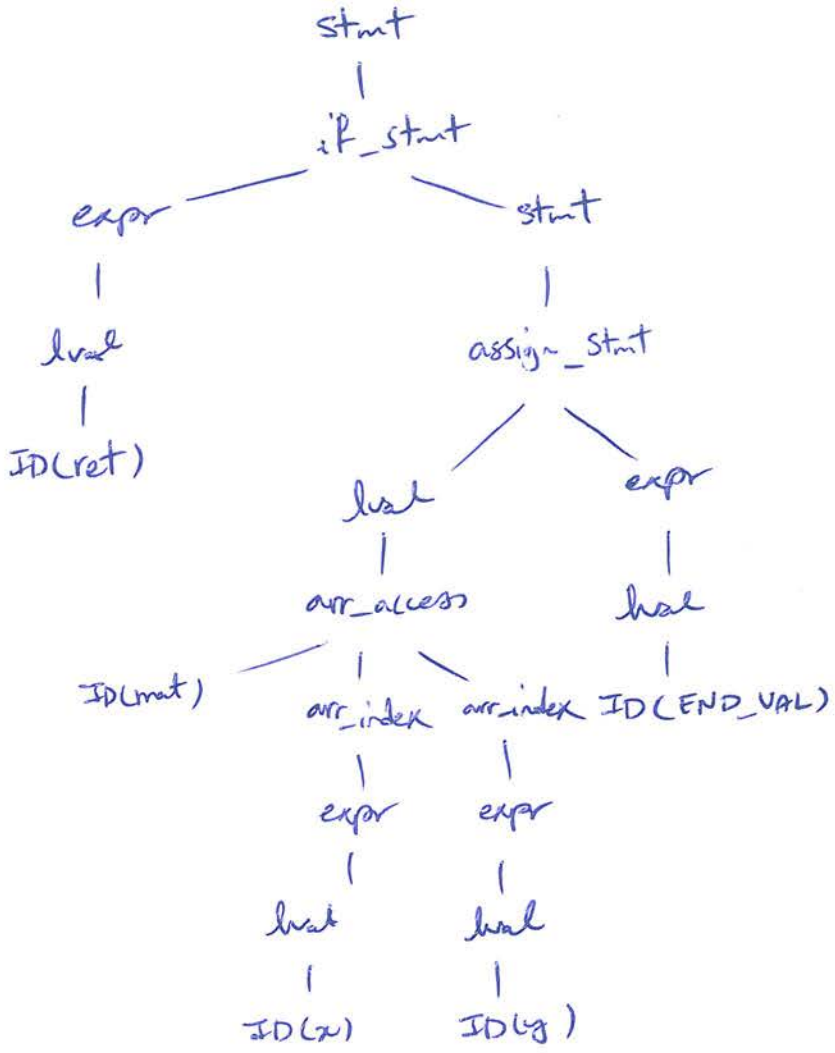
تعریف بازگشتی، context_free, recursive

Example -

```
stmt ::= if_stmt | assign_stmt
if_stmt ::= IF_KW LPAREN expr RPAREN stmt
expr ::= lval
assign_stmt ::= lval EQUAL expr SEMICOLON
lval ::= ID | arr_access
arr_access ::= ID arr_index+
arr_index ::= LBRACKET expr RBRACKET
```

برای برنامه‌نویسی باید derivation رو بلد باشیم تا parse رو پیاده‌سازی کنیم.
مثال:

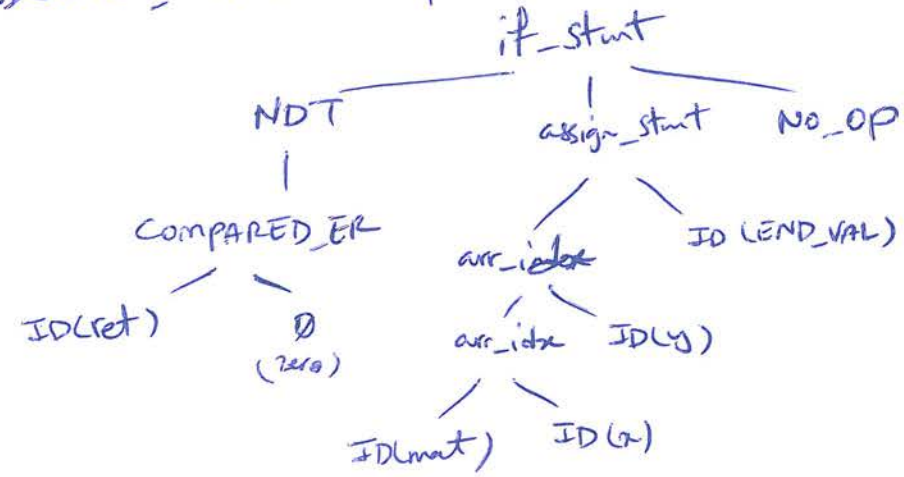
(صحیح) (با در نظر گرفتن تقدم عملگرها)



حالا اطلاعات بیشتری نسبت به قبل داریم. مثلا میدانیم که ret یک متغیر است و mat یک آرایه دو بعدی است و ... اما هنوز اطلاعات زیادی وجود دارد که در این درخت وجود دارد و ما به آن نیاز نداریم. مثلا ما نیازی نداریم که بدانیم ret یک متغیر است یا mat یک آرایه است. ما فقط نیاز داریم که بدانیم ret و mat و ... چه هستند و چه نوع داده‌ای دارند. به همین دلیل به این درخت، درخت تجزیه مشخص (Concrete Parse Tree) می‌گویند. به این دلیل که تمام جزئیات نحوی زبان در آن وجود دارد و هیچ چیز از آن حذف نشده است.

از معروف‌ترین ابزارهای کمکی برای نوشتن تحلیل‌گرهای نحوی و کوی، lex و yacc هستند. البته javacc نیز هست، ...

این درختی که به دست آمد، جزئیات زیادی دارد و ممکن است بسیاری از آن اطلاعات redundant باشند. پس به دنبال راهی هستیم که این اطلاعات اضافی را حذف کنیم. به این درخت، درخت نحو انتزاعی (Abstract Syntax Tree (AST)) می‌گویند. یعنی مثلا دیگر نوع ret در آن ذکر نمی‌شود:



تمام AST برنامه‌هایی که از نظر ساختار زبان مثلا در زبان‌های مختلف شبیه هم هستند، شبیه هم خواهد بود. البته AST هر برنامه منحصر به فرد است.

یا حتی در یک زبان

۹۵/۹/۱۴ ۵۵۵

Semantic Analysis است.

Symbol Table

Context_Free

Context_Sensitive

Context_Constraint

declaration

Pointer

Type checking

Security Type checking

Constraint checking

type

optimization

Tracking Control Flow:

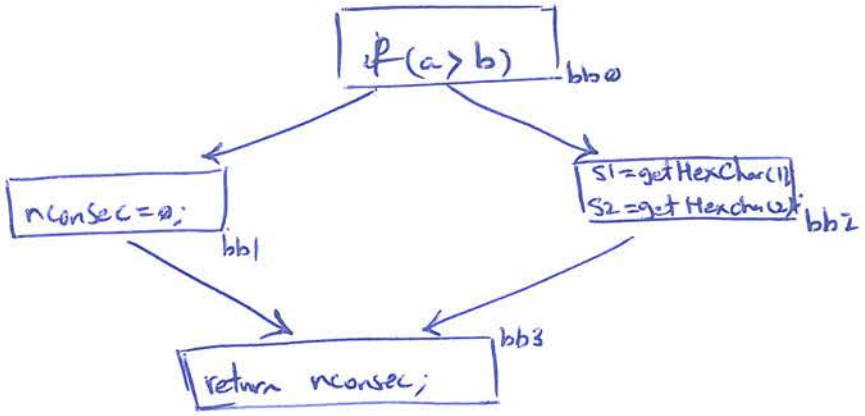
Control Flow Graph (CFG)

basic block

CFG

```
if (a>b) {
    nconsec = 0;
}
else
{
    s1 = getHexChar(1);
    s2 = getHexChar(2);
}
return nconsec;
```

Example —



در ابزارهای کامپایلر، از روی AST، CFG را برای برنامه تولید می‌کند.

مسیرهای مختلف در CFG، یک trace است. در شکل بالا دو trace داریم. هر trace تعدادی بلوک از برنامه است.

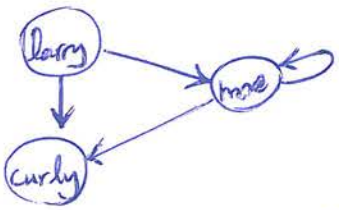
گراف زیر هم برای توابع و متدها استفاده می‌شود، Call Graph.

Call Graph:

Nodes: functions

Edges: potential calls of other functions

یعنی فانکشن‌های کدام تابع را در زمان فراخوانی ممکن است صدا بزند، از آن تابع به تابع‌های فراخوانی شده، یال داریم.



هر تابع locally در نظر گرفته می‌شود، سپس ترکیب کدهای هر تابع با فراخوانی Call آن‌ها.

در Function Pointers و Dynamic Dispatch (Virtual method) داشتن تشخیص فراخوانی خیلی سخت می‌شود حتی غیرممکن!

در virtual method و dynamic loaded code داشتن تشخیص کدام اسمبلی باید اجرا شود غیرممکن است.

کدهای دیگر به صورت سورس، کدهای جزء داده است.

Data Flow Analysis:

* فصل اول کتاب (dragon book) Aho — Program analysis خواندنی است.

کدهای جزء داده یعنی بررسی اطلاعاتی که داده می‌توانند حرکت کنند و جزء داده است.

آیا داده‌ای که از کاربر دریافت می‌شود (ورودی برنامه) به درستی بررسی می‌شود یا نه؟ این نوع آسیب‌پذیری‌ها را بررسی می‌کند.

ساده‌ترین حالت، assignment است و guard های شرط و تکرار نیز جزئی هستند که بیشتر برای ورودی‌های برای بررسی می‌شوند.

SSS - 95/9/21

Tracking Data Flow :

کنترل جریان داده

Dataflow analysis algorithms examine the way data move through a program.

Optimization ← برای بهینه‌سازی کامپایلرها برای اجرای دستورات، حذف dead code ها و ... از این نوع آنالیز استفاده می‌شود.

Def - Use یا Syntactic یعنی کامپایلر به دنبال نحو خاصی است و کنترل جریان داده، برای کامپایلر مهم است.

Semantic ← یعنی به دنبال معنای کد می‌گردد. (تعریف - استفاده Define - Use) یعنی کنترل جریان داده از نظر معنایی

Notion Generate - Use

flow واقعی یعنی هر دستور به کجا می‌رود و از کجا آمده است.

non interference ← تغییر یک متغیر روی متغیر دیگر تاثیر نگذارد. (شرط‌هایی که باید رعایت شود)

در کنترل جریان داده، برنامه‌ها از static Single Assignment (SSA) پیروی می‌کنند. یعنی Value یک متغیر فقط یک بار مقدار می‌گیرد.

to assign a value to a variable only once.

استاندارد این است که برنامه SSA - form باشد و همه مقادیر کنترل جریان داده feasible باشد.

Example - Tiny Encryption Algorithm (TEA)

```
Sum = Sum + delta;
Sum = Sum & top;
y = y + (z << 4) + k[0] ^ z + Sum ^ (z >> 5) + k[1];
        ↳ shift left
y = y & top;
z = z + (y << 4) + k[2] ^ y + Sum ^ (y >> 5) + k[3];
z = z & top;
```

برای SSA باید فرم index گذاری را به صورت زیر در آوریم:

$Sum_2 = Sum_1 + delta_1$;

$Sum_3 = Sum_2 \;\&\; top_1$;

$y_2 = y_1 + (z_1 << 4) + k_0[0]_1 \;\hat{}\; z_1 + Sum_3 \;\hat{}\; (z_1 >> 5) + k[1]_1$;

$y_3 = y_2 \;\&\; top_1$;

$z_2 = z_1 + (y_3 << 4) + k[2]_1 \;\hat{}\; y_3 + Sum_3 + (y_3 >> 5) + k[3]_1$;

$z_3 = z_2 \;\&\; top_1$;

مثلا در branching زیر، متغیر tail نیاز به declare (در top) دارد.

if (bytesRead < 8) {

    tail = (byte) bytesRead;

}

در SSA-form برای این کار از تابع φ استفاده میشود. به این نوع نمایش φ-notation گفته میشود.

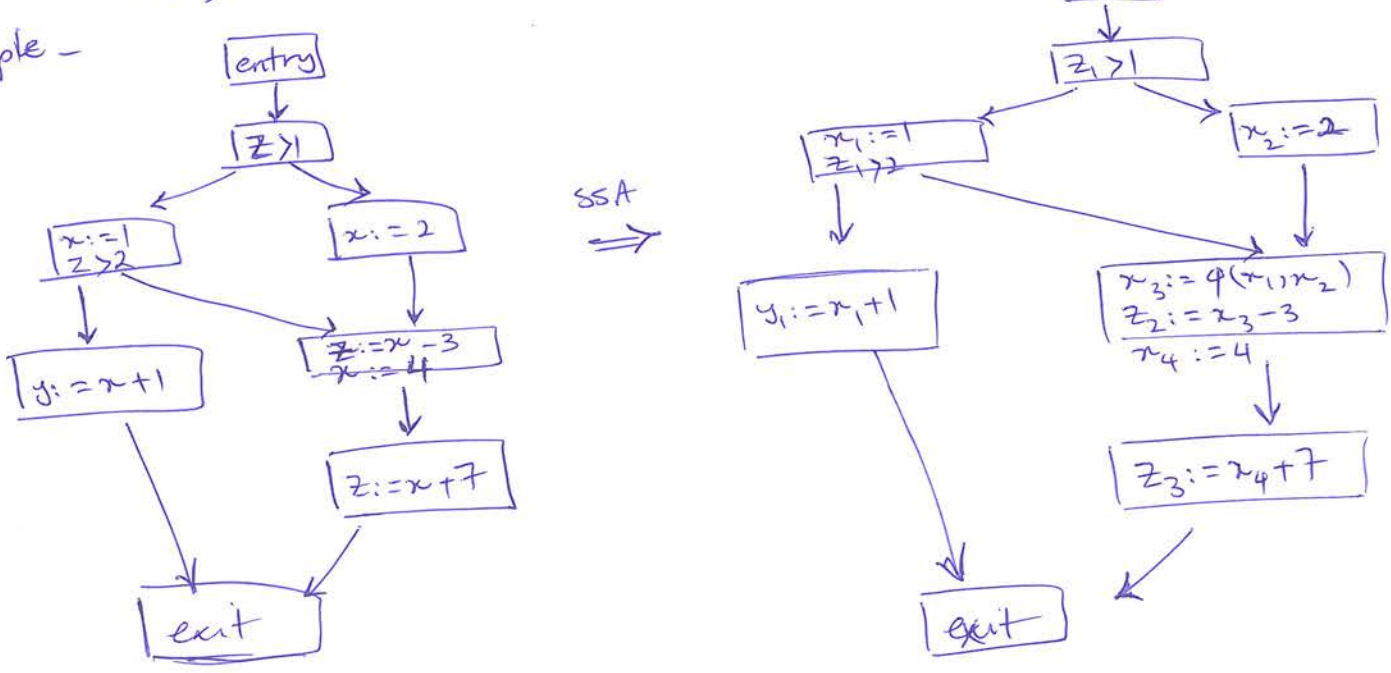
if ($bytesRead_1$ < 8) {

    $tail_2$ = (byte) $byteReads_1$;

}

$tail_3 = \varphi(tail_1, tail_2)$;

Example –



555 - ۹۵۹,۲۱

Program Dependence Graph ⇐ Data Dependence ، Control Dependence ترکیب

(PDG)

The program dependence graph and its use in optimization (Jeanne Ferrata, Knol ottenstein 1987 Joe Warress) TopLAS

System Dependence Graph ، Concurrency ، ... ، PDG ← Rewriting-based Enforcements of Noninterference in Programs with observable Intermediate Values Afshin Lamei, MS Fallah, JUCS

Taint Analysis:

Using dataflow to determine what an attacker can control is called taint propagation.

- Input Validation Problems → BOF, XSS, SQL-Injection

Pointer Analysis:

(Points-to analysis)

(Alias analysis)

Analysis Algorithms:

- To improve context sensitivity

پیاده‌سازی را خوب کنیم، Context زیاد اهمیت ندارد.

مثل) یک ابزار، هر جا strcpy یا error گرفتن می‌بیند، باگ‌های کهن دارد.

Sanitize می‌شود، خیر؛ یعنی همه‌ی حالت‌ها را چک نمی‌کند.

دو الگوریتم کلی در مورد اصل طرح:

intraprocedural analysis

interprocedural analysis

یک function را انتخاب کرده، به صورت جزیره‌ای اجرا می‌کنیم، کار می‌کنیم.
(Individual Functions)

کل برنامه را اجرا می‌کنیم

این دو بخش جدا از هم نیستند، الگوریتم کلی هر دو را با هم پشتیبانی می‌کند.

Local Analysis ← یک AST، CFG، ... (اول‌های اصلی) intra proc. } روش
Global Analysis ← Call Graph استفاده می‌کند. inter proc. } روش

نوع Checking Assertions: (ابزار)

مسئله کنترل امنیتی یک برنامه را به این برمی‌گردانیم که بخش‌های مختلف برنامه با assertion‌های مناسب مشخص شده و اجرا شوند.

Example -

```
assert(alloc_size(dest) > strlen(src));
strcpy(dest, src)
```

≡

BOF نمی‌شود به لطف این ابزار، چیزی را رد نمی‌کند.

نوشتن این assertion‌ها برای مسائل امنیتی خوب می‌تواند سخت باشد و البته کار ماشین و حتی خوب نیست.

چند نوع assertion برای بیان خواص امنیتی داریم ←

1 - taint Propagation e.g. XSS, BOF, ...
(Input Validity)

2 - range analysis
بررسی بازه‌ها (اندیس) (خروجی‌ها)

3 - type state ← یک state ماشین‌وار برای متغیرها e.g. allocated, freed

۱۳۹۵-۹۵,۹,۲۱

- استفاده از ورودی‌های کلی برای برنامه، assertion‌های خصوصی خود را دارد.

e.g. Naive Local Analysis

```
x = 1;
y = 1;
assert(x < y);
```

facts
{}
{x=1}
{y=1, x=1}

از آنجایی که کلی است، مقداری fact نمی‌توان در نظر گرفت.

یکی از ارزش‌های assertion، fact‌ها را می‌سازد.

نمی‌دانیم {x=1, y=1} همواره (x<y) هست یا نه.

به این دلیل نمی‌توان استدلال معتبری ارائه داد.

به اصوص می‌پردازیم به شکل زیر:

```
x = v;
y = v;
assert(x < y);
```

{}
{x=v}
{x=v, y=v}

v، ثابت است؛ یعنی عددی که y برابر با آن است و x از آن کمتر است.

حالا از سمبل‌های مختلف استفاده می‌کنیم:

```
x = v;
if (x < y) {
    y = v;
}
assert(x < y);
```

→

```
x = v;
if (x < y1)
    y2 = v;
assert(x < y2)
```

{}
{x=v}
{x=v, x<y1}
{x=v, x<y1, y2=v}

↳

```
x = v;
if (x < y)
assert(x < y)
```

{}
{x=v}
{x=v, ¬(x<y)} (not)

پس از هر سری عبور، assertion - false از بین می‌رود ← SAT Solver یا Constraint Solver

پس بررسی می‌کنیم که آیا از ای اجرا می‌توان برای assertion جمله‌ای از ارزش‌ها را پیدا کرد یا خیر.

↓

ایجاد ارزش‌های خاص خود را دارد. بعضی از ارزش‌های آن استفاده می‌شود

↳ Constraint Satisfaction Problem (CSP)

SSS_۹۵/۹/۲۴

این مقاله از ابزار Symbolic Execution به معنای جدید استفاده می‌کند.

Narrow کردن روش‌های Program Slicing برای حلقه‌ها دشواری‌هایی دارد. loop-carried data dependency

می‌شود. این مقاله روشی برای حلقه‌ها ارائه داده است که Flow insensitive است و دقت کمتری دارد.

Approach ها

Local Analysis وجود دارد عبارتند از:

Abstract Interpretation: (1996)

(Non-Standard Semantics)

Standard Semantics در آن به معنای اجرای برنامه‌ها با ورودی‌های مختلف است.

Abstract Interpretation معنایی روی معنای برنامه‌ها تعریف می‌کند و به جای اجرای واقعی برنامه، آن را

تحلیل می‌کند.

روش‌ها در تفسیر انتزاعی این است که مقادیر متغیرها از اعداد به بازه‌ها تبدیل می‌شوند.

transition ها state ها برنامه و اجرای هر خط از برنامه روی مقادیر متغیرها است و فقط تعریف می‌شود.

Domain های مختلفی دارد.

Denotational Semantics نوعی از تفسیر انتزاعی است که Category است.

یکی از روش‌های مهم در تفسیر انتزاعی این است که دو کلاس برنامه Flow Insensitive و Flow Sensitive

را جدا می‌کند.

Flow Sensitive به Statement توجه دارد.

→ we want to be Partially flow sensitive.

Flow Sensitive است. کلاس‌های Partially Sensitive بخشی از Flow Sensitive ها را

پیاده‌سازی می‌کند.

* فقط Flow Sensitive در کلاس برنامه Security lattices قابل استفاده است.

این مقاله به جهت بخشی از متغیرها Floating است را از بین می‌برد.

اجرا بستگی دارد.

Predicate Transformers:

approach این یک

یکی از جایگاه‌های Hoare Logic در تعیین صحت ساختار برنامه

↓

مهم‌ترین دستاورد این حوزه

↓

توصیف برنامه

شروع برنامه و پایان آن تعیین شده، ادعاهای برنامه هم قرارهایی دارند.

(Post condition) (Precondition)

ادعاهای برنامه در کنار کدام اعمال هستند قرارهای صحت اولیه - کجاست.

Compositional است

backward است Postcondition ، Precondition این روش

است Postcondition از Precondition ساختن

Model checking:

Temporal Properties یعنی: Temporal Safety Properties بیشتر از جنس ایمنی و نیست

↓

to Property تبدیل

↳ هیچ وقت چیزی بد اتفاق نمیفته.

و اتفاق نمیفته، irremediable است

↓

memory should be freed only once. (مثال)

Computational Induction کلیت این

Property — Safety → Prescribe bad things

Liveness → Prescribe good things

static enforcement قرارها است، dynamic است.

Only non-null pointers should be dereference. (مثال)

(در جنس جایگاه‌ها که نمیشه دسترسی داشت)

Model Checking قابلیت خوبی داره که Temporal Properties رو بررسی کنه.

برای زبان‌های

برنامه‌ها از جنس ماشین هستند. ترتیب دارند و Sequence ها دارند و ... (زبان برنامه‌نویسی)

(Temporal)

... Interval Temporal Logic, Continous Time Logic

Model Checker ⟹ + State machine

(Temporal Logic)

↓

ERROR state

Principles of Model Checking,

Christel Baier and Joost-Pieter Katoen

Model checker

Security Automaton

Global Analysis

Function Call

Global:

```
static char ProgName[128];
void setName (char * newName) {
        strcpy (ProgName, newName);
}
int main (int argc, char * argv[]) {
        setName (argv[0]);
}
```

پیاده‌سازی ← تحلیل‌های بازگشتی مثل memcpy

Work_quene algorithm:

بررسی تمام کلیدها (global یا program) و local انجام می‌شود.

شکل دوم تبدیل این‌ها به Summary برای هر کلید است. اگر تابع را در چند تابع دیگر صدا زده باشیم، همه‌ی آن‌ها هم می‌خواهیم! برایم مهم است!

در این روش می‌رویم تابع‌ها را یک‌به‌یک بررسی می‌کنیم و کلیدها را پیدا می‌کنیم. هر تابع که خلاصه‌ی کلیدها را تغییر دهد، تغییر می‌کند.

تمام کلیدهای local هم باید حذف شوند تا خلاصه‌ی تابع را به‌روز کنیم.

خلاصه‌هایی که با آن کار داریم. مثلاً تابع‌هایی که آن‌ها را صدا زده‌اند.

```
analyze_program (P, Summaries) {
    cg = build_callgraph (p);
    for each function f in P {
        add f to quene
    }
    while (quene is not empty) {
        f = first function in quene
        remove f from quene
        analyze_function (f, quene, cg, summaries);
    }
}
```

این الگوریتم باید برای کلید local داریم دقت کند. داریم global می‌رود. داریم کلید local را (دقت کنید باید از روی تعریف باشد):

```
analyze_function (f, quene, cg, Summaries) {
    old = get summary for f from Summaries;
    do_local_analysis (f, summaries);   ← تابع کلید خروجی خلاصه‌ی تابع را به‌روز می‌کند.
    new = get summary for f from Summaries;   ← آپدیت خلاصه‌ی تابع این‌گونه می‌شود.
    if (old != new) {
```

for each function g in cg that calls f {
    if (g is not in queue) {
        add g to queue;
    }
}

Function Summaries : ... Sound ... whole-program analysis

Call Graph

Research Tools :

- ARCHER → Array Bounds (ARRAY CHECKER)

inter-procedural ... C

Solver ... custom-built-Solver

- BOON → Integer Range Analysis

flow-insensitive

inter-procedural

- LAPSE (light-weight Analysis Programming -- Eclipse)

J2EE

Taint Propagation

SQL Injection, XSS, Cookie-Poisoning, ...

- Pixy

PHP XSS

Inter-Procedural ... Flow-sensitive ... Sanitization

کلاس جزئیات تابع باید از کلاس اصولی باشد. ← مطابقت شود!

ازاین ها: بعضی از این ها: چیزهای امنیتی ← مثلاً ۱۰,۷ تا Shostack

Threat Model based Testing

۹۵/۹/۲۸

موضوع سمینار

← باید: موضوع سمینار ⇒ از ما بپرسند ← broad امنیت موضوع

چیزی شبیه Model-based Security Testing ← (Threat model based Testing) ... ابزارهای Taxonomy

ابزار: Microsoft Threat Modeling tool ← مایکروسافت

کلاس جلسه: بخش: Technical Report for Program Dependence Graph, Ferrante (۱

(Think about PDG for Concurrent systems)

Technical Report ← Pixy (۲

(the method based on Automata) ← اختیاری ← Farzaneh Aghamir's BS Project

کلاس: سمینار: iNTegrity: چیزی شبیه Threat Modeling و جزوه، filing bugs

Security Testing: Security Testing and Ethical Hacking

الان خودت هم از این کار میکنی! Testing

قبلا دیدیم برای هر برنامه یک سری STRIDE امنیتی پیدا میکنیم. الان باید بیاییم ببینیم چه حمله هایی ممکن است Process این ها از طریق ... بیفتد. چه کارهایی باید بکنیم تا فایل ها ... نشوند. برای این کار باید سوالاتی بپرسیم.

What to do when? چه زمانی این کار را انجام دهیم؟

- When to threat model?

1) as you get started on a project. → trust boundary

دکتر بهزادی

۹۵/۹/۲۸
SSS

2) as you work through features.

با تمرکز بیشتر روی ساخت مدل برای کمک کردن در تحقق آن

3) as you get ready to deliever

زمانی که آماده کردن محصول می‌شویم.

پس سه منظر زمانی برای انجام مدل‌سازی تهدید مورد نظر ما هستند:

Starting at the beginning:

- Modeling the system you are planning or building.
- Finding threats against the model.
- Filing bugs

*Bugs may be test cases, feature work, and deployment decisions.

Working through features:

مدل‌سازی تهدید در حین کار روی ویژگی‌ها

- علاوه بر در نظر گرفتن First-order Threats، باید به Second and Third-order هم توجه داشته باشیم.

↳ در کنار mitigation ها

مثال) برای جلوگیری از ورود زور به خانه، تهدید اول این است که شیشه شکسته شود. شیشه را ثابت و دوجداره می‌کنیم، اما این به این معنی است که زور به دیوار بیاید و ... ← تهدیدهای مرتبه‌ی بالاتر

Close to delivery:

موقع تحویل محصول باید مجددا تست‌ها و تهدیدها بررسی و مورد توجه قرار گیرند. تا جایی که شدنی باشد کار را دوباره چک می‌کنیم.

برای سیستم‌های استاتیک پیچیده، فرایند مدل‌سازی کمی‌سخت‌تر هفته، برای پروژه‌های کوچک چندین برابر زمان‌بر است.

What to start and End with?

- A diagram is something between useful and essential input.
  - ↳ e.g. UML diagrams, Data Flow diagram, ...
- You have filed bugs. → باگ‌های مدل‌سازی کمی انتخاب شده
- You have a diagram or diagrams that everyone agrees represents the system.

↓

دیاگرام‌ها در فرایند مدل‌سازی کمی معمولاً waterfall است.

تعداد باگ‌های مدل‌شده با تعداد عناصر دیاگرام‌ها تناسب دارد.

مثلاً STRIDE-Per-elements

(Number of Processes * 6) + (Num of Data Flows * 3) +

(Num of Data Stores * 3.5) + (Num of distinct External Entities * 2)

↓

ارتباط 4، غیر از آن 3

انتظار داریم حداقل این تعداد باگ مطرح شده تا دیاگرام تمام شود. معمولاً بیشتر از حدود STRIDE Per Elements عنصر آن عناصر.

چطوری مدل‌سازی را verify کنیم، خوب است.

بررسی مدل‌سازی‌های مهندسی نرم‌افزار مثل تئوری‌های روان‌شناسی است، تجربی، empirical است نه practice شده و ... .

از چیستم. احیانا باز از این جهت است، شرکت‌های بزرگ دارند از این‌ها استفاده می‌کنند.

Where to start?

ما دوست داریم bottom-up باشد؛ یعنی کامپوننت‌ها از اول شروع کنیم و جمع کنیم. مثل این Shostack

ما دوست داریم top-down باشیم. یعنی کلیه سیستم را نگاه کنیم و حال این را Finding threats "Across" بگیریم.

۹۹/۲/۸
۵۵۵

finding threats "Across"

(Breadth_first Approach)

حرف Shostack این است، یکی یکی المان‌ها را از نظر Component ها بررسی کنیم، یعنی اول سراغ Component ها برویم، بعد سراغ Data Flow ها، بعد سراغ Trust Boundary ها و در آخر سراغ Data Store ها.

Bottom-up ← در این روش ابتدا جزئیات را بررسی می‌کنیم و سپس به سراغ کل می‌رویم.

_ To Iterate across:

- A list of trust boundaries
- A list of diagram elements
- A list of threats

سوال‌ها را در مورد هر یک از این موارد بپرسیم. مثلاً در مورد Trust Boundary ها بپرسیم که آیا Spoofing رخ می‌دهد یا خیر، آیا Tampering رخ می‌دهد یا خیر، و ...

Playing Chess:

در این روش، ابتدا چیزی وارد بازی می‌کنیم و سپس واکنش‌های دفاعی را بررسی می‌کنیم.

به همین ترتیب ادامه می‌دهیم، یعنی واکنش‌های دفاعی را بررسی می‌کنیم و سپس حمله‌ی بعدی را انجام می‌دهیم.

اگر حمله bypass نشد، به همین ترتیب کار را ادامه می‌دهیم.

اصلاحیه‌ها:
- Re-design
- Configuration change
- addition of features

به این روش می‌گویند "Playing Chess" و تمام کتاب‌های هکر در مورد این روش صحبت می‌کنند.

(Arms Race)

Playing Chess یا Across!! چه زمانی باید Prioritize کرد؟ بسته به هدف و استراتژی، برای پیدا کردن تهدیدات از روش Breadth_First استفاده می‌کنیم و ...

Tracking with tables and lists:

- Tracking Threats

Organizing by diagram elements

by threat types

by order of discovery

Diagram Elements, Threat Type, Threat, Bug ID / Title

Example -

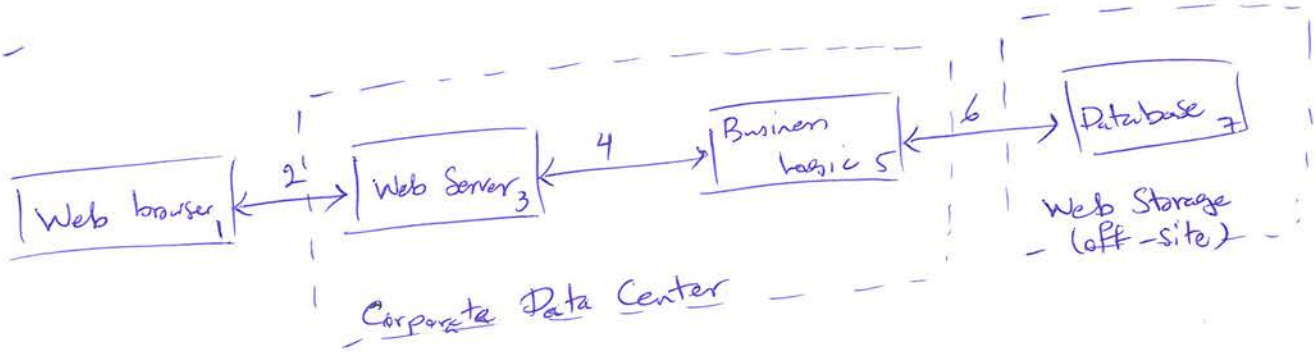


Diagram Element	Threat Type	Threat	Bug ID / Title
Data Flow (4) Web Server to Business Logic	Tampering	Add orders without Payment checks	4553 "need integrity Controls on channels"
	Information Disclosure	Payment instruments in the clear	4554 "need crypto"
	Denial of Service	Can we just accept all these inside the	4555 "check with Alice in IT if these are acceptable"

در ورد بالا Threat Types ، انواع حملات این شکل :

Threat Type, Diagram Elements, Threat, Bug ID / Title

Threat Type	Diagram Element	Threat	Bug ID / Title
Tampering	Web browser (1)	Attacker modifies our JavaScript order checker	4556, "Add order_checking logic to the server"
	Data Flow (2) from browser to server	Failure to use HTTPS	4557 "Build unit tests to ensure that there are no HTTP listeners to these data flow"

در این بخش این روش از حملات را Threat_model-driven Testing ، (یا) چگونه می‌گویند؟

الگوریتم تولید خودکار تست بر اساس مدل تهدید

(یعنی : Automated Threat-based Test Generation)

553 ۹۵،۹۰

خروجی این روش deliverable artifact (outcome) :

۱) مدل تهدید ، مشکلات امنیتی پیدا شده

۲) تست های تولید شده

این guideline برای DFD Component ها و نحوه تحلیل آنها

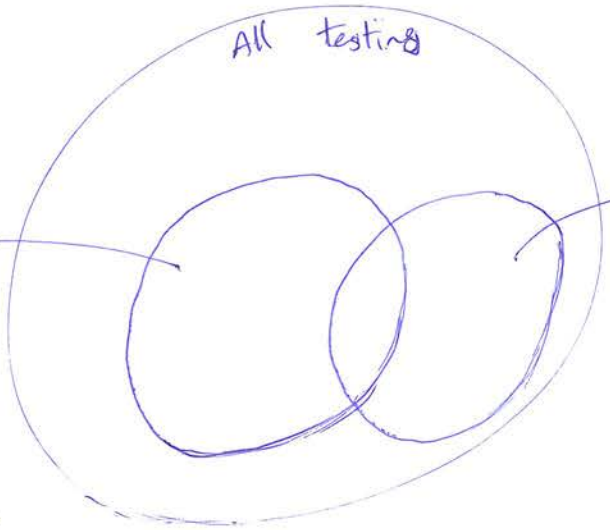
نحوه پیاده سازی mitigation ها برای حملات مختلف

Test های تولید شده Threat-based Testing بر اساس taxonomy

۱۴۳

Testing:

The creation and management of tests.



Reliable Testing

Customer ...

Fuzzing

Threat-model-based

Model-based Testing

Threat-model-based

agile

test-driven development

Two test per threat:

one that exploits the easy (no mitigation) case

and at least one that attempts to bypass the mitigation.

Bug Tracking

IF you use bugs to track test development, you might file two test-creation bugs per threat. One will track the threat, and the other test code for the threat.

Bug ID / Title	

Bug ID / Title	ID (test code)

SSS ۹۵٫۹٫۲۵

- You should setup a way to reproduce the conditions that cause the bug to triggers, and create a test that demonstrates whether a fix is working. (So what?!.. How?!

Tracking bugs for Fixing

Bug ID	Threat	Risk Management Technique	Prioritization Approach	Tactic	Tester	Done?
4556	orders not checked at server	Design change	Fix now to address, avoid dependencies	Code Change	Alice	No
⋮	⋮	⋮	⋮	⋮	⋮	⋮

۱۲۵

تست نفوذ Penetration Testing

A threat-model-based Approach to Security Testing

Software - Practice and Experience Journal

- generate test cases or identify vulnerabilities focusing on specific attacks.
- generate test cases using model-based approaches.
- generate test cases from control policy specifications.

model-based testing

(1) Automated test sequence generation from threat trees.

(2) Transforming the security test into an executable form by considering valid and invalid inputs.

Mutation Testing (Analysis)

۸۸۵, ۹۵, ۹, ۳۰

می‌توانند مراکز ...

... Transforming ... input‌ها هم خودکار کنیم. هم دوری تکراری هم پیچیده.

↓

از روی مدل سناریوی برای ... چالشی

challenging

... Test Case ... Social Engineering ...

... سناریوی حمله ... ! ... ترین ... ممکن ... را ... می‌دهد.

Threat Tree ، attack tree ، misuse case

... "threat based on Petri net nets"

STRIDE ... attack vector ...

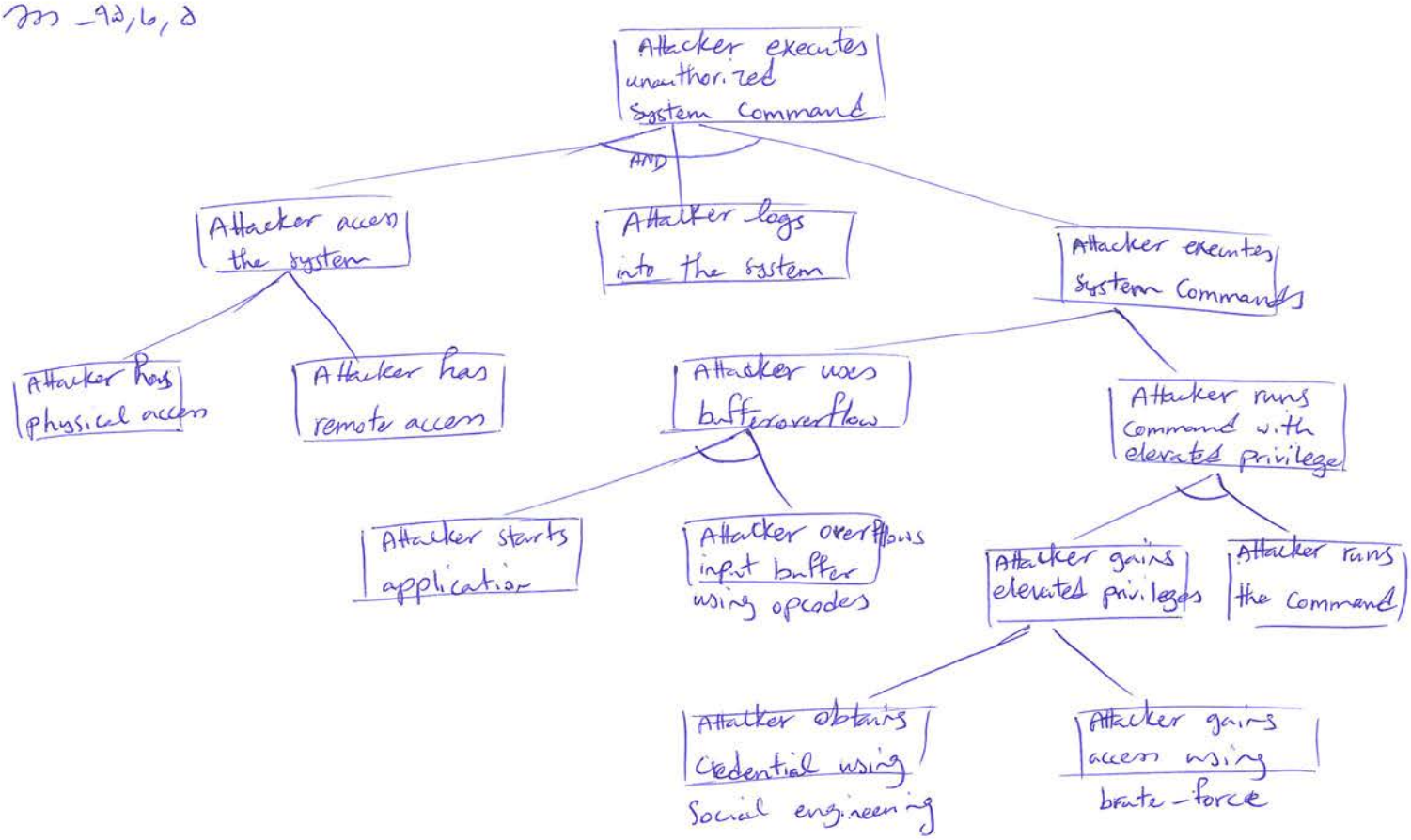
۸۸۵ ۹۵/۰۵

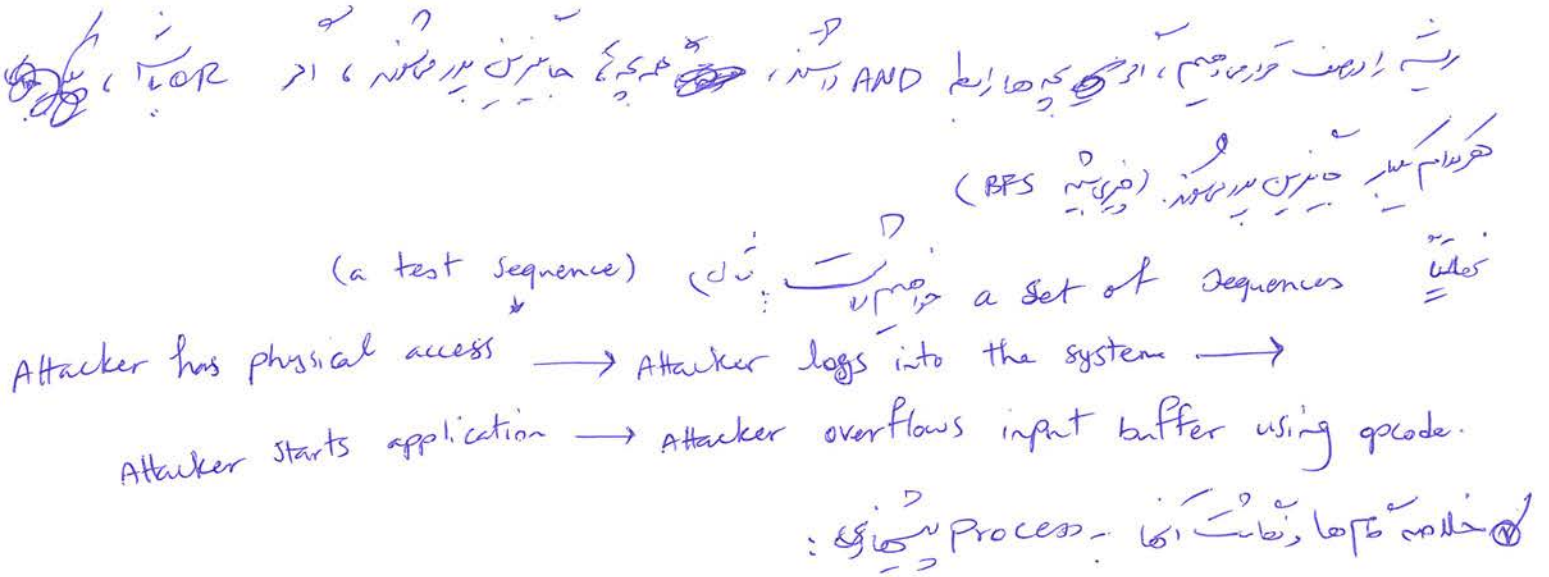
TestGen Activities:

1) imports threat trees created using the 'Microsoft Threat Modeling Tool' and generate test sequences.
2) Provides a graphical interface so that testers map events in the test sequences to the applicable unit tests.
3) add input parameters to test sequences.
4) generate test inputs including valid and invalid inputs.
5) generate test scripts that execute all test sequences with assigned input values.

... TestGen Activities ...

... Attack Tree ...


Attacker executes unauthorized System command
AND
Attacker accesses the system
Attacker logs into the system
Attacker executes System Command
Attacker has physical access
Attacker has remote access
Attacker uses buffer overflow
Attacker runs command with elevated privilege
Attacker starts application
Attacker overflows input buffer using opcodes
Attacker gains elevated privileges
Attacker runs the Command
Attacker obtains credential using Social engineering
Attacker gains access using brute-force


Threat Trees
Test Sequence Generator (1,2,3)
Test Sequences
Integerate Test Inputs into Test Sequences (5)
Model-level Security Test
Convert Model-level Security Tests into Executable Test Cases (5)
Implementation Knowledge
Executable Security Test Cases
Input Syntax
Test Parameter Generator (4)
Test Parameters
(Syntax-based Testing) Input Generation

SSS - 99/10/a

Test Case = Test Value + Prefix Value + Postfix Value + Expected Results + ...
(Test Input)

Test Input

Test Case

Automated

Regular Expression

BNF

(Context-Free)

generator

recognizer

Syntax-based Test Input Generation:

1) retrieve parameter information

(the number of parameters, data type, format, constraints on values)

actual

abstract

overflowed input

2) Create a syntax by using the data type format information and constraints.

3) Constrain the values produced by the syntax.

Context

4) Generate the values.

Example - Overflowed input (BNF-Grammar)

<Overflowed Input> ::= <Input Buffer Characters><Valid Instructions>,
<Jump Opcode><Malicious Return Address>

<Input Buffer Chars> ::= [0-F]^10 → Hex

<Valid Instructions> ::= [0-F]*

<Jump Opcode> ::= EB

<Malicious Return Address> ::= [0-F]^16

حال به راحتی (خودکار) می‌توانید این ها را تولید کنید به صورت خودکار و automatic.

پس روش ما برای تولید تست‌ها دارای سه بخش خواهیم اتوماتیک شد.

ضمن این که ما یک فرم دارد Web App هست، که ISBN دارد. می‌خواهیم این را تست کنیم. به این صورت:

Perl نوشته شده، این را به اتوماتیک کنیم.

```
#usr/bin/Perl_w
#P0 : overflowed Input
use WWW::Mechanize
use HTTP::Cookies
use WWW::Mechanize::FormFiller
use URI::URL;
my $agent = WWW::Mechanize -> new (autocheck => 1);
$agent -> cookie_jar ( HTTP::Cookies -> new);
$myargs = 0;
$agent -> get('http://example.com/Product_info.php?ISBN='."$ARGV[$myargs]");
$agent -> click();
```

این اسکریپت اتوماتیک چون فرم تست را انجام می‌دهد.

- Research Papers + Tools
  - مقالات خوب
  - ابزارهای مورد نیاز
- (Web-Facing Attacks)
- (Language-based Security)
  - Type System - Monitoring