Hartowanie kodu

W ostatnich latach podatności bezpieczeństwa komputerowego zaczęły trafiać na pierwsze strony gazet. Czy to poprzez chwytliwie brzmiące nazwy typu Heartbleed lub Spectre, czy to w wyniku ogromnych wycieków danych organizacji rządowych lub mediów społecznościowych. Nawet największe i najbardziej szanowane firmy z branży IT, które mogą sobie pozwolić na własne zespoły specjalistów ds. bezpieczeństwa komputerowego, nie są w stanie poradzić sobie z prostym faktem – nowoczesne oprogramowanie staje się tak duże i tak skomplikowane, że błędy bezpieczeństwa wydają się nieuchronne.

W tym artykule przyjrzymy się podejściu do projektowania oprogramowania, które pomaga eliminować całe klasy błędów, bez potrzeby zatrudniania setek inżynierów bezpieczeństwa, którzy będą zajmować się stałym audytem kodu.

Podejście to w firmie Google znane jest od lat pod szyldem *safe coding*. Wzmianki o nim można znaleźć m.in. w artykule Christopha Kerna pt. *Securing the Tangled Web*[1] w *Communications of the ACM (2014)*.

CZEMU TRUDNO UNIKNĄĆ BŁĘDÓW BEZPIECZEŃSTWA?

Popularne języki programowania i biblioteki były przez lata projektowane z myślą o ekspresywności, nie o bezpieczeństwie, jako nadrzędnym celu. Aby napisać bezpieczny kod, często trzeba się mocno nagimnastykować i mieć dużą wiedzę, a i tak często musimy polegać na tym, jak zachowuje się reszta oprogramowania. To ostatnie zmienia się z biegiem czasu, często w innych plikach źródłowych, a zmiany te dokonywane są przez różnych członków zespołu. Założenia, które zapewniały kontrolowany dostęp do wrażliwych danych, mogą się zmienić w sposób kompletnie niezauważalny.

Jak pokazuje doświadczenie, nawet najmniejsze błędy zostaną skrupulatnie wykorzystane przez kreatywnych atakujących.

JAK WYGENEROWAĆ URL

Na przykładzie tworzenia adresów URL pokażemy, jak można nie tylko ułatwić audyt kodu, ale też i całkowicie wyeliminować taką potrzebę w większości przypadków.

Listing 1. Generowanie adresu URL

```
String url = "https://" + address;
if (isCustomer(user)) {
  List<String> urlParams = new ArrayList<String>();
  for (Map.Entry<String, String> p: params) {
    urlParams.add(
      p.getKey() + "="
      + URLEncoder.encode(p.getValue(), StandardCharsets.UTF_8));
  }
  url += "/?" + Strings.join("&", urlParams);
}
```

Kod w Listingu 1 generuje adres URL. W linijce 7 da się zauważyć, że autor próbuje w jakiś sposób zabezpieczyć swój program poprzez kodowanie wartości parametru zapytania. Mimo tego jest w nim wiele potencjalnych zagrożeń. Główne z nich to:

1. Klucze słownika `params` nie są kodowane przed dodaniem ich do `urlParams`. Czy jest to pomyłka, czy może są już one w poprawnej formie w słowniku?[2]
2. Nie wiemy niczego o tym, co może zawierać zmienna `address`. Czy jest pod kontrolą użytkownika? Czy może się to zmienić z biegiem czasu w otaczającym kodzie?

BEZPIECZEŃSTWO KOSZTEM ELASTYCZNOŚCI

Powyższy sposób na budowanie adresów URL daje nam bardzo dużo swobody. Taka elastyczność z pewnością zostanie wykorzystana do granic przez kreatywnych programistów. Nawet jeśli mają oni najlepsze intencje, znacząco podnosi to prawdopodobieństwo pojawienia się błędu – jeśli nie od razu, to po wielu modyfikacjach kodu z biegiem czasu.

W powyższym przykładzie znajduje się kilka błędów. Jednak nawet gdyby one nie występowały, bardzo trudno jest taki kod zrecenzować (ang. *code review*). Zmienna `address` na pierwszy rzut oka może przyjąć dowolne wartości, niewiele wiadomo o tym, czy klucze w słowniku `params` są odpowiednio zakodowane itd.

Zamiast swobodnego sklejania napisów, proponujemy funkcję `TrustedURL::FromConstant`, której prototyp wygląda tak:

```
TrustedURL FromConstant(StringLiteral url)
```

gdzie `StringLiteral` jest typem, którego wartościami są wyłącznie literały napisowe (różnie realizowane w zależności od języka programowania, patrz ramka „Akceptowanie wyłącznie stałych czasu kompilacji”).

1. https://dl.acm.org/doi/abs/10.1145/2643134

2. Niestety, aplikowanie `URLEncoder.encode` „na wszelki wypadek” nie zda tutaj egzaminu, bowiem podwójne zakodowanie popsułoby adres URL.

Zamów Programistę w prenumeracie >>