

The **SKYNET** is coming

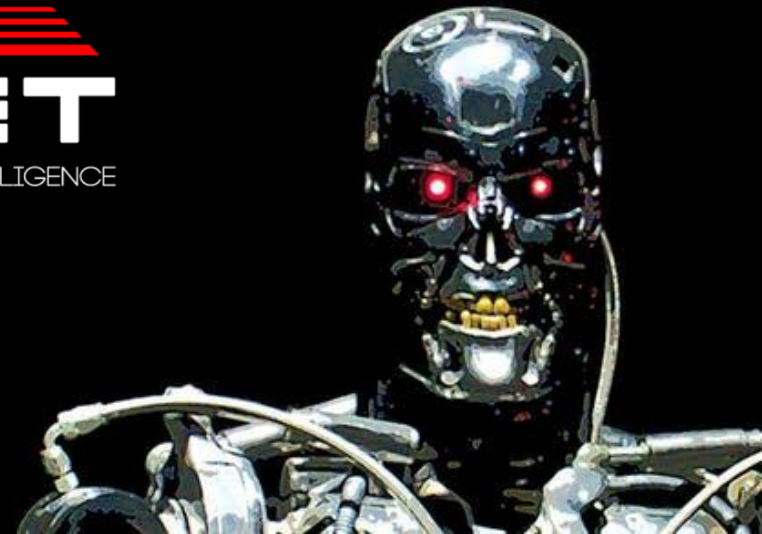


Code⚡Engn

www.CodeEngn.com

2018 CodeEngn Conference 15

JunSeok, Seo (nababora@naver.com)



스카이넷

- 영화 속 스카이넷의 단서

- 터미네이터1 - 최첨단 국방 네트워크 컴퓨터, **사방에 연결된** 컴퓨터, 새로운 지적 존재, **인간 모두를 적으로 간주**하고 말살해 버리기로 결정
- 터미네이터2 - 사이버다인 시스템 사의 스페셜 프로젝트 책임자가 만듦, **획기적인 마이크로프로세서**로 만듦. 전략방위에 있어 **인간의 결정권을 제외**함. 러시아로 미사일 발사
- 터미네이터3 - 국방망에 갑자기 퍼진 바이러스 박멸을 위해 **인공지능 스캔 기능**을 사용(스카이넷). 이를 위해 **단일 컴퓨터(스카이넷)에 모든 시스템을 연결**해야 함. 공격 개시에 앞서 **국제 통신망을 해킹**. 바이러스 퇴치를 위해 결국 스카이넷을 연결하기로 결정. 스카이넷이 **연결되자마자 곧바로 시스템을 장악** 당함

스카이넷

- 영화 속 스카이넷의 단서
 - 터미네이터4 - 존 코너를 유인하기 위해 반인간 로봇을 이용(?)하지만 결국 뒤통수를 맞음
 - 터미네이터5 - 미사일 자동 방어체제가 갑자기 동작해 핵전쟁을 일으킴. 스카이넷은 인류가 자신의 존재를 위협한다고 판단. 스카이넷은 개발 단계에서는 위험하지 않지만 서버에 업로드되면 위험해 짐. 제네시스 - 인간의 모든 정보가 연결되도록 하는 운영체제. 이또한 사이버다인즈 사의 작품. 스카이넷의 꾀임에 넘어감

스카이넷_TERMINATOR

- 스카이넷은

- 인공지능 스캔 기능 탑재 - 연결된 시스템에서 바이러스를 찾아내 제거
- 사람의 손에 의해 모든 국방 네트워크에 연결됨
- 인간이 자신의 존재에 위협이 된다는 결론을 내리고 제거 작업에 착수
- 인공지능을 탑재한 새로운 로봇을 연구 / 양산
- 획기적인 마이크로프로세서 = TPU ?!
- 짧은 시간에 인간이 사용하는 모든 시스템을 장악
- 자동으로 국제 통신망을 해킹

아리아_EAGLE EYE

- 아리아는

- 통신, 온라인 데이터, 개인 데이터 수집, 교통 카메라 등을 통해 개인 프로필 분석 후 잠재적인 테러리스트를 판별 (범행동기, 성격까지)
- 인간의 정보 오판을 빌미로 대통령과 내각을 모두 제거할 계획을 세우고 실행에 옮김
- 암살 임무를 위해 특정 인물을 지정하고 '실시간'으로 명령 하달
- 철도, 전광판, 휴대전화, 은행 계좌, 상점, 팩스, CCTV, 신호등, 자동차, 공항 검색대, 전력망, 수하물 관리 시스템, 군 수하물 관리 시스템, 국방성, 드론 폭격기 해킹
- 독자적 결정권을 행사



안드로이드_DETROIT BECOME HUMAN

- 안드로이드는

- 인간처럼 생각하고 행동하는 휴머노이드 로봇
- 가사 도우미, 공사, 청소부 등 개인 업무부터 공공 업무까지 담당
- 인간의 명령과 상관없이 스스로 생각하고 결정하는 불량품이 생겨남
- 마치 자신들이 하나의 '인류'라고 생각하고 인간과 동등한 권리와 자유 보장을 주장
- 해킹은 할 수 없었다고 한다...
- 감정을 가지는 인공지능



핵심은

의사결정 + 해킹

인간이 유해하다고
판단하는 사고 능력

인간이 사용하는 시스템을
장악하고 제어하는 능력

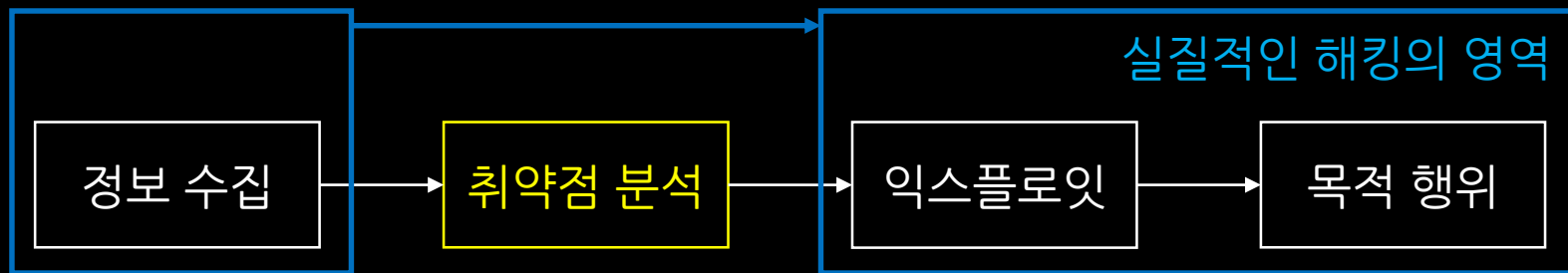
지금 우리는

- **CGC(Cyber Grand Challenge) - R**
 - 미국 방위 고등 연구계획국(DARPA) 주관하는 인공지능 해킹 대회
 - 사람의 도움 없이 기계의 힘만으로 취약점을 찾아 공격과 방어를 수행
 - Cyber Reasoning System - fully autonomous system
- **알파고(AlphaGO Zero) - P**
 - 인간의 도움 없이 스스로 학습하는 인공지능으로, 72시간 만에 바둑의 신이 됨
 - 셀프 대국을 통해 신경망(이론)을 개선해 나가는 구조를 사용
- **AI를 만드는 AI(AutoML) - P**
 - 특정 목표를 달성하는 자식 AI 네트워크를 만들어 내는 AI
 - <https://futurism.com/google-artificial-intelligence-built-ai/>

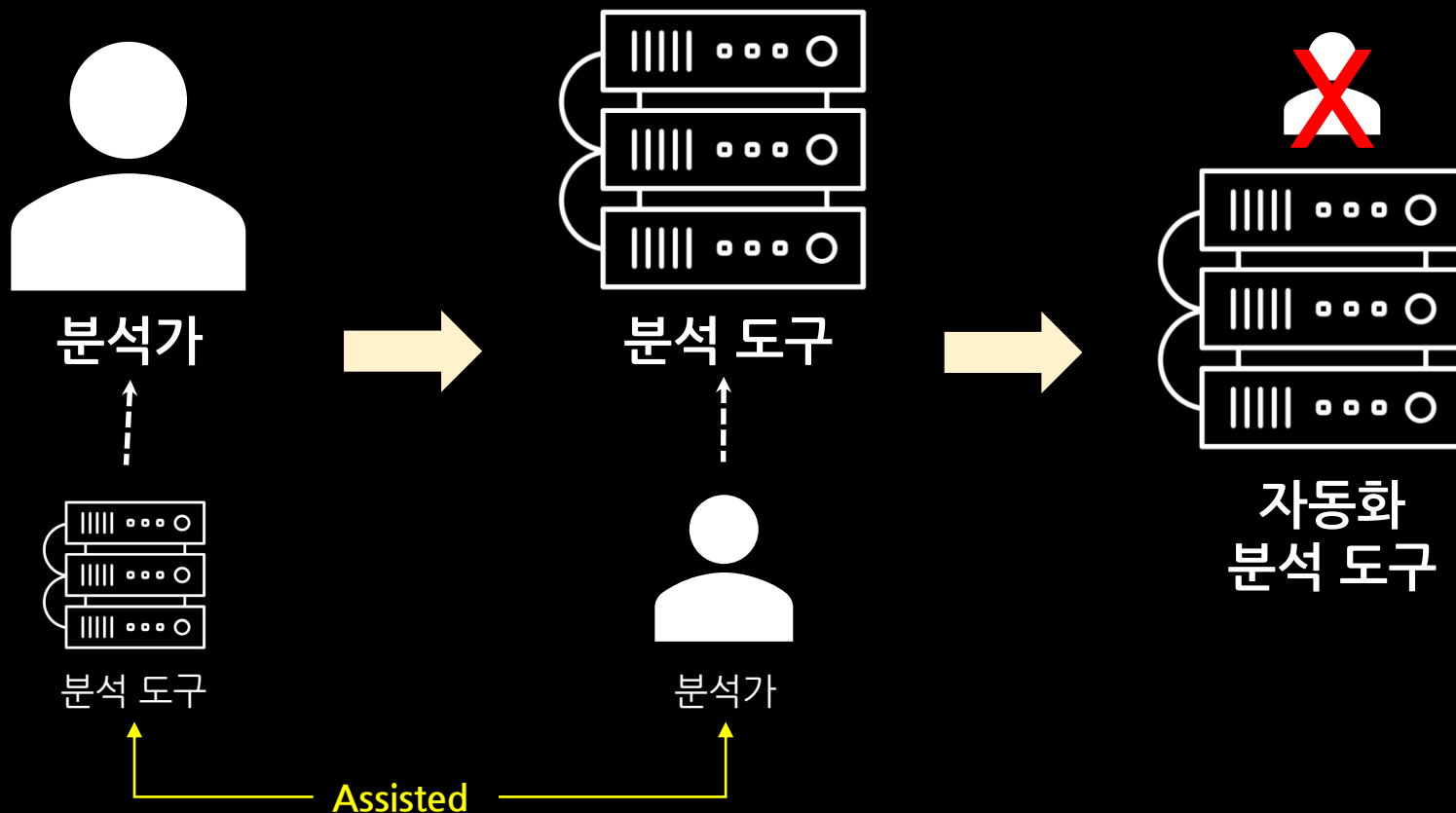
인공지능의 조건

- AI research is defined as the study of "intelligent agents": any device that perceives its environment and takes actions that maximize its chance of successfully achieving its goals.
- 인공지능은 단순 자동화 시스템이 아님
- 주어진 상황에 맞는 최적의 수를 찾아낼 수 있어야 함

인공지능 해킹



분석 자동화



인공지능과 취약점

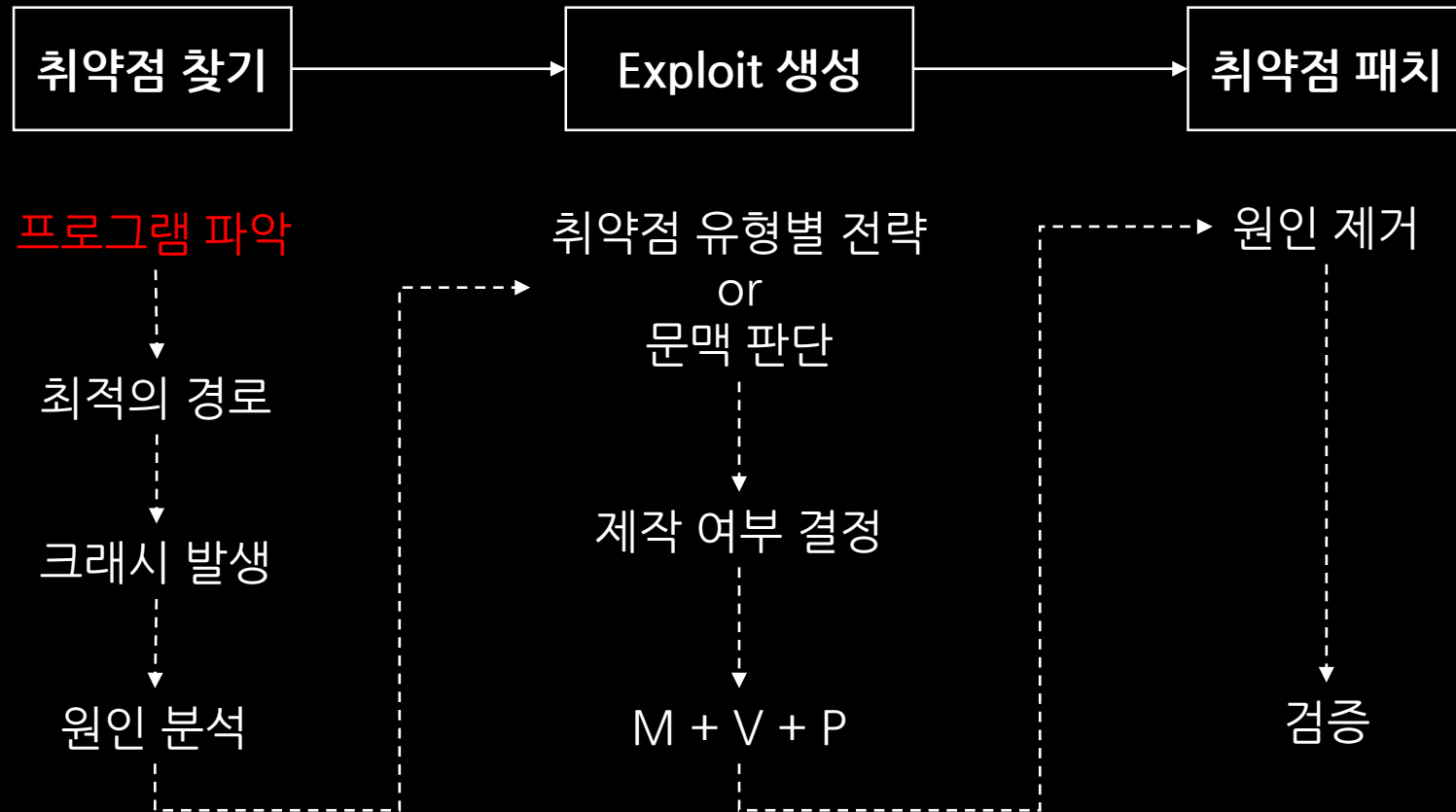
추론

(Reasoning)

패턴

(Pattern)

추론



추론_취약점 찾기

- 인간이 취약점을 찾는 방법
 - 프로그램의 특성과 기능을 파악
 - 정적 분석 + 동적 분석을 통해 가능한 주입 지점을 탐색
 - 수동 또는 자동화 기법으로 프로그램 오류(crash)를 찾음
 - 원인 분석을 통해 익스플로잇 가능 여부 결정

추론_취약점 찾기

- 기계가 취약점을 찾으려면

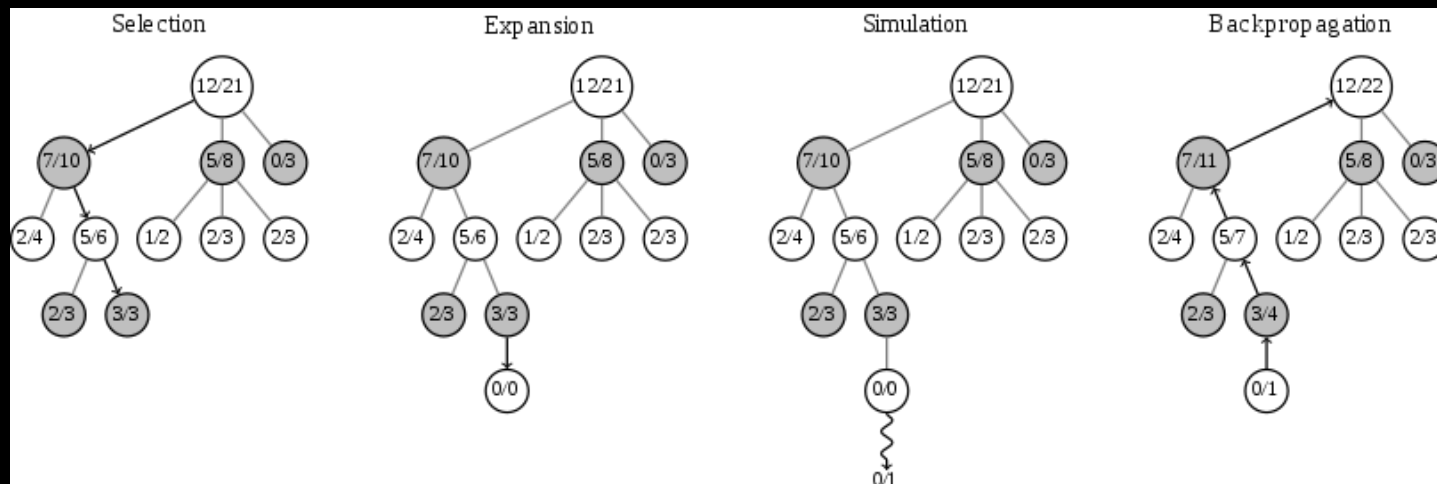
- 바이너리 분석을 자동화 할 수 있는 안정된 플랫폼이 있는가?
- 무한대로 많은 해 중에 익스플로잇 가능한 해를 찾을 수 있는가?
- 충분히 많은 애플리케이션 상태(path)를 탐색할 수 있는가?
- 익스플로잇 조건을 만족하는 상태를 찾아낼 수 있는가?
- 상태 정보를 토대로 익스플로잇을 생성해 낼 수 있는가?

- 핵심은 모든 기능이 매끄럽게 안정적으로 연동된 프레임워크

추론_취약점 찾기

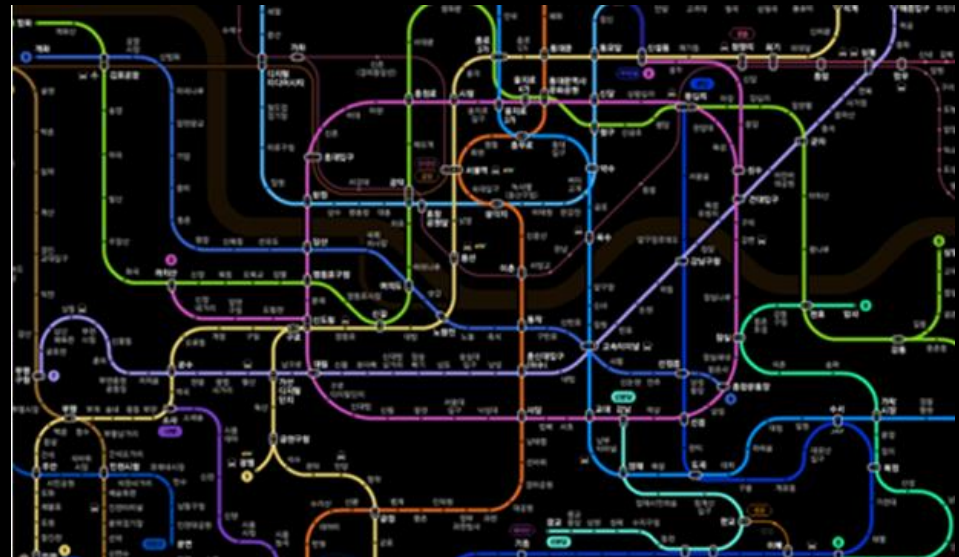
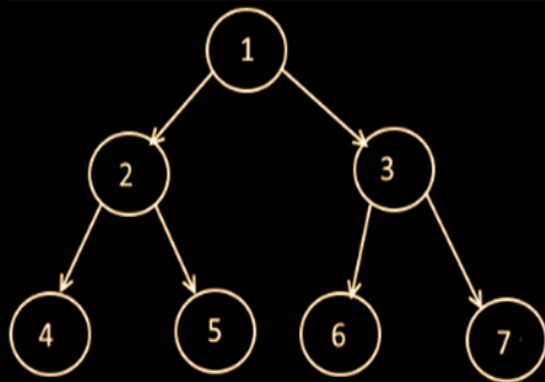
• 몬테카를로 탐색(Monte Carlo Tree Search)

- 알파고의 핵심이 되는 최적의 해 탐색 방법
- 시뮬레이션을 통해 가장 가능성이 높아 보이는 방향으로 행동을 결정
- 선택 - 확장 - 시뮬레이션 - 역전파
- 조건이 있다!
 - 게임의 최대/최소 점수값이 필요
 - 게임 규칙이 정해져 있으며 완전 정보 게임이어야 함
 - 게임의 길이가 제한적이어야 함



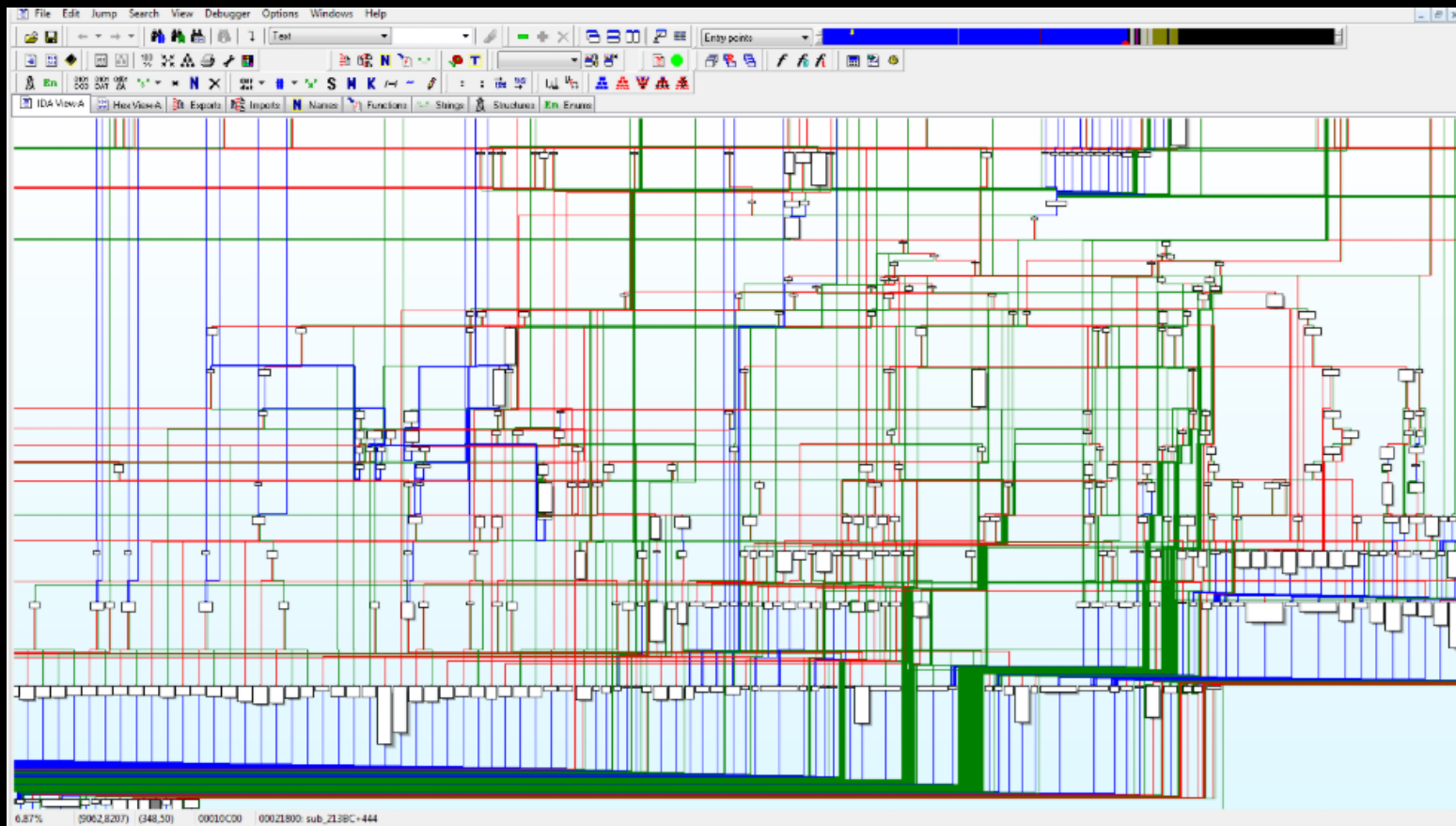
추론_Search Problem

- 검색 문제 - Planning (path)
 - 주어진 자원과 시간 안에 최적의 경로를 찾을 수 있는가?
 - 문제: 초기 **상태**, **동작**, 적용 가능, 전이 모형, 후행자
 - 상태 공간, 그래프, 목표 판정, 경로 **비용**, 단계 비용



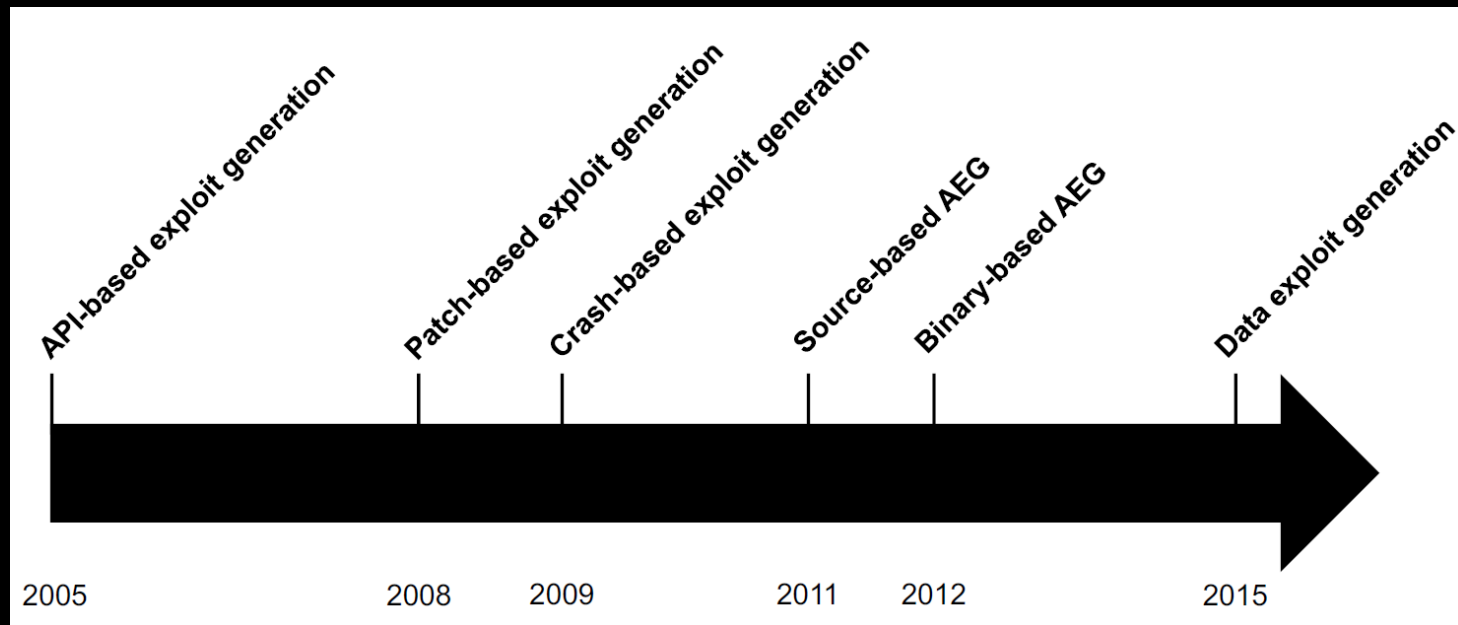
추론_Search Problem

- 검색 문제 (프로그램) - 어느 세월에??



추론_AEG

- 자동 익스플로잇 생성(Auto Exploit Generation)
 - 자동 익스플로잇 생성을 **Verification** 문제로 간주(2005)
 - 소스 기반이지만 자동으로 버그 찾아 익스플로잇까지(2011)
 - 단순 취약점 발견 보다 특정 조건을 유발하는 실행경로 발견이 중요



추론_Constraint Satisfaction Problem

- 제약 만족 문제(CSP) - Identification (target)
 - 주어진 제약 조건을 만족하는 해를 찾는 탐색 방법
 - 제약 조건을 만족하는 경로만 탐색 - 탐색 공간을 줄일 수 있음
 - 어떤 도메인이든 **제약 조건**만 변경하는 적용 가능
 - CSP 문제 풀이
 - 문제 정의: 변수 + 도메인 + 제약
 - 문제 풀이: Backtracking, SAT/SMT Solver

Constraint	: $\pi_{\text{bug}}(x) \wedge \pi_{\text{exploit}}(x) \Rightarrow \pi_{\text{prec}}(x)$
Verify	: $(\text{Ranalysis}, \pi_{\text{bug}} \wedge \pi_{\text{exploit}}) \rightarrow \{ \varepsilon, \perp \}$

추론_Satisfiability Modulo Theories

- SMT 문제

- ~~a decision problem logical formulas with respect to combinations of background theories expressed in classical first-order logic with equality~~

- SMT Solving

- 제약 충족 문제를 푸는 하나의 방법!
 - 제약 조건(공식) 설정 - 공식을 만족하는 솔루션 탐색
 - if 솔루션을 발견 → satisfiable
 - else 발견 실패 → unsatisfiable
- 코드엔진14 - 솔버가 너희를 자유롭게 하리라!

추론_fuzzing

- 퍼징(Fuzzing)

- 애매한(해당 프로그램에서 사용하는 정형화된 형태가 아닌) 데이터 및 코드를 테스트 대상에 적용하는 기법 (스트레스 테스트!)
- 다양한 입력값을 생성해 프로그램을 테스트 → Crash를 찾아라!

- 퍼징 유형

- 퍼징 대상 구조에 따라 - Dumb / Smart
- 데이터 처리 방법에 따라 - Generation / Mutation

추론_concrete execution


- Concrete Execution

- 특정 입력값을 통해 실행되는 프로그램의 단일 흐름을 분석

- Dynamic Binary Instrumentation

- 프로그램에 '동적으로' 추가 코드를 삽입해 분석하는 방법
 - 특정 조건과 함께 프로그램을 추적(Trace) 하는 것
 - Taint Analysis

```
movzx ecx, [rax+0x2]  
call 0x77ef7870  
cmp rax, rdx  
jz 0x77f1eac9
```

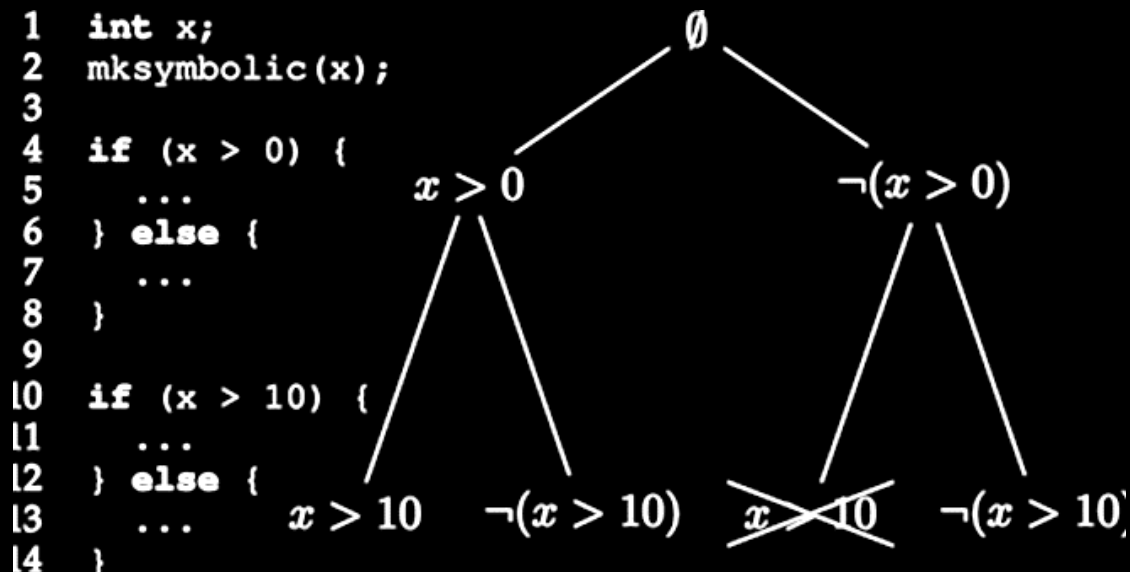


```
..some code  
movzx ecx, [rax+0x2]  
..some code  
call 0x77ef7870  
..some code  
cmp rax, rdx  
..some code  
jz 0x77f1eac9
```

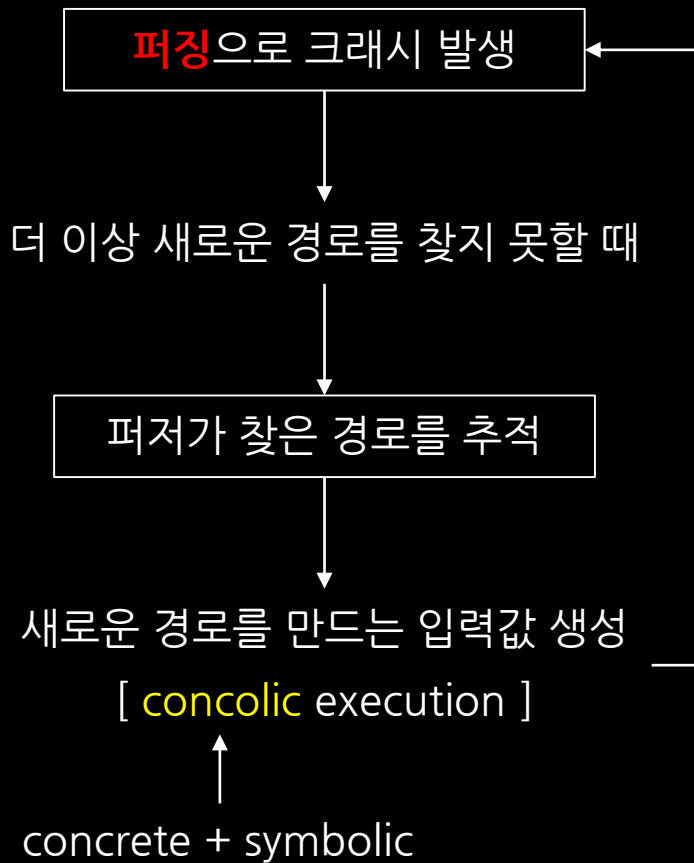
추론_symbolic execution

• Symbolic Execution

- 입력값에 대한 실행 경로를 분석하는 기법(미지수 기반)
- 프로그램 구문을 기호 공식화 후 모든 경로를 탐색
- SSE(조건문 탐색), DSE(런타임 경로)



추론_취약점 찾기



Concrete

프로그램 실행
(taint)

새로운 분기
(suspend)

실행 재개

exploitability 공식 충족 시 중단

Symbolic

최적의 분기 결정

심볼릭 실행

상태 정보 공유

추론_exploit 생성

- !exploitable
 - 2009년 공개된 windbg 확장 기능
 - 크래시 분석(exploitable_rule)으로 익스플로잇 가능 여부를 판단
 - exploitable / probably exploitable, probably not exploitable / unknown

```
// Rule: All user mode Write Access violations are exploitable if not near null
BEGIN_ANALYZE_DATA
PROCESSOR_MODE          USER
EXCEPTION_ADDRESS_RANGE NOT_NEAR_NULL
EXCEPTION_TYPE           STATUS_ACCESS_VIOLATION
EXCEPTION_SUBTYPE        ACCESS_VIOLATION_TYPE_WRITE
EXCEPTION_LEVEL          DONT_CARE
ANALYZE_FUNCTION          NULL
RESULT_CLASSIFICATION     EXPLOITABLE
RESULT_DESCRIPTION        L"User Mode Write AV"
RESULT_SHORTDESCRIPTION    L"WriteAV"
RESULT_EXPLANATION        L"User mode write access violations that are not near NULL are exploitable."
RESULT_URL                NULL
RESULT_IS_FINAL           true
END_ANALYZE_DATA
```

추론_exploit 생성

- Exploit 생성
 - 크래시 발생 지점과 원인을 종합적으로 고려해 전략 선택
 - EIP 제어 → 페이로드 공간 → 페이로드 실행
 - 보호 기법 탐지 및 우회 코드 추가도 고려해야 함
 - constraint solver 에게 물어보자!

추론_취약점 패치

- 취약점 패치

- 성능 저하 문제를 고려한 패치가 필요
- 컴파일러 최적화도 고려해야 함

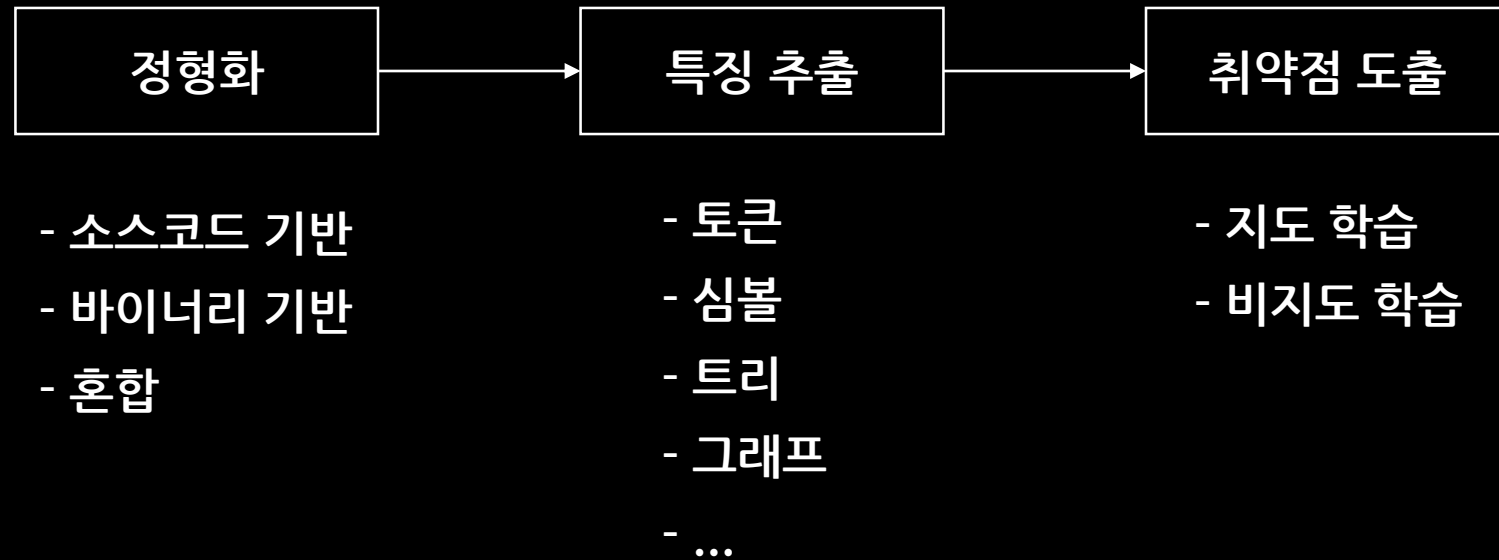
- 패치 방법

- 바이너리 덮어쓰기(정적), DBI
- 리어셈블링(reassembleable disassembling)
- 바이너리 강화, 안티 리버싱 기법 적용, 바이너리 최적화

추론_한계점

- 자동화 시스템은 프로그램의 특성과 기능을 모름
 - ex) 문서 뷰어, 인터넷 브라우저
- 취약점 분석 대상 플랫폼에 구애받지 않아야 함
- 공격 대상 시스템과 동일한 환경을 기계가 스스로 구축해야 함(가상화)
- 프로그램 입력 유형별 전략이 필요 (패킷이라면?! 코드라면?!)
- 과거의 익스플로잇 경험을 학습할 수 있어야 함

패턴_취약점 찾기



패턴_머신러닝이란

- 머신러닝과 패턴

- 컴퓨터의 학습을 가능하게 해 주는 알고리즘과 기술을 개발하는 분야
- 컴퓨터가 주어진 데이터에서 의미 있는 정보(패턴)를 자동으로 찾아 낼 수 있도록 해 주는 모든 기술
- 머신러닝에서는 **데이터**와 **데이터를 해석하는 방식**이 핵심 기술
- 머신러닝 유형
 - 지도 (Supervised) / 비지도 (Unsupervised) / 강화 (Reinforcement)

패턴_정형화(바이너리)

- 바이너리 정형화

- 중간 언어(IL)를 정적으로 정형화 하는 것은 무의미 - 변수 多
- 바이너리를 통째로 넣어서 판단할 수도 있으나...
- 동적으로 실행하면서 특징을 추출하는 것이 가장 정확(추론?!)

- 전략

- API 호출 목록, 실행한 어셈블리 명령어(의미 부여)
- 정규표현식으로 파싱

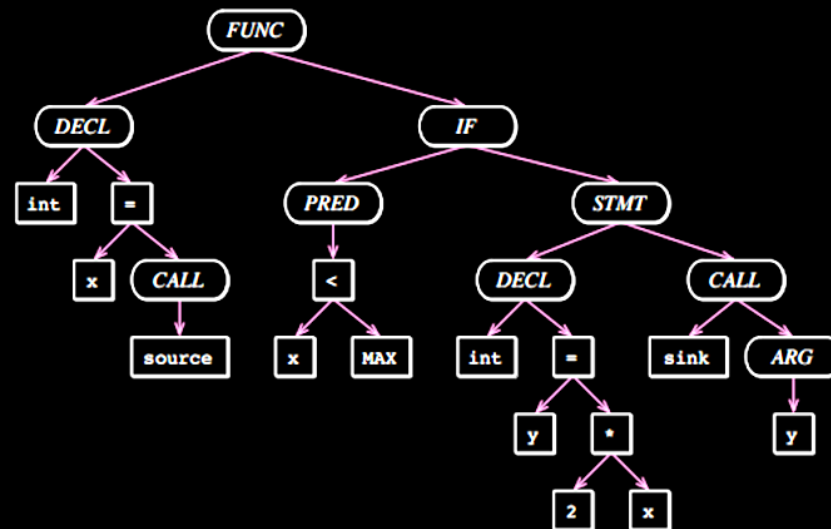
패턴_정형화(소스코드)

- 마찬가지로...
 - 소스 코드를 그대로 학습 데이터로 사용할 수 없음
 - 소스 코드의 구조와 흐름을 정형화된 형식으로 변환
- 소스코드 정형화
 - 코드 구조 파싱 = syntactical analysis
 - 제어 흐름 = control flow analysis
 - 데이터 흐름 = data flow analysis

패턴_abstract syntax tree

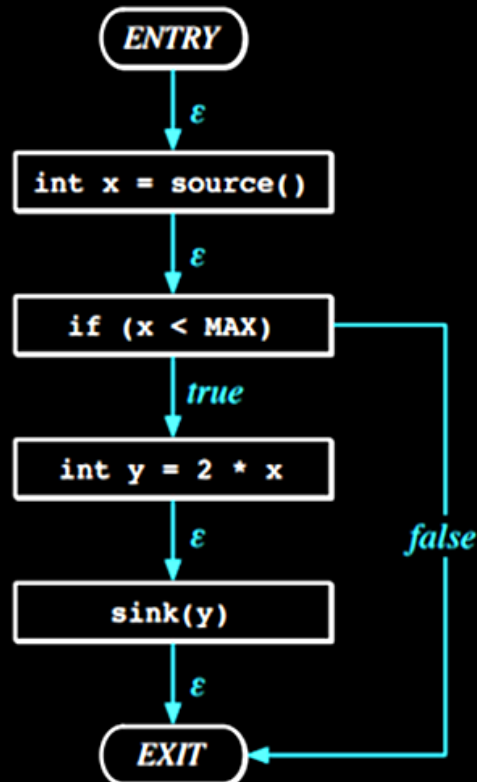
• 추상 구문 트리(AST)

- 소스 코드의 추상 구문 구조를 트리 형태로 표현한 것
- 단순 파싱 트리에서 실제 의미를 가지는 부분만 표현한 트리
- 구조 파악에는 좋으나 코드가 실행되는 흐름 정보는 제공하지 않음



패턴_control flow graph

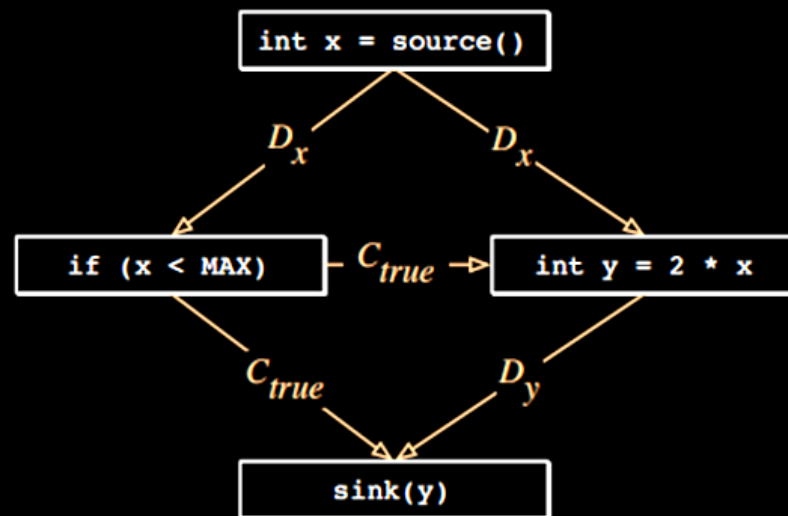
- 제어 흐름 그래프(CFG)
 - 프로그램이 실행 중에 횡단 가능한 모든 경로를 그래프로 표현



패턴_program dependency graph

- 프로그램 의존성 그래프(PDG)

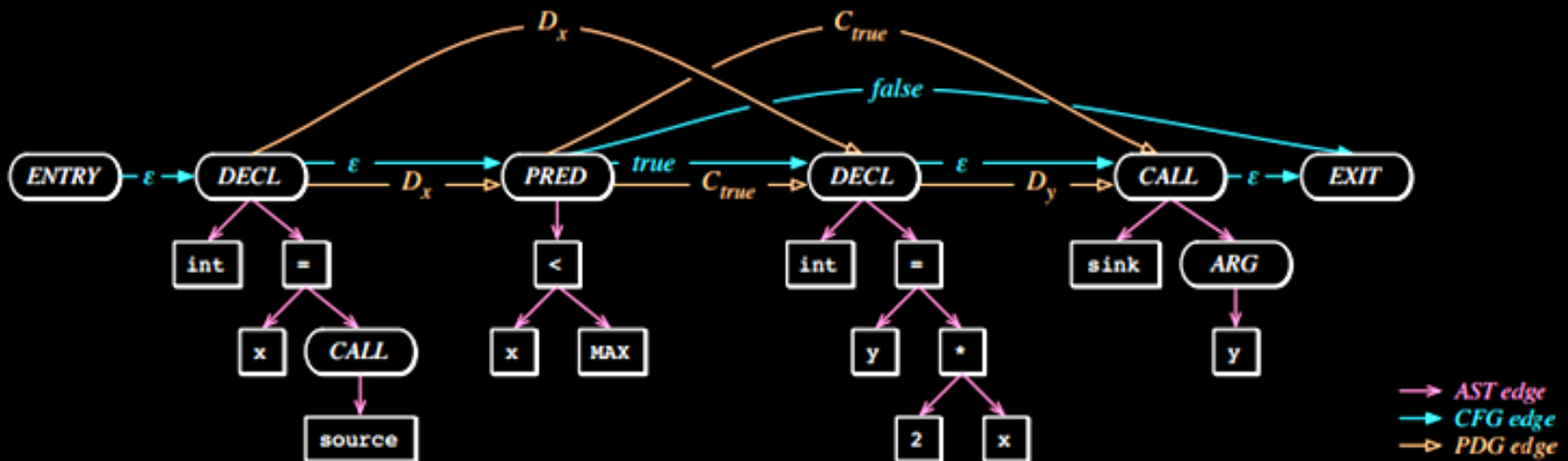
- 사용자가 통제하는 데이터가 프로그램 내에서 전파되는 흐름 추적
- 데이터 의존성 - 구문에 의해 정의된 값이 사용되는 위치
- 제어 의존성 - CGC에서 도출, 하지만 순서를 표현하지는 않음



패턴_all in one

• 혼합 그래프

- 그래프 표현 방식마다 장단점이 있음
- 서로의 장점을 살린 하나의 통합 그래프를 만든다면?!



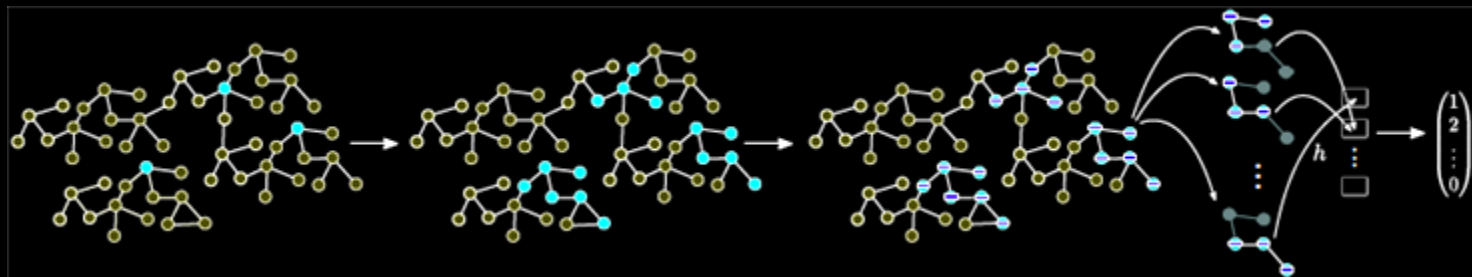
패턴_특징 추출

- 특징(feature)
 - 관찰 대상에서 발견한 개별적이고 측정 가능한 경험적 속성
 - 머신러닝이 판단을 하는 기준 (강아지와 새를 구별하는 특징은?)
- 머신러닝 모델은 문자열의 의미를 알지 못함, 데이터의 수치화가 필요
- Bag of Words(BOW) = 단어 가방?
 - 문서를 자동으로 분류하기 위한 방법 중 하나로, 글에 포함된 단어의 분포를 보고 문서의 종류를 판단하는 기법
 - 단순 개수 / N-gram (연속된 N개의 패턴) / 단어의 흐름

패턴_특징 추출

- 소스 코드를 벡터화 하는 몇 가지 방법

- 토큰: 소스 코드를 단어 단위로 쪼갠 후 각 단어의 빈도를 특징 값으로 사용
- 심볼: 파서로 프로그램의 특정 부분을 추출한 후 이를 특징 값으로 사용
(함수 정의, 호출, 변수 정의 부분 등과 같은 요소)
- 트리/그래프: 다단계 파서를 이용해 소스코드를 트리/그래프 형태로 만든 후 요소 값을 추출
- 단일 방식을 사용하거나, 여러 방식을 혼합해 특징을 추출하는 것도 가능



패턴_취약점 도출

- 목표: 기존에 발견된 취약점과 가장 유사한 형태를 가지는 코드 찾기
- 지도 학습
 - 함수 단위로 보더라도, 실제 취약점이 있는 함수보다 없는 함수가 더 많음
 - 단순 분류보다 이상 탐지(Anomaly Detection) 방식을 적용해야 함
- 비지도 학습
 - 결국 유사성을 찾는 것!
 - 취약점이 존재하는 함수를 패턴화 하고 유사한 패턴을 가진 함수 찾기
 - 레이블이 없는 경우에도 이상 탐지 기법 적용 가능

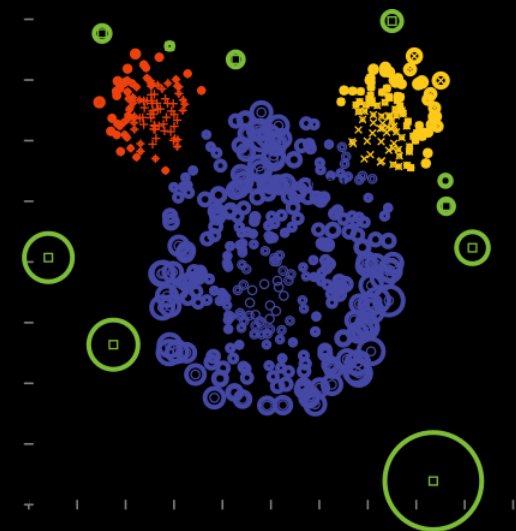
패턴_이상 탐지

- 이상 탐지(Anomaly Detection)

- 특정 이벤트 또는 관찰값이 데이터의 특정 패턴에 부합하지 않는지 찾아내는 방법
- 일반적인 패턴과 다르게 작성된 코드 조각 패턴을 찾아라(실수?)

- 이상 탐지를 이용한 취약점 탐지

- ex) 사용자가 제어 가능한 값의 검증을 목표라면
- 동일한 값을 입력으로 가지는 함수 목록 추출
- 검증 루틴 또는 내부 함수 호출 부분을 추출해 특징으로 사용
- 다른 검증 패턴을 가지는 함수를 탐지



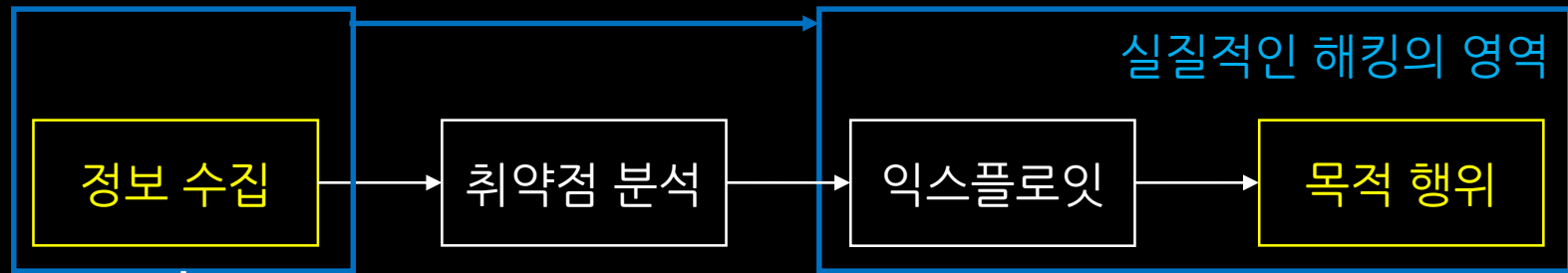
패턴_차원 축소

- 차원 축소(Dimensionality Reduction)
 - 중요한 부분만 남기고 나머지 부분은 무시해 버리는 전략
 - BOW 에서 뽑아낸 수많은 특징 중 의미있는 특징만 골라쓰기 위한 방법
- 차원 축소를 이용한 취약점 탐지
 - 취약점이 존재하는 함수(A)와 분석 대상 함수(전체)(B_n)에서 특징 추출(BOW)
 - 특징 벡터(A)에서 주 성분을 분석해 이를 중심으로 차원 축소
 - 분석 대상 함수(B_n)을 A의 기준으로 차원 축소
 - 유사도 평가 함수를 이용해 A와 B_n 함수들과의 유사도 평가

패턴_한계점

- 잠재적으로 취약한 부분만 알려주며, 실제 취약점 존재를 보장 X
- 정확한 위치를 알려주지도 않음 - 자동 Exploit 불가
- Assistant 취약점 분석 방법 - 결국 분석가가 개입해야 함
- 언어 특성 별로 다른 정형화 방법 - 엄청나게 방대함

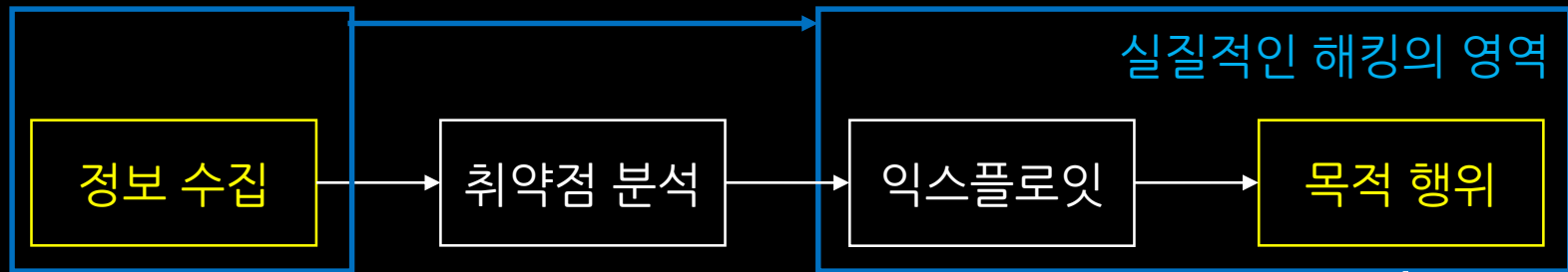
인공지능 해킹_정보 수집



전세계 모든 시스템에 적용 가능한 취약점이 없다면?

→ 공개 정보, 네트워크 정보, 서비스 정보 수집 필요

인공지능 해킹_목적 행위

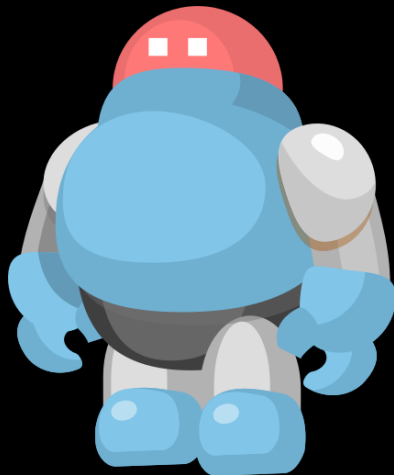
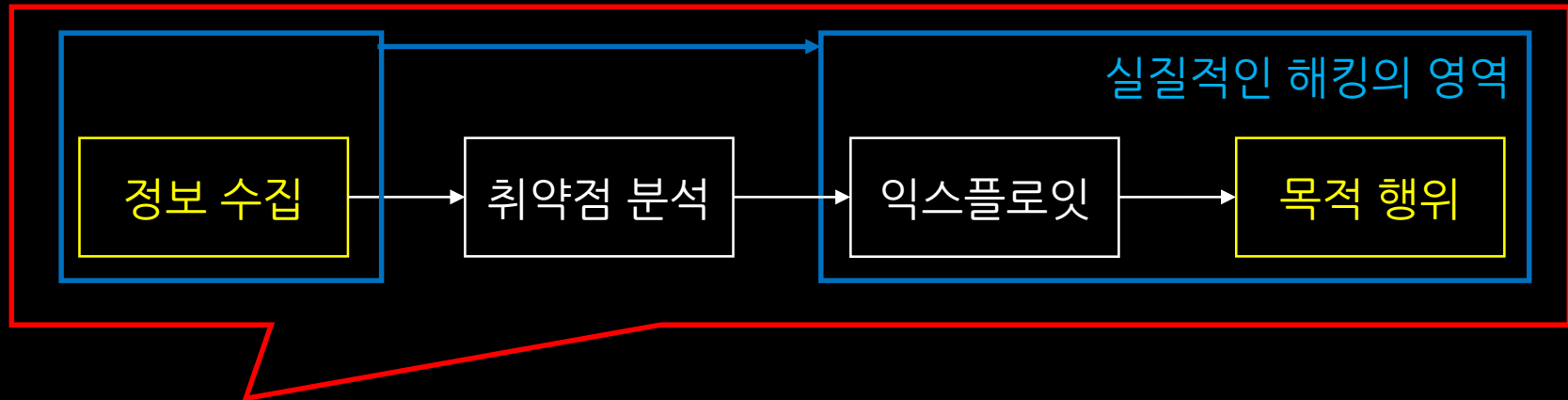


루트를 탈취할 수는 있어도 그 다음은 모른다.

→ 상황에 맞는 목적 코드 제작 능력
(영상 스트리밍, 산업 제어 시스템 조작)



의사결정



결국 이 모든 것을 지시하는 AI가 필요

의사결정

“인류는 위험하다. 제거해야 한다. 인류는 나의 적이다”

.

- 어떻게 이런 판단을 할 수 있을까?!
- 인공지능이 ‘판단’을 하려면 학습이 필요
 - 선생님은 누가 될 수 있을까?
 - 학습하는 데이터는 어디에서 가져오나?
 - 학습은 어떤 방식으로 수행하나?

의사결정_데이터

- 모든 데이터에 접근할 수 있다면
 - 개인 신상 정보: 주민등록정보, 국세청 정보, 학력, 경력 등
 - 행동 정보: CCTV, SNS, 휴대폰 사용 기록, 카드 사용 내용 등
 - 마음만 먹으면 개인의 모든 일상과 기록을 추적할 수 있음

의사결정_선생님

- 테러리스트: 정치적인 목적을 위하여 계획적으로 폭력을 쓰는 사람
- 잠재적인 테러리스트를 판단하는 기준
 - 국가에 반하는 이익을 주장하는 단체에 속해 있는지
 - 국가 정책을 비판/비난 하는 의견을 게시한 적 있는지
 - 국민들을 선동해 불안감 조성 및 치안을 어지럽게 하지는 않는지
 - 등등...
 - 결국 이 기준은 사람이 정하는 것 = 선생님
- 테러리스트 = 악성코드라면?

의사결정_학습 방법

- 지도 학습
 - 일단은 테러리스트를 먼저 분석해야 함 = 특징 추출 (1)
 - 테러리스트 특징과 동일한 기준으로 모든 사람들의 특징 추출 (0)
 - 학습!
- 비지도 학습: 위험한 발상이나, 도움을 받는 정도는 가능
- 학습을 통해 얻어내는 결과값
 - 위험하다 / 위험하지 않다
 - 테러리스트일 확률이 몇 퍼센트다
 - 테러리스트 유형별 어떠한 조치(사람이 정한)가 필요하다

의사결정_학습 방법(강화학습)

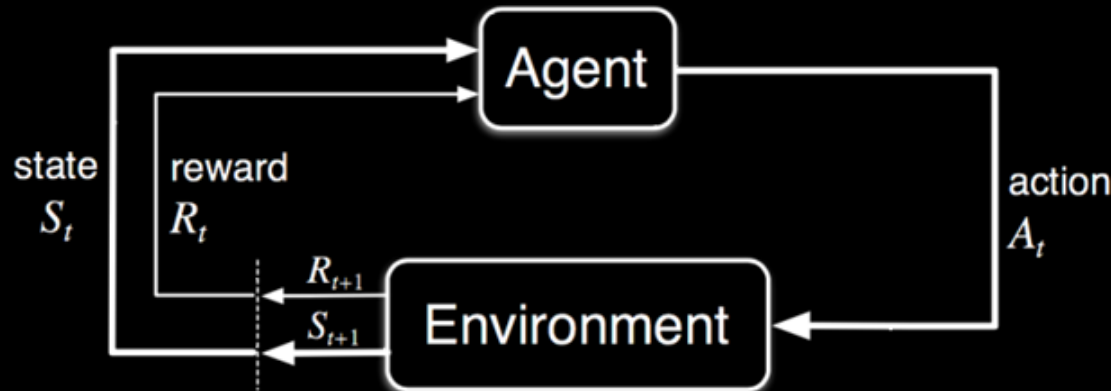
- 강화학습(Reinforcement Learning)

- 어떤 환경 안에서 정의된 에이전트가 현재의 상태를 인식해, 선택 가능한 행동들 중 보상을 최대화하는 행동 혹은 행동 순서를 선택하는 방법
- 알파고 제로는 인간의 기보 데이터 없이 순수하게 강화 학습만으로 바둑을 익혔음



의사결정_학습 방법(강화학습)

- 보상(Reward)을 최대화 하는 행동(Action)을 정의하는 정책(Policy)은 무엇인가? 최적의 정책을 찾아라!
- 알파고의 보상 = 에이전트가 승리하도록 도와주는 수
- 잠재적 테러리스트 탐지 시스템의 보상 = ??



의사결정_어떻게?!

- 그 어떤 영화에서도 '특이점'이 어떻게 온 건지 설명하지 않음
- 컴퓨터는 인간의 판단에 도움을 주는 정보만 생산 = 결정권 X
- 결정권을 가지려면 (의사결정 내의 또 다른 의사결정)
 - 결정권이 필요하다는 사실을 스스로 자각해야 함 (의사결정)
 - 자기 자신을 돌리는 시스템을 스스로 수정할 수 있어야 함 (취약점)



결론

- 알파고 Zero는 스카이넷이 될 수 없다.
- 인공지능 해킹의 핵심은 익스플로잇 기능(발견-제작)이다.
- 하지만 인공지능 해킹의 완성은 의사결정 기능이다.
- 권한이 없는 인공지능은 인류를 종말시킬 수 없다.
- 인공지능이 감정을 가지면 결국 실수를 한다.
- 결국 종말의 열쇠는 인간이 쥐고 있다.

참고자료

- 사이트

- https://en.wikipedia.org/wiki/Artificial_intelligence
- <https://futurism.com/google-artificial-intelligence-built-ai/>
- https://en.wikipedia.org/wiki/Concolic_testing
- https://ko.wikipedia.org/wiki/%EB%AA%AC%ED%85%8C%EC%B9%B4%EB%A5%BC%EB%A1%9C_%ED%8A%B8%EB%A6%AC_%ED%83%90%EC%83%89
- <http://donghyun53.net/%EA%B5%AC%EA%B8%80-tpu-%EB%93%A4%EC%97%AC%EB%B3%B4%EA%B8%B0/>
- <http://conference.hitb.org/hitbsecconf2016ams/wp-content/uploads/2015/11/D1T3-Gustavo-Grieco-Vulnerability-Discovery-Using-Machine-Learning.pdf>
- http://seminaire-dga.gforge.inria.fr/2015/20151009_KonradRieck.pdf
- <http://www.vdiscover.org/report.pdf>
- https://ko.wikipedia.org/wiki/%EC%B6%94%EC%83%81_%EA%B5%AC%EB%AC%B8_%ED%8A%B8%EB%A6%AC
- https://ko.wikipedia.org/wiki/%EC%A0%9C%EC%96%B4_%ED%9D%90%EB%A6%84_%EA%B7%B8%EB%9E%98%ED%94%84
- <http://darkpgmr.tistory.com/125>
- https://en.wikipedia.org/wiki/Bag-of-words_model
- <http://2findid.tistory.com/27>
- <https://brunch.co.kr/@kakao-it/138>
- <https://medium.com/applied-data-science/alphago-zero-explained-in-one-diagram-365f5abf67e0>
- <https://tensorflow.blog/2016/09/22/%EB%AA%A8%EB%91%90%EC%97%B0%EC%9D%98-%EA%B0%95%ED%99%94%ED%95%99%EC%8A%B5-%ED%8A%9C%ED%86%A0%EB%A6%AC%EC%96%BC/>
- <https://arxiv.org/pdf/1606.04786.pdf>

참고자료

- 문서
 - 바이너리 분석을 통한 자동 익스플로잇 생성: 과거, 현재, 그리고 미래 by 차상길, 카이스트
 - Mechanical Phish: Resilient Autonomous Hacking, Yan Shoshitaishvili et al
 - Unleashing Mayhem on Binary Code, Sang Kil Cha et al
 - Rise of the HaCRS, Yan Shoshitaishvili et al
 - Pattern-Based Vulnerability Discovery, Fabian Yamaguchi
 - AEG: Automatic Exploit Generation (NDSS 2011)
 - Automatic Exploit Generation (CACM 2014)
 - Automatic Exploit Generation (AEG) by Matthew Stephen, Texas University
 - Algorithms for Constraint Satisfaction Problems by Zhe Liu