프로그램 구조와 상수 값을 이용하는 바이너리 실행 파일의 차이점 분석

Journal of KIISE 2008. 7
박희완, 최석우, 서선애, 한태숙

김영철
2016. 3. 18.

서론

- 바이너리 실행 파일의 차이점 분석
  ➔ 패치를 통해 변경된 부분 탐지
  ➔ 코드 도용 탐지

- 상향식 분석 방법
  ➔ 명령어 차이점 같은 세밀한 분석

- 하향식 분석 방법
  ➔ 프로그램 구조를 이용한 분석

서론

- 하향식 분석 방법 + 상수 값 요약
  ➔ 하향식 분석법의 장점에 섬세함을 더함.

관련연구

- 명령어 기반 비교 방법
  T. Sabin – 그래프 일치를 이용하여 비교하는 알고리즘 제안

  DarunGrim – 기본 블록부터 비교를 시작하고 함수 단위 매칭으로 비교를 끝내는 상향식 분석 방법 사용

관련연구

- 구조 정보를 이용하는 비교 방법
  T. Dullien – 코드의 구조 정보를 바탕으로 비교하는 방법 제안
  *구조 정보 : 함수 내의 CFG와 함수 호출에 대한 요약 정보

  T. Dullien & R. Rolles
  - CFG 기반 비교 기법을 일반화, selector와 property 개념 도입

바이너리 코드 비교를 위한 알고리즘

- 구조적인 비교 + 상수 단위 비교

- 구조적인 비교
  함수 단위 구조 비교
  함수를 구성하는 기본 블록 단위 비교

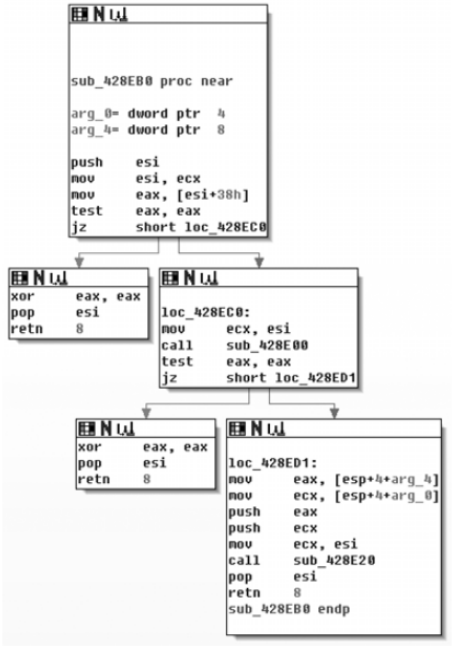
- 상수 단위 비교
  구조적인 비교에서 섬세함이 떨어지는 것을 보완

바이너리 코드 비교를 위한 알고리즘

- 바이너리 구조 정보의 요약
  1. 함수 요약하기

$$\alpha(f) = (i, j, k)$$

i : 함수에 포함된 기본 블록의 개수
j : 함수에 포함된 기본 블록 간의 연결선의 개수
k : 함수에 포함된 함수 호출의 개수



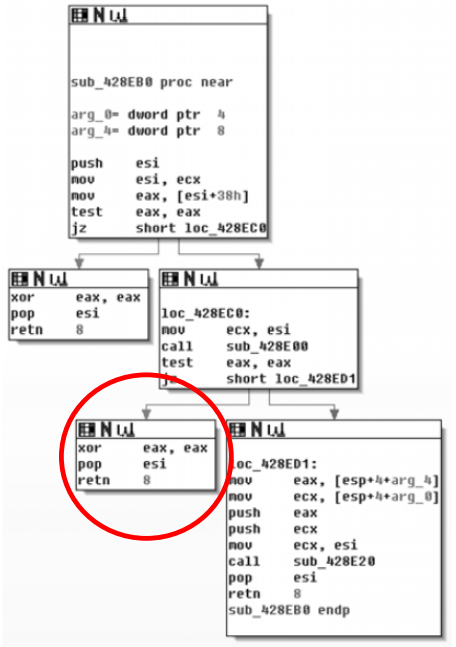
$\alpha(f) = (5, 4, 2)$

바이너리 코드 비교를 위한 알고리즘

- 바이너리 구조 정보의 요약
  2. 기본 블록 요약하기

$$\beta(n) = (i, j, k)$$

i : 함수 시작 지점 – 기본 블록까지의 최소 블록 개수
j : 기본 블록 – 함수 종료 지점까지의 최소 블록 개수
k : 기본 블록에서의 함수 호출 개수



$\beta(n) = (3, 1, 0)$

바이너리 코드 비교를 위한 알고리즘

- 바이너리 구조 정보의 요약
  2. 기본 블록 요약하기

$$\gamma(n) = SPP(\{c_1, \ldots, c_k\})$$

$C = \{c_1, c_2, \ldots, c_n\}$ 모든 명령어 집합

$P = \{2, 3, 5, \ldots p_n\}$ 모든 소수 집합

$$SPP(\{c_1, c_2, \ldots, c_k\}) = \prod_{i=1}^{k} sp(c_i)$$

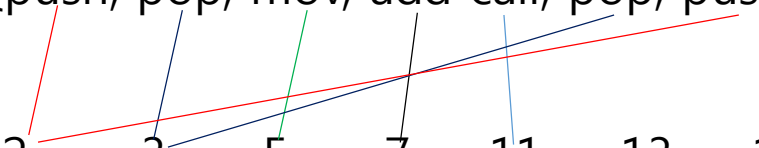
sp : 명령어를 하나의 유일한 소수에 매핑하는 함수

바이너리 코드 비교를 위한 알고리즘

- 바이너리 구조 정보의 요약
  2. 기본 블록 요약하기

$$\gamma(n) = SPP(\{c_1, \cdots, c_k\})$$

C = {push, pop, mov, add call, pop, push}



P = {2, 3, 5, 7, 11, 13, 17}

SPP{C} = 2 * 3 * 5 * 7 * 11 * 13 * 16 * 2 * 3

➔ SPP값의 소인수 분해를 통해 원본 명령어 집합을 구하는 것이 가능

바이너리 코드 비교를 위한 알고리즘

- 상수 정보의 요약

$$\delta(f) = \{(v_1, n_1), (v_2, n_2), \ldots, (v_n, n_n)\}$$

v_n : 상수 값
n_i : 블록 내에서 v_n이 나타나는 개수

$\delta(f) = \{(0x800, 2), (0x100, 1)\}$

➔ 함수 f내에서 0x800이 2번, 0x100이 1번 쓰임

바이너리 코드 비교를 위한 알고리즘

- 바이너리 비교 알고리즘
  1. 함수의 매칭 알고리즘

$$p(f_i) = g_j \quad \stackrel{def}{=} \quad (\forall k \neq i: \ \alpha(f_i) \neq \alpha(f_k))$$
$$\wedge (\forall l \neq j: \ \alpha(g_j) \neq \alpha(g_l))$$
$$\wedge (\alpha(f_i) = \alpha(g_j))$$

알고리즘을 통해서 최대한 많은 함수 매칭을 하도록 한다

바이너리 코드 비교를 위한 알고리즘

- 바이너리 비교 알고리즘
  1. 함수의 매칭 알고리즘

$$p_0 = p.$$

$$p_{i+1}(f_a) = \begin{cases} p_i(f_a) & \text{if } f_a \in dom(p_i), \\ g_c & \text{if } \exists f_b \in dom(p_i) : \begin{array}{l}(p_i(f_b) = g_d) \\ \wedge (f_b \rightarrow_f f_a) \\ \wedge (g_d \rightarrow_f g_c),\end{array} \\ undefined \text{ otherwise.} \end{cases}$$

$dom(p_i)$: 함수 p_i의 정의역

$\rightarrow_f$: 함수 호출 관계

바이너리 코드 비교를 위한 알고리즘

- 일치율 계산
  1. 구조적 일치율 계산

α(f) = (i1, j1, k1) α(g) = (i2, j2, k2)

$$\text{구조적 일치율} = \frac{\alpha(f) \cdot \alpha(g)}{|\alpha(f)|\,|\alpha(g)|}$$

$$= \frac{i_1 i_2 + j_1 j_2 + k_1 k_2}{\sqrt{i_1^2 + j_1^2 + k_1^2}\,\sqrt{i_2^2 + j_2^2 + k_2^2}}$$

α(f) = (1, 2, 3) α(g) = (2, 4, 6)

이 경우, 구조적 일치율은 1이지만,
함수 매칭 알고리즘에서 매칭되지 않음

바이너리 코드 비교를 위한 알고리즘

- 일치율 계산
  1. 상수 일치율 계산

$$\delta(f) = \{(v_1, q_1), (v_2, q_2), \ldots, (v_m, q_m)\}$$

$$\delta(g) = \{(w_1, r_1), (w_2, r_2), \ldots, (w_n, r_n)\}$$

$$\text{상수일치율} = \frac{2|\delta(f) \otimes \delta(g)|}{|\delta(f)| + |\delta(g)|},$$

단, $|\delta(f)| = \Sigma_{i=1}^{m} q_i$, $|\delta(g)| = \Sigma_{i=1}^{n} r_i$ 이다. ➔ 분모 = 상수의 개수의 총 합 * 2

$$\delta(f) \otimes \delta(g) = \{(v, \min(q, r)) | ((v, q) \in \delta(f)) \wedge ((v, r) \in \delta(g))\}$$

➔ 분자 = 두 함수에서 일치하는 상수의 개수 * 2