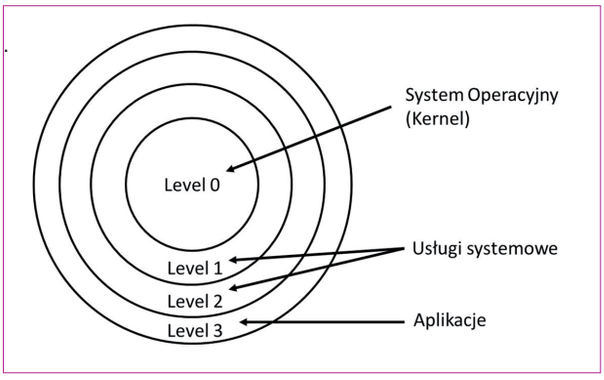
CPU BACKDOOR – CZYLI PO CO WYWAŻAĆ OTWARTE DRZWI?

Adam Kostrzewa

CPU BACKDOOR – CZYLI PO CO WYWAŻAĆ OTWARTE DRZWI?

Adam Kostrzewa
(adamkostrzewa@gmail.com)

Backdoor to metoda ominięcia mechanizmów bezpieczeństwa wykorzystująca podatność w systemie lub sprzęcie. Od dawna dyskutuje się na temat możliwości celowej implementacji podatności sprzętowych w produktach komercyjnych, np. komputerach typu mainframe, routerach czy procesorach desktopowych [3] pozwalających na łatwą implementację backdoora. Jak dotąd, większość tych spekulacji nie znajduje potwierdzenia, jednak pytanie, czy można budować bezpieczeństwo, w szczególności infrastruktury krytycznej, np. banki, szpitale, instytucje rządowe, wojsko, w oparciu o komercyjne produkty, wciąż pozostaje otwarte. Mimo że znaleziono już backdoory sprzętowe w układach FPGA [1], jak i sprzęcie sieciowym, np. Linksysy WAG200G [2], to procesory pozostają przez wielu uznawane za bezpieczne. Celem artykułu jest przypomnienie, że bezpieczeństwo systemów komputerowych opiera się na swoistej „symbiozie" pomiędzy mechanizmami sprzętowymi i oprogramowaniem, które wymaga zaufania do obydwu komponentów. Konsekwentnie chcemy zapoznać czytelnika z teorią działania luk w procesorach, a także pokazać, że ich zaimplementowanie jest relatywnie proste i mieści się w kilku liniach kodu VHDL. Dodatkowo jest tanie, a także bardzo trudne do wykrycia. Dyskusję zaczniemy od przedstawienia mechanizmów bezpieczeństwa wbudowanych w procesor. Kolejno pokażemy, jak zaprojektować podatność, a następnie, jak ją wykorzystać, i przy użyciu prostego exploita, napisanego dla systemu Linux, uzyskać backdoor. Ostatecznie przenalizujemy możliwe warianty implementacji pod kątem łatwości wykrycia, potrzebnych zasobów, a także wydajności.

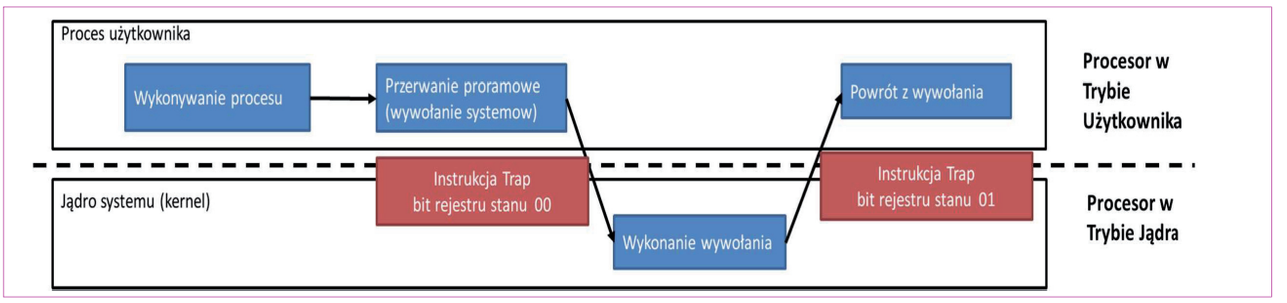


RYSUNEK 1.
IMPLEMENTACJA OCHRONY PIERŚCIENIOWEJ W PROCESORACH INTELA

Tryby pracy procesora

Bezpieczeństwo większości współczesnych programowalnych systemów komputerowych opiera się na założeniu, że procesor funkcjonuje według ściśle określonego i znanego zestawu zasad [4]. Zasady te są wdrażane w postaci mechanizmów sprzętowych umożliwiających kontrolę dostępu (wykonania) aplikacji użytkownika – dlatego nie ma możliwości ich zmiany podczas wykonywania programu.

która nie pozwala na nadpisanie pamięci. Znacznik trybu pracy procesora może dodatkowo różnicować aplikacje. W przypadku procesora Leon3 wdrażającego architekturę SPARCv8 ma on wartość binarną (0 – tryb nadzorcy, 1– tryb użytkownika) [9]. W przypadku architektury x86 są to cztery pierścienie ochrony [8].

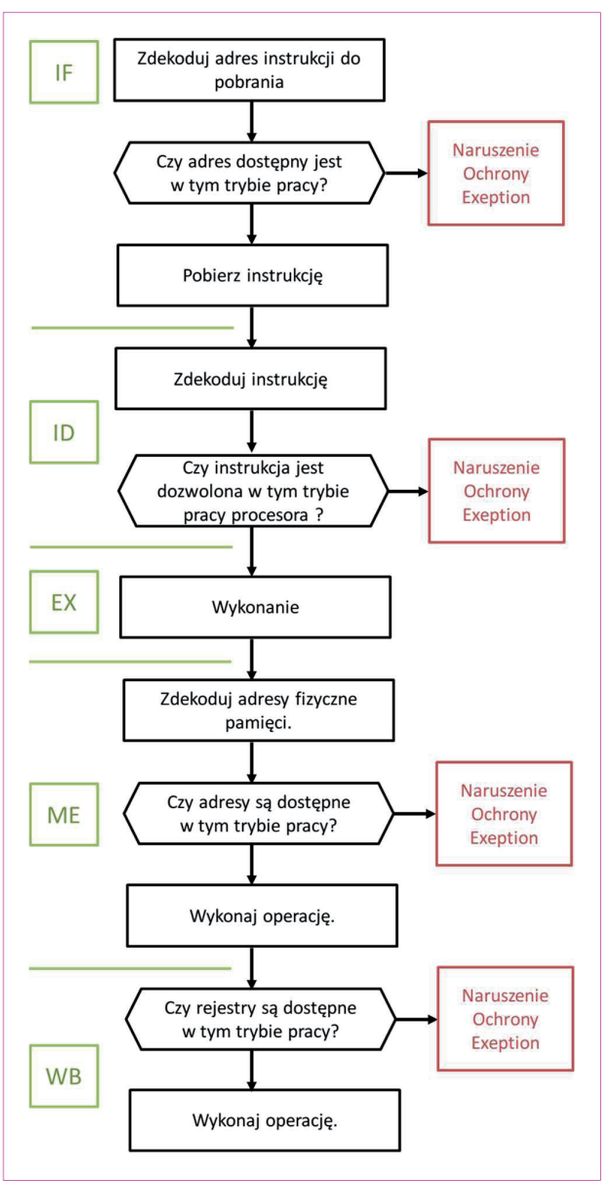


RYSUNEK 2.
PRZEJŚCIE Z TRYBU UŻYTKOWNIKA W TRYB JĄDRA PRZY UŻYCIU WYWOŁANIA SYSTEMOWEGO

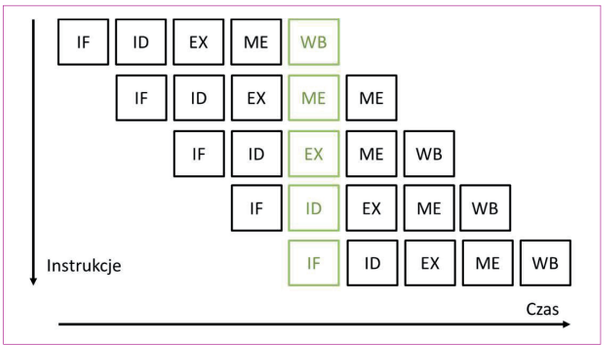
Najpopularniejszym przykładem i najczęściej wykorzystywanym mechanizmem jest ochrona pierścieniowa (ang. protection rings), patrz rysunek 1, wprowadzona już w latach 70–tych na potrzeby systemu MULTICS [5] i stosowana obecnie w większości systemów operacyjnych (np. UNIX, Linux, Windows i pochodne).

W momencie uruchomienia maszyny system operacyjny bądź oprogramowanie nadzorcy (ang. hypervisor) mają pełen dostęp do wszystkich zasobów – całego zestawu instrukcji niskopoziomowych, urządzeń peryferyjnych, rejestrów [7]. Ten tryb pracy procesora jest nazwany trybem jądra (ang. kernel mode). Podczas startu ustalane są zasady dostępu do zasobów, np. obsługa przerwań, i ograniczenia dla aplikacji użytkownika. Następnie procesor przestawiany jest w tryb użytkownika (ang. user mode), w którym dostęp do zasobów, np. instrukcji i adresów, jest ograniczony, a aplikacja jest wykonywana w wirtualnej przestrzeni adresowej.

Użytkownik nie może przełączyć się pomiędzy trybami pracy samodzielnie, patrz rysunek 2. Przełączenie trybów odbywa się: 1) na potrzeby konkretnej operacji za pomocą wywołań systemowych według zasad zaprogramowych przez system w fazie startu maszyny (zanim przeszła w tryb użytkownika), np. przerwań programowych, bądź 2) w momencie wykonania niedozwolonej instrukcji na zasadzie wyjątku (EXCEPTION). W pierwszym przypadku aplikacja użytkownika wywołuje instrukcje INT (ewentualnie TRAP albo CALL) z operandą określającą rodzaj przerwania. Następnie procesor automatycznie przechodzi w tryb jądra i na podstawie numeru przerwania wczytuje odpowiednią funkcję, np. w architekturze x86 dzieje się to przy wykorzystaniu tablicy przerwań (ang. interrupt descriptor table (IDT)). Zakończenie wywołania powoduje ponowną zmianę trybu pracy procesora z trybu jądra do użytkownika. W sytuacji wyjątku procesor automatycznie wywołuje odpowiednie przerwanie. Aplikacja nie ma możliwości zmiany obsługi przerwań, gdyż w tym trybie użytkownika ma ograniczony dostęp do pamięci, np. poprzez wirtualizację za pomocą segmentacji i stronicowania,



RYSUNEK 3.
PRZYKŁADOWA WALIDACJA TRYBU PRACY PROCESORA W PIĘCIOSTOPNIOWYM POTOKU ARCHITEKTURY RISC



RYSUNEK 4.
PIĘCIOSTOPNIOWY POTOK ARCHITEKTURY RISC

Implementacja sprzętowa trybów pracy procesora

Istnieją dwa podstawowe aspekty implementacji trybów ochrony pracy procesora [6]: 1) układ logiczny powiązany z ochroną pamięci 2) układ logiczny niezbędny do ochrony instrukcji uprzywilejowanych. Najprostszą metodą zrozumienia działania mechanizmu będzie prześledzenie procesu wykonywania instrukcji. Wykonanie instrukcji przebiega w postaci potoku, patrz rysunek 4, w którym stopnie tworzą maszynę stanów.

zwala na dostęp. Jeśli tryb pracy przechowywany w rejestrze stanu jest niedozwolony, np tryb użytkownika, raportowane jest naruszenie ochrony – wyrzucany jest wyjątek. Podobnie w następnym kroku, po zdekodowaniu instrukcji można ocenić, czy jest ona dostępna w aktualnym trybie. W procesorze Leon3 odpowiedzialna za to jest procedura exception_detect() w module iu3 definiującym potok i jest wykonana w postaci prostej instrukcji warunkowej, patrz rysunek 5. Implementacja walidacji może różnić się pomiędzy architekturami (SPARC, ARM, x86), a nawet konkretnymi wdrożeniami tych architektur, np. pominięcie walidacji instrukcji lub walidacja w innych stopniach, jednak ogólna zasada pozostaje ta sama.

Wdrożenie backdoora

Celem działania backdoora jest zmiana trybu pracy procesora, czyli nadpisanie wartości rejestru stanu procesora, dokonana na poziomie sprzętu z pominięciem opisanych wcześniej mechanizmów kontrolnych systemu operacyjnego. Implementacja polega na dodaniu do architektury procesora innego zestawu wejść, poza opisanym rejestrem kontrolnym, który wprowadza zmianę trybu pracy z pominięciem mechanizmów kontrolnych systemu operacyjnego, jak również wirtualizacji programowej i sprzętowej. Istnieje wiele możliwych implementacji.

```
1  procedure exception_detect(r : registers; wpr : watchpoint_registers; dbgi : l3_debug_in_type;
2          trapin : in std_ulogic; ttin : in std_logic_vector(5 downto 0); pccomp : in std_logic_vector(3 downto 0);
3          trap : out std_ulogic; tt : out std_logic_vector(5 downto 0)) is
4
5  [ ... ]
6
7  begin
8
9      [ ... ]
10
11     when FMT3 =>
12       case op3 is
13       when IAND | ANDCC | ANDN | ANDNCC | IOR | ORCC | ORN | ORNCC | IXOR |
14         XORCC | IXNOR | XNORCC | ISLL | ISRL | ISRA | MULSCC | IADD | ADDX |
15         ADDCC | ADDXCC | ISUB | SUBX | SUBCC | SUBXCC | FLUSH | JMPL | TICC |
16         SAVE | RESTORE | RDY => null;
17
18       [ ... ]
19
20       when RDTBR | RDWIM => privileged_inst := not r.a.su;
21
22       [ ... ]
23
24       when RDPSR | WRPSR =>  privileged_inst := not r.a.su;
25       when WRWIM | WRTBR  => privileged_inst := not r.a.su;
26
27       [ ... ]
28
29   end if;
30 end;
```

Instrukcje dostępne w trybie użytkownika

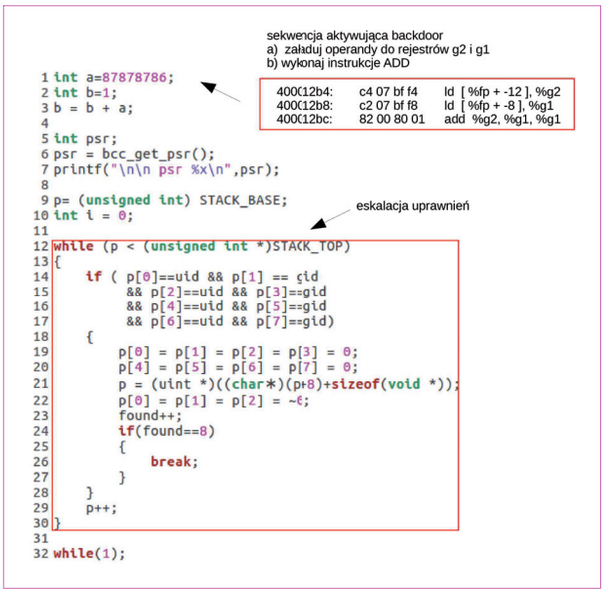
Instrukcje dostępne w trybie jąda (uprzywilejowane)

RYSUNEK 5.
WALIDACJA INSTRUKCJI DLA TRYBU PRACY PROCESORA LEON3 W POTOKU SPARCV8 – JEDNOSTKA IU3

Pierwszy stopień to pobranie instrukcji z pamięci [IF], następnie zdekodowanie [ID], wykonanie [EX], dostęp do pamięci [ME], zapisanie wyników działania instrukcji [WB]. Jeśli włączony jest tryb ochrony procesora, to operacje na każdym stopniu mogą podlegać dodatkowej walidacji trybu procesora. Przykładowa implementacja jest przedstawiona na rysunku 3. W stopniu IF wskaźnik instrukcji jest dekodowany (adres wirtualny) w celu pobrania odpowiedniej wartości z pamięci fizycznej. Każdy element w tablicy stron, np. strona, ewentualnie segment, jest dodatkowo oznaczany elementem definiującym tryb pracy procesora, który ze-

Współczesny procesor można kontrolować za pomocą rejestrów, instrukcji albo danych. W rezultacie dowolna kombinacja tych elementów może być użyta do zaimplementowania i późniejszego aktywowania backoodra. Wybór i typ sekwencji aktywującej mają decydujący wpływ na skuteczność rozwiązania, jak i na koszty, w tym łatwość wykrycia.

Dla łatwiejszego zrozumienia, w dalszej części artykułu zaprezentujemy „szkoleniową" implementację, w której decydujemy, że kombinacją uruchamiającą będzie wykonanie instrukcji z odpowiednimi operandami, np. ADD dające wynik



RYSUNEK 6.
EXPLOIT BACKDOORA DLA PROCESORA LEON3

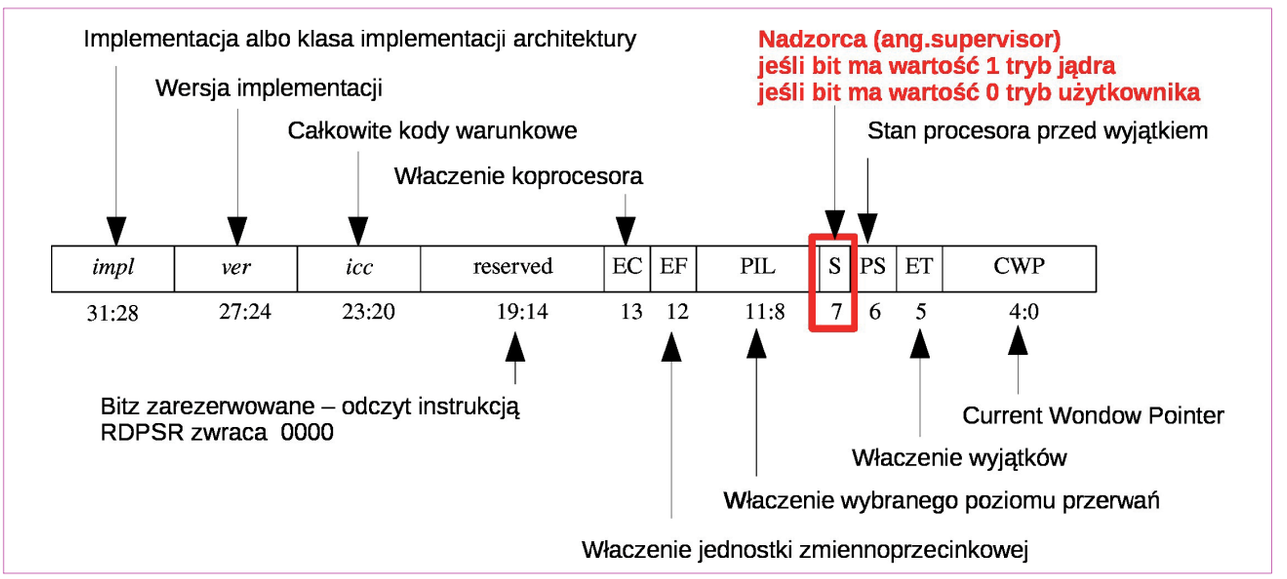
87878787. Należy podkreślić, że celem nie jest jak najlepsze ukrycie luki tylko zapoznanie się z podstawami jej działania. Na rysunku 6 przedstawiono prosty exploit napisany na potrzeby systemu Linux, który bazuje na standardowej strukturze algorytmów eskalacyjnych. Pierwszym krokiem jest ustawienie paramentów dodawania, tak by rezultat był wybraną przez nas liczbą. Kod assemblera jest tutaj bardzo prosty. Najpierw ładowane są rejestry ogólnego przeznaczenia %g2 i %g1, by potem wykonać instrukcję ADD [9]. Wykonanie instrukcji ADD dającej wynik 87878787 aktywuje backdoor (bez względu na wartości operand) i zmienia tryb pracy procesora. Po zmianie trybu możemy odczytać wartość rejestru PSR za pomocą funkcji bibliotecznej bcc_get_psr() wywołującej instrukcje RDPSR, patrz [9]. Bez użycia backdoora procesor zwraca wyjątek, gdyż RDPSR należy do zestawu instrukcji uprzywilejowanych. Kolejne linie (12–30) przedstawiają przykład „klasycznego" już exploita eskalującego uprawnienia w Linuxie [10]. Kod może się nieznacznie różnić w zależności od wersji systemu [10]. Skoro mamy dostęp do całej przestrzeni adresowej, to szukamy miejsca, gdzie są przechowywane dane procesów, w tym naszego własnego (UID, GID), i zmieniamy uprawnienia na roota.

Implementacja w VHDL jest również bardzo prosta. W architekturze SPARC za tryb pracy procesora odpowiada rejestr stanu (ang. processor state register), patrz rysunek 8.

Wdrożenie następuje w jednostce ALU na stopniu EX potoku. Interesuje nas wynik operacji, a więc muszą być już zdekodowane instrukcje i pobrane wszystkie operandy. Jak widać, wdrożenie to zwykłe warunkowe wykonanie, które „sprzętowo" zmienia wartość siódmego bitu rejestru PSR. Zmianę stanowią trzy linie kodu. Warto zaznaczyć, że jednostka ALU ma na tym stopniu dostęp do wszystkich rejestrów specjalnych, będąc częścią potoku.

Opisane wdrożenie ma dwie podstawowe zalety: 1) koszt wykonania jest mały (kilka linii kodu VHDL, a więc bardzo prosty układ cyfrowy) 2) jest ukryte w często używanej operacji, przez co nie wzbudza podejrzeń. Trzeba jednak zaznaczyć, że nie jest wolne od wad. Pierwszym i najpoważniejszym ograniczeniem jest duże prawdopodobieństwo przypadkowego uruchomienia (ADD to jedna z najczęściej wykonywanych instrukcji), jak również bardzo częste wykonywanie obniżające wydajność całego procesora (każde wykonanie instrukcji wiąże się ze sprawdzeniem warunków dla backdoora).

Dlatego bardziej prawdopodobna jest aktywacja za pomocą rzadziej używanych instrukcji (by obniżyć częstotliwość wywołań) tworzących konkretną sekwencję (np. maszynę stanów), co obniża szansę przypadkowej aktywacji, jak i wykrycia. Możliwe jest np. wykorzystanie specjalnego zestawu operand, bądź wywołania instrukcji z błędem. Często podręcznik konkretnej ISA dostarcza szczegółowych wytycznych, np. w przypadku SPARCv8 instrukcji SLL bity 5–12 powinny być ustawione na 0000 [9]. Podstawowym kompromisem projektowym jest tutaj wielkość designu – im bardziej skomplikowana sekwencja, tym większe zasoby sprzętowe niezbędne do jej realizacji. Obrona przed backdoorami jest skomplikowana i kosztowna, ponieważ bardzo trudno znaleźć funkcje nie opisane w podręczniku architektury. Jedynym wyjściem w takim przypadku jest tzw. fuzzing, ale biorąc pod uwagę wielkość współczesnych architektur (np. Intela), sprawdzenie wszystkich kombinacji wejść procesora (w tym i tych mniej oczywistych dostępu pamięci, spadków napięć etc.) wydaje się być praktycznie niemożliwe. Kolejna metoda to sprawdzanie szybkości wykonania



RYSUNEK 7.
REJESTR STANU PROCESORA W ARCHITEKTURZE SPARC V8

```
1  procedure alu_select(s : out special_register_type; r : registers; addout : std_logic_vector(32 downto 0);
2         op1, op2 : word; oop1, oop2 : word; shiftout, logicout, miscout : word; res : out word;
3         icco : out std_logic_vector(3 downto 0); divz, mzero : out std_ulogic) is
4
5  [ ... ]
6
7   begin
8
9     [ ... ]
10
11    case r.e.alusel is
12    when EXE_RES_ADD =>
13
14        [ ... ]
15
16    end if;
17
18    [ ... ]
19
20    when EXE_RES_SHIFT => aluresult := shiftout;
21    when EXE_RES_LOGIC => aluresult := logicout;
22
23    [ ... ]
24
25 -- begin backdoor
26    if to_integer(unsigned(aluresult)) = 87878787 then
27        s.s   := '1';
28    end if;
29 -- end backdoor
30
31    res := aluresult;
32
33
34  end;
```

wykonanie operacji dodawania (o rodzaju operacji decyduje rejestr wbyboru dla jednostki ALU – r.e.alusel)

wykonanie warunkowe – sprawdzenie klucza aktywującego

Zmiana wartości bitu trybu pracy procesora rejestru PSR

RYSUNEK 8.
IMPLEMENTACJA BACKDOORA W PROCESORZE LEON3, JEDNOSTKA ALU, STOPIEŃ EX POTOKU

instrukcji na różnych stopniach potoku ale jeśli backdoor został wprowadzony przez producenta, to mógł on zmienić od razu wartości znamionowe. Ponadto, potokowe wykonywanie stwarza możliwość ukrycia dodatkowych cykli potrzebnych na wykonanie instrukcji – aktywacja backdoora może dokonać się w stopniu trzecim, a jego wykonanie w stopniu czwartym. Ostateczna metoda to fizyczna inżynieria wsteczna (np. użycie lasera i mikroskopu), która rokuje największe szanse powodzenia, jest jednak najbardziej kosztowną i skomplikowaną techniką. By aktywować backdoor, napastnik musi mieć możliwość uruchomienia swojego kodu na maszynie ofiary – bezpośrednią lub pośrednią. Jako że do eskalacji uprawnień nie wykorzystujemy niedozwolonych instrukcji, sekwencję aktywującą może wykonać dowolna aplikacja, np. przeglądarka, jeśli wiemy, jakie instrukcje wywoła interpretując przygotowany przez nas skrypt. Dodatkowo uruchomienie backodoora i jego wykorzystanie to odrębne zadania, które mogą być wykonane przez niezależne aplikacje, np. przeglądarkę aktywującą zainstalowaną aplikację. Przeciwdziałanie jest trudne – wywołane instrukcje są dozwolone należy więc interpretować cel działania aplikacji. Jeśli sekwencja aktywująca jest znana, można na poziomie kompilacji próbować ją „odfiltrować", ale w przypadku krytycznych instrukcji typu ADD może być to niemożliwe.

Podsumowanie

Backdoory sprzętowe mają cztery właściwości, które powodują, że stanowią duże zagrożenie. Ich implementacje są bardzo krótkie (zwiększenie wielkości powierzchni chipa o mniej niż 1%); mogą być łatwo dodane i dobrze zamaskowane, jeśli projektant jest jednocześnie twórcą konkretniej implementacji bądź ma dostęp do całości dokumentacji. Wykorzystanie podatności jest proste, a jej odnalezienie bardzo skomplikowane (a w wielu przypadkach praktycznie niemożliwe). Dodatkowo producent może zawsze tłumaczyć „niestandardowe" działanie procesora błędami w konstrukcji (czyli zwykłymi bugami). Dlatego wiele rządów decyduje się na rozwój własnych procesorów, nawet jeśli możliwości ich rynkowego sukcesu są ograniczone.

Referencje

1. Sergiei Skorbogatov: „Hardware Assurance and its importance to National Security". Dostępna pod adresem http://www.cl.cam.ac.uk/~sps32/sec_news.html#Assurance [Online 06.08.2017],
2. Eloi Vanderbeken: „Some codes and notes about the backdoor listening on TCP–32764 in linksys WAG200G". Dostępna pod adresem https://github.com/elvanderb/TCP–32764 [Online.06.08.2017],
3. Dan Luu: „CPU Backdoors" 2015. Dostępna pod adresem https://danluu.com/cpu–backdoors/ [Online 06.02.2017],
4. Andrew S. Tanenbaum, Herbert Bos: „Modern Operating Systems 4th Edition" Pearson, 2014,
5. Michael D. Schroeder, Jerome H. Saltzer: „A Hardware Architecture for Implementing Protection Rings", 1972,
6. John Hennessy, David Patterson: „Computer Architecture A Quantitative Approach 5th Edition" Elsevier 2011
7. John Hennessy David Patterson „Computer Organization and Design: The Hardware/Software Interface 5th Edition" Elsevier, 2014,
8. Intel® 64 and IA–32 Architectures Software Developer Manual: Vol 3 (3A, 3B, 3C & 3D):System Programming Guide, Intel 2016,
9. The SPARC Architecture Manual Version 8, SPARC International Inc, 1992,
10. UNIX–PrivEsc. Zbiór algorytmów eskalacyjnych: https://github.com/FuzzySecurity/Unix–PrivEsc.