Linux Capabilities

Poćwiartować roota!

Krzysztof Adamski (k@japko.eu)

10 grudnia 2009

POSIX Capabilites

Motywacja.

- W typowym systemie uniksopodobnym (Linux, BSD, etc) występują dwie role użytkowników:

POSIX Capabilites

Motywacja.

- W typowym systemie uniksopodobnym (Linux, BSD, etc) występują dwie role użytkowników:
  - Root, który może wszystko.

POSIX Capabilites

Motywacja.

- W typowym systemie uniksopodobnym (Linux, BSD, etc) występują dwie role użytkowników:
  - Root, który może wszystko.
  - Nie-root, który może niewiele.

POSIX Capabilites

Motywacja.

- W typowym systemie uniksopodobnym (Linux, BSD, etc) występują dwie role użytkowników:
  - Root, który może wszystko.
  - Nie-root, który może niewiele.
- Procesy posiadają dokładnie takie same prawa jak ich właściciele.

POSIX Capabilites

Motywacja.

- W typowym systemie uniksopodobnym (Linux, BSD, etc) występują dwie role użytkowników:
  - Root, który może wszystko.
  - Nie-root, który może niewiele.
- Procesy posiadają dokładnie takie same prawa jak ich właściciele.
- Powoduje to iż wiele procesów, do swojego działania, wymaga uprawnień roota (bit SUID-0, fuuu..)

POSIX Capabilites

Motywacja.

- W typowym systemie uniksopodobnym (Linux, BSD, etc) występują dwie role użytkowników:
  - Root, który może wszystko.
  - Nie-root, który może niewiele.
- Procesy posiadają dokładnie takie same prawa jak ich właściciele.
- Powoduje to iż wiele procesów, do swojego działania, wymaga uprawnień roota (bit SUID-0, fuuu..)
- ZUO!

POSIX Capabilites c.d.

- POSIX Capabilites to pomysł na podzielenie uprawnień roota na wiele oddzielnych, znacznie mniejszych.

POSIX Capabilites c.d.

- POSIX Capabilites to pomysł na podzielenie uprawnień roota na wiele oddzielnych, znacznie mniejszych.
- W Linuksie mechanizm ten (częściowo) był dostępny począwszy od jądra 2.2 (pierwsze patche włączone zostały do jądra 2.1 w 1998 roku).

POSIX Capabilites c.d.

- POSIX Capabilites to pomysł na podzielenie uprawnień roota na wiele oddzielnych, znacznie mniejszych.
- W Linuksie mechanizm ten (częściowo) był dostępny począwszy od jądra 2.2 (pierwsze patche włączone zostały do jądra 2.1 w 1998 roku).
- Był jednak jeden problem, który najtrafniej opisał Jake Edge w swoim artykule rok temu:

POSIX Capabilites c.d.

- POSIX Capabilites to pomysł na podzielenie uprawnień roota na wiele oddzielnych, znacznie mniejszych.
- W Linuksie mechanizm ten (częściowo) był dostępny począwszy od jądra 2.2 (pierwsze patche włączone zostały do jądra 2.1 w 1998 roku).
- Był jednak jeden problem, który najtrafniej opisał Jake Edge w swoim artykule rok temu:
- *"Capabilities are an interesting, but complicated, security feature. For most of the ten years they have been part of the Linux kernel, they have either been broken, ignored, or both."*

POSIX Capabilites c.d.

- POSIX Capabilites to pomysł na podzielenie uprawnień roota na wiele oddzielnych, znacznie mniejszych.
- W Linuksie mechanizm ten (częściowo) był dostępny począwszy od jądra 2.2 (pierwsze patche włączone zostały do jądra 2.1 w 1998 roku).
- Był jednak jeden problem, który najtrafniej opisał Jake Edge w swoim artykule rok temu:
- *"Capabilities are an interesting, but complicated, security feature. For most of the ten years they have been part of the Linux kernel, they have either been broken, ignored, or both."*
- Na szczęście nie jest to już prawdą!

Linux Capabilites

- Począwszy od jądra 2.6.24, Linux posiada pełną i wreszcie działającą obsługę POSIX Capabilities.

Linux Capabilites

- Począwszy od jądra 2.6.24, Linux posiada pełną i wreszcie działającą obsługę POSIX Capabilities.
- Obecnie zdefiniowane są 34 uprawnienia z możliwością wprowadzenia dodatkowych.

Linux Capabilites

- Począwszy od jądra 2.6.24, Linux posiada pełną i wreszcie działającą obsługę POSIX Capabilities.
- Obecnie zdefiniowane są 34 uprawnienia z możliwością wprowadzenia dodatkowych.
- Mechanizm zaprojektowany jest tak aby możliwa była bezproblemowa koegzystencja zarówno tradycyjnych aplikacji (SUID-0) jak i tych korzystających z tego nowego mechanizmu bezpieczeństwa.

Linux Capabilites

- Począwszy od jądra 2.6.24, Linux posiada pełną i wreszcie działającą obsługę POSIX Capabilities.
- Obecnie zdefiniowane są 34 uprawnienia z możliwością wprowadzenia dodatkowych.
- Mechanizm zaprojektowany jest tak aby możliwa była bezproblemowa koegzystencja zarówno tradycyjnych aplikacji (SUID-0) jak i tych korzystających z tego nowego mechanizmu bezpieczeństwa.
- Wszystkie informacje są indywidualne dla każdego wątku w systemie.

Linux Capabilites

- Począwszy od jądra 2.6.24, Linux posiada pełną i wreszcie działającą obsługę POSIX Capabilities.
- Obecnie zdefiniowane są 34 uprawnienia z możliwością wprowadzenia dodatkowych.
- Mechanizm zaprojektowany jest tak aby możliwa była bezproblemowa koegzystencja zarówno tradycyjnych aplikacji (SUID-0) jak i tych korzystających z tego nowego mechanizmu bezpieczeństwa.
- Wszystkie informacje są indywidualne dla każdego wątku w systemie.
- Uprawnienia są "przeliczane" podczas wykonywania funkcji z rodziny *exec*()*, a nie *fork()*.

Zbiory uprawnień.

- Uprawnienia procesu(wątku) w PCaps nie są jednym zbiorem a czterema:

Zbiory uprawnień.

- Uprawnienia procesu(wątku) w PCaps nie są jednym zbiorem a czterema:
  - *Permited set (pP)*, jest zbiorem definiującym z jakich uprawnień proces MOŻE skorzystać.

Zbiory uprawnień.

- Uprawnienia procesu(wątku) w PCaps nie są jednym zbiorem a czterema:
  - *Permited set (pP)*, jest zbiorem definiującym z jakich uprawnień proces MOŻE skorzystać.
  - *Effective set (pE)*, jest zbiorem definiującym jakie uprawnienia proces aktualnie posiada.

Zbiory uprawnień.

- Uprawnienia procesu(wątku) w PCaps nie są jednym zbiorem a czterema:
  - *Permited set (pP)*, jest zbiorem definiującym z jakich uprawnień proces MOŻE skorzystać.
  - *Effective set (pE)*, jest zbiorem definiującym jakie uprawnienia proces aktualnie posiada.
  - *Inheritable set (pI)*, jest zbiorem definiującym uprawnienia "dziedziczone" (czyli takie, które przetrwają *exec()*).

Zbiory uprawnień.

- Uprawnienia procesu(wątku) w PCaps nie są jednym zbiorem a czterema:
  - *Permited set (pP)*, jest zbiorem definiującym z jakich uprawnień proces MOŻE skorzystać.
  - *Effective set (pE)*, jest zbiorem definiującym jakie uprawnienia proces aktualnie posiada.
  - *Inheritable set (pI)*, jest zbiorem definiującym uprawnienia "dziedziczone" (czyli takie, które przetrwają *exec()*).
  - *Bounding set (X)*, jest zbiorem specjalnym będącym nadrzędnym ograniczeniem.

Zbiory uprawnień.

- Uprawnienia procesu(wątku) w PCaps nie są jednym zbiorem a czterema:
  - *Permited set (pP)*, jest zbiorem definiującym z jakich uprawnień proces MOŻE skorzystać.
  - *Effective set (pE)*, jest zbiorem definiującym jakie uprawnienia proces aktualnie posiada.
  - *Inheritable set (pI)*, jest zbiorem definiującym uprawnienia "dziedziczone" (czyli takie, które przetrwają *exec()*).
  - *Bounding set (X)*, jest zbiorem specjalnym będącym nadrzędnym ograniczeniem.
- Wątek może dowolnie manipulować zbiorami *pP*,*pE* oraz *pI* pod warunkiem, że zachowane są poniższe zależności:

Zbiory uprawnień.

- Uprawnienia procesu(wątku) w PCaps nie są jednym zbiorem a czterema:
  - *Permited set (pP)*, jest zbiorem definiującym z jakich uprawnień proces MOŻE skorzystać.
  - *Effective set (pE)*, jest zbiorem definiującym jakie uprawnienia proces aktualnie posiada.
  - *Inheritable set (pI)*, jest zbiorem definiującym uprawnienia "dziedziczone" (czyli takie, które przetrwają *exec()*).
  - *Bounding set (X)*, jest zbiorem specjalnym będącym nadrzędnym ograniczeniem.
- Wątek może dowolnie manipulować zbiorami *pP*,*pE* oraz *pI* pod warunkiem, że zachowane są poniższe zależności:
  - Do zbioru *pP* nie można dodać nowych uprawnień, można je jedynie zabrać.

Zbiory uprawnień.

- Uprawnienia procesu(wątku) w PCaps nie są jednym zbiorem a czterema:
  - *Permited set (pP)*, jest zbiorem definiującym z jakich uprawnień proces MOŻE skorzystać.
  - *Effective set (pE)*, jest zbiorem definiującym jakie uprawnienia proces aktualnie posiada.
  - *Inheritable set (pI)*, jest zbiorem definiującym uprawnienia "dziedziczone" (czyli takie, które przetrwają *exec()*).
  - *Bounding set (X)*, jest zbiorem specjalnym będącym nadrzędnym ograniczeniem.
- Wątek może dowolnie manipulować zbiorami *pP*,*pE* oraz *pI* pod warunkiem, że zachowane są poniższe zależności:
  - Do zbioru *pP* nie można dodać nowych uprawnień, można je jedynie zabrać.
  - Zbiory *pE* oraz *pI* są podzbiorami zbioru *pP* (jest wyjątek związany z *X*).

Zbiory uprawnień c.d.

- Oprócz zbiorów uprawnień przypisanym wątkom, istnieją również uprawnienia przypisane plikom wykonywalnym:

Zbiory uprawnień c.d.

- Oprócz zbiorów uprawnień przypisanym wątkom, istnieją również uprawnienia przypisane plikom wykonywalnym:
  - *Permited set (fP)*, czasem nazywany zbiorem wymuszonym.

Zbiory uprawnień c.d.

- Oprócz zbiorów uprawnień przypisanym wątkom, istnieją również uprawnienia przypisane plikom wykonywalnym:
  - *Permited set (fP)*, czasem nazywany zbiorem wymuszonym.
  - *Effective set (fE)*, właściwie niezupełnie jest zbiorem a raczej wartością logiczną. Nie można powiem nadać go jedynie dla pojedycznych uprawnień. Przypomnijcie mi żebym to zaraz wyjaśnił...

Zbiory uprawnień c.d.

- Oprócz zbiorów uprawnień przypisanym wątkom, istnieją również uprawnienia przypisane plikom wykonywalnym:
  - *Permited set (fP)*, czasem nazywany zbiorem wymuszonym.
  - *Effective set (fE)*, właściwie niezupełnie jest zbiorem a raczej wartością logiczną. Nie można powiem nadać go jedynie dla pojedycznych uprawnień. Przypomnijcie mi żebym to zaraz wyjaśnił...
  - *Inheritable set (fI)*, dodaje się do zbioru *pI*.

Zbiory uprawnień c.d.

- Oprócz zbiorów uprawnień przypisanym wątkom, istnieją również uprawnienia przypisane plikom wykonywalnym:
  - *Permited set (fP)*, czasem nazywany zbiorem wymuszonym.
  - *Effective set (fE)*, właściwie niezupełnie jest zbiorem a raczej wartością logiczną. Nie można powiem nadać go jedynie dla pojedycznych uprawnień. Przypomnijcie mi żebym to zaraz wyjaśnił...
  - *Inheritable set (fI)*, dodaje się do zbioru *pI*.
- Uprawnienia przypisane plikom przechowywane są w postaci *extended attributes*. Nie wszystkie systemy plików to umożliwiają (np. NFS nie).

Reguły przekształceń.

- Podczas wykonania funkcji *exec()* przeliczane są uprawniena procesu wg poniższych reguł:
  - $pI' = pI$

Reguły przekształceń.

- Podczas wykonania funkcji *exec()* przeliczane są uprawniena procesu wg poniższych reguł:
  - $pI' = pI$
  - $pP' = (pI \ \& \ fI) \mid (fP \ \& \ X)$

Reguły przekształceń.

- Podczas wykonania funkcji *exec()* przeliczane są uprawniena procesu wg poniższych reguł:
  - $pI' = pI$
  - $pP' = (pI \ \& \ fI) \mid (fP \ \& \ X)$
  - $pE' = fE \ ? \ pP' \ : \ 0$

Reguły przekształceń.

- Podczas wykonania funkcji *exec()* przeliczane są uprawniena procesu wg poniższych reguł:
  - $pI' = pI$
  - $pP' = (pI \ \& \ fI) \mid (fP \ \& \ X)$
  - $pE' = fE \ ? \ pP' \ : \ 0$
- Dla zachowania kompatybilności wstecznej, wprowadzono dwie dodatkowe reguły:

Reguły przekształceń.

- Podczas wykonania funkcji *exec()* przeliczane są uprawniena procesu wg poniższych reguł:
  - $pI' = pI$
  - $pP' = (pI \ \& \ fI) \mid (fP \ \& \ X)$
  - $pE' = fE \ ? \ pP' \ : \ 0$
- Dla zachowania kompatybilności wstecznej, wprowadzono dwie dodatkowe reguły:
  - Jeśli program ma ustawiony SUID-0 lub UID=0, wtedy $fI = fP = \neg 0$.

Reguły przekształceń.

- Podczas wykonania funkcji *exec()* przeliczane są uprawniena procesu wg poniższych reguł:
  - $pI' = pI$
  - $pP' = (pI \ \& \ fI) \mid (fP \ \& \ X)$
  - $pE' = fE \ ? \ pP' \ : \ 0$
- Dla zachowania kompatybilności wstecznej, wprowadzono dwie dodatkowe reguły:
  - Jeśli program ma ustawiony SUID-0 lub UID=0, wtedy $fI = fP = \neg 0$.
  - Jeśli program ma ustawiony SUID-0 lub EUID=0, wtedy $fE = 1$.

Zbiór X.

- Jak widać z reguł podanych wcześniej, zbiór X ogranicza uprawnienia, które można nadać procesowi przez uprawnienia przypisane do plików.

Zbiór X.

- Jak widać z reguł podanych wcześniej, zbiór X ogranicza uprawnienia, które można nadać procesowi przez uprawnienia przypisane do plików.
- W starszych wersjach kernela był to parametr globalny dla całego systemu. Począwszy od 2.6.25, każdy wątek ma swój zbiór X. Jest on dziedziczony podczas forkowania i pozostaje niezmieniony po wywołaniu *exec()*.

Zbiór X.

- Jak widać z reguł podanych wcześniej, zbiór X ogranicza uprawnienia, które można nadać procesowi przez uprawnienia przypisane do plików.
- W starszych wersjach kernela był to parametr globalny dla całego systemu. Począwszy od 2.6.25, każdy wątek ma swój zbiór X. Jest on dziedziczony podczas forkowania i pozostaje niezmieniony po wywołaniu *exec()*.
- Do zbioru X oczywiście nie można dodawać uprawnień, jedynie je stamtąd zabierać.

Zbiór X.

- Jak widać z reguł podanych wcześniej, zbiór X ogranicza uprawnienia, które można nadać procesowi przez uprawnienia przypisane do plików.
- W starszych wersjach kernela był to parametr globalny dla całego systemu. Począwszy od 2.6.25, każdy wątek ma swój zbiór X. Jest on dziedziczony podczas forkowania i pozostaje niezmieniony po wywołaniu *exec()*.
- Do zbioru X oczywiście nie można dodawać uprawnień, jedynie je stamtąd zabierać.
- Usunięcie uprawnień ze zbioru X nie powoduje usunięcia ich ze zbioru *pI*!

Efekty zmian UID

- W celu uzyskania kompatybilności wstecznej, wprowadzono następuje reguły poczas zmian UID pomiędzy 0 a >0:

Efekty zmian UID

- W celu uzyskania kompatybilności wstecznej, wprowadzono następuje reguły poczas zmian UID pomiędzy 0 a >0:
  - Jeśli UID=0 lub EUID=0 lub SUID=0 przed zmianą, natomiast po niej (UID=EUID=SUID)!=0, wtedy $pP = pE = 0$.

Efekty zmian UID

- W celu uzyskania kompatybilności wstecznej, wprowadzono następuje reguły poczas zmian UID pomiędzy 0 a >0:
  - Jeśli UID=0 lub EUID=0 lub SUID=0 przed zmianą, natomiast po niej (UID=EUID=SUID)!=0, wtedy $pP = pE = 0$.
  - Jeśli EUID=0 przez zmianą, natomiast po niej EUID!=0, wtedy $pE = 0$.

Efekty zmian UID

- W celu uzyskania kompatybilności wstecznej, wprowadzono następuje reguły poczas zmian UID pomiędzy 0 a >0:
  - Jeśli UID=0 lub EUID=0 lub SUID=0 przed zmianą, natomiast po niej (UID=EUID=SUID)!=0, wtedy $pP = pE = 0$.
  - Jeśli EUID=0 przez zmianą, natomiast po niej EUID!=0, wtedy $pE = 0$.
  - Jeśli EUID!=0 przed zmianą, natomiast po niej EUID=0, wtedy $pE = pP$.

Efekty zmian UID

- W celu uzyskania kompatybilności wstecznej, wprowadzono następuje reguły poczas zmian UID pomiędzy 0 a >0:
  - Jeśli UID=0 lub EUID=0 lub SUID=0 przed zmianą, natomiast po niej (UID=EUID=SUID)!=0, wtedy $pP = pE = 0$.
  - Jeśli EUID=0 przez zmianą, natomiast po niej EUID!=0, wtedy $pE = 0$.
  - Jeśli EUID!=0 przed zmianą, natomiast po niej EUID=0, wtedy $pE = pP$.
- Można zapobiec tym zmianom przez użycie specjalnego wywołania sytemowego (lub securebits, o czym za chwilę).

Efekty zmian UID

- W celu uzyskania kompatybilności wstecznej, wprowadzono następuje reguły poczas zmian UID pomiędzy 0 a >0:
  - Jeśli UID=0 lub EUID=0 lub SUID=0 przed zmianą, natomiast po niej (UID=EUID=SUID)!=0, wtedy $pP = pE = 0$.
  - Jeśli EUID=0 przez zmianą, natomiast po niej EUID!=0, wtedy $pE = 0$.
  - Jeśli EUID!=0 przed zmianą, natomiast po niej EUID=0, wtedy $pE = pP$.
- Można zapobiec tym zmianom przez użycie specjalnego wywołania sytemowego (lub securebits, o czym za chwilę).
- Dodatkowo usuwane są niektóre uprawnienia przypisane podczas zmiany właściciela jego właściciela pliku z UID=0 na UID!=0.

Securebits

- Wprowadzone w jądze 2.6.26.

Securebits

- Wprowadzone w jądze 2.6.26.
- Są indywidualne dla każdego wątku.

Securebits

- Wprowadzone w jądze 2.6.26.
- Są indywidualne dla każdego wątku.
- Umożliwiają zmianę (wyłączenie) specjalnego traktowania roota i poleganie jedynie na PCaps.

Securebits

- Wprowadzone w jądze 2.6.26.
- Są indywidualne dla każdego wątku.
- Umożliwiają zmianę (wyłączenie) specjalnego traktowania roota i poleganie jedynie na PCaps.
- Możliwe są trzy wartości:

Securebits

- Wprowadzone w jądze 2.6.26.
- Są indywidualne dla każdego wątku.
- Umożliwiają zmianę (wyłączenie) specjalnego traktowania roota i poleganie jedynie na PCaps.
- Możliwe są trzy wartości:
  - **SECURE_KEEP_CAPS**, zapobiega zerowaniu uprawnień przy zmianie na (UID=EUID=SUID)!=0.

Securebits

- Wprowadzone w jądze 2.6.26.
- Są indywidualne dla każdego wątku.
- Umożliwiają zmianę (wyłączenie) specjalnego traktowania roota i poleganie jedynie na PCaps.
- Możliwe są trzy wartości:
  - **SECURE_KEEP_CAPS**, zapobiega zerowaniu uprawnień przy zmianie na (UID=EUID=SUID)!=0.
  - **SECURE_NO_SETUID_FIXUP**, zapobiega wszystkim zmianom opisanym na poprzednim slajdzie.

Securebits

- Wprowadzone w jądze 2.6.26.
- Są indywidualne dla każdego wątku.
- Umożliwiają zmianę (wyłączenie) specjalnego traktowania roota i poleganie jedynie na PCaps.
- Możliwe są trzy wartości:
  - **SECURE_KEEP_CAPS**, zapobiega zerowaniu uprawnień przy zmianie na (UID=EUID=SUID)!=0.
  - **SECURE_NO_SETUID_FIXUP**, zapobiega wszystkim zmianom opisanym na poprzednim slajdzie.
  - **SECURE_NOROOT**, root traktowany jest jak zwykły użytkownik i nie ma żadnych dodatkowych uprawnień.

Securebits

- Wprowadzone w jądze 2.6.26.
- Są indywidualne dla każdego wątku.
- Umożliwiają zmianę (wyłączenie) specjalnego traktowania roota i poleganie jedynie na PCaps.
- Możliwe są trzy wartości:
  - **SECURE_KEEP_CAPS**, zapobiega zerowaniu uprawnień przy zmianie na (UID=EUID=SUID)!=0.
  - **SECURE_NO_SETUID_FIXUP**, zapobiega wszystkim zmianom opisanym na poprzednim slajdzie.
  - **SECURE_NOROOT**, root traktowany jest jak zwykły użytkownik i nie ma żadnych dodatkowych uprawnień.
- Możliwe jest zablokowanie wprowadzania dalszych zmian w każdej z tych flag.

Sposoby wykorzystania.

Istnieją dwa główne sposoby na skorzystanie z możliwości Linux Capabilities:

- Zastąpienie SUID-0 przez nadanie uprawnień przypisanych do pików wykonywalnych:

Sposoby wykorzystania.

Istnieją dwa główne sposoby na skorzystanie z możliwości Linux Capabilities:

- Zastąpienie SUID-0 przez nadanie uprawnień przypisanych do pików wykonywalnych:
  - Zgodne z ideą POSIX capabilities.

Sposoby wykorzystania.

Istnieją dwa główne sposoby na skorzystanie z możliwości Linux Capabilities:

- Zastąpienie SUID-0 przez nadanie uprawnień przypisanych do pików wykonywalnych:
  - Zgodne z ideą POSIX capabilities.
  - Dzięki zastosowaniu zbioru *fE*, często nie trzeba zmieniać kodu aplikacji.

Sposoby wykorzystania.

Istnieją dwa główne sposoby na skorzystanie z możliwości Linux Capabilities:

- Zastąpienie SUID-0 przez nadanie uprawnień przypisanych do pików wykonywalnych:
  - Zgodne z ideą POSIX capabilities.
  - Dzięki zastosowaniu zbioru *fE*, często nie trzeba zmieniać kodu aplikacji.
  - Administrator decyduje o tym jakie uprawnienia otrzyma proces.

Sposoby wykorzystania.

Istnieją dwa główne sposoby na skorzystanie z możliwości Linux Capabilities:

- Zastąpienie SUID-0 przez nadanie uprawnień przypisanych do pików wykonywalnych:
  - Zgodne z ideą POSIX capabilities.
  - Dzięki zastosowaniu zbioru *fE*, często nie trzeba zmieniać kodu aplikacji.
  - Administrator decyduje o tym jakie uprawnienia otrzyma proces.
  - Czasem niemożliwe do osiągnięcia.

Sposoby wykorzystania.

Istnieją dwa główne sposoby na skorzystanie z możliwości Linux Capabilities:

- Zastąpienie SUID-0 przez nadanie uprawnień przypisanych do pików wykonywalnych:
  - Zgodne z ideą POSIX capabilities.
  - Dzięki zastosowaniu zbioru *fE*, często nie trzeba zmieniać kodu aplikacji.
  - Administrator decyduje o tym jakie uprawnienia otrzyma proces.
  - Czasem niemożliwe do osiągnięcia.
- Ograniczenie uprawnień użytkownika root.

Sposoby wykorzystania.

Istnieją dwa główne sposoby na skorzystanie z możliwości Linux Capabilities:

- Zastąpienie SUID-0 przez nadanie uprawnień przypisanych do pików wykonywalnych:
  - Zgodne z ideą POSIX capabilities.
  - Dzięki zastosowaniu zbioru *fE*, często nie trzeba zmieniać kodu aplikacji.
  - Administrator decyduje o tym jakie uprawnienia otrzyma proces.
  - Czasem niemożliwe do osiągnięcia.
- Ograniczenie uprawnień użytkownika root.
  - Przez długi czas był to jedyny sposób wykorzystania Capabilities w Linuksie.

Sposoby wykorzystania.

Istnieją dwa główne sposoby na skorzystanie z możliwości Linux Capabilities:

- Zastąpienie SUID-0 przez nadanie uprawnień przypisanych do pików wykonywalnych:
  - Zgodne z ideą POSIX capabilities.
  - Dzięki zastosowaniu zbioru *fE*, często nie trzeba zmieniać kodu aplikacji.
  - Administrator decyduje o tym jakie uprawnienia otrzyma proces.
  - Czasem niemożliwe do osiągnięcia.
- Ograniczenie uprawnień użytkownika root.
  - Przez długi czas był to jedyny sposób wykorzystania Capabilities w Linuksie.
  - Aplikacja musi zostać zmodyfikowana aby było to możliwe.

Sposoby wykorzystania.

Istnieją dwa główne sposoby na skorzystanie z możliwości Linux Capabilities:

- Zastąpienie SUID-0 przez nadanie uprawnień przypisanych do pików wykonywalnych:
  - Zgodne z ideą POSIX capabilities.
  - Dzięki zastosowaniu zbioru *fE*, często nie trzeba zmieniać kodu aplikacji.
  - Administrator decyduje o tym jakie uprawnienia otrzyma proces.
  - Czasem niemożliwe do osiągnięcia.
- Ograniczenie uprawnień użytkownika root.
  - Przez długi czas był to jedyny sposób wykorzystania Capabilities w Linuksie.
  - Aplikacja musi zostać zmodyfikowana aby było to możliwe.
  - Użycie FCaps+Securebits lub "ręczne" ustawianie uprawnień za pomocą libcap(-ng).

Przykład 1, ping

- /bin/ping posiada najczęściej ustawiony bit SUID-0.

Przykład 1, ping

- /bin/ping posiada najczęściej ustawiony bit SUID-0.
- Jeśli ten bit usuniemy otrzymamy błąd *ping: icmp open socket: Operacja niedozwolona*.

Przykład 1, ping

- /bin/ping posiada najczęściej ustawiony bit SUID-0.
- Jeśli ten bit usuniemy otrzymamy błąd *ping: icmp open socket: Operacja niedozwolona*.
- Ping wymaga uprawnień do otworzenia gniazda w trybie RAW.

Przykład 1, ping

- /bin/ping posiada najczęściej ustawiony bit SUID-0.
- Jeśli ten bit usuniemy otrzymamy błąd *ping: icmp open socket: Operacja niedozwolona*.
- Ping wymaga uprawnień do otworzenia gniazda w trybie RAW.
- Zamiast SUID-0 wystarczy więc przypisać plikowi uprawnienia dla (CAP_NET_RAW).

Przykład 1, ping

- /bin/ping posiada najczęściej ustawiony bit SUID-0.
- Jeśli ten bit usuniemy otrzymamy błąd *ping: icmp open socket: Operacja niedozwolona*.
- Ping wymaga uprawnień do otworzenia gniazda w trybie RAW.
- Zamiast SUID-0 wystarczy więc przypisać plikowi uprawnienia dla (CAP_NET_RAW).
- Sprawdzmy to...

Przykład 1, c.d.

- Ping to dość prosty program ale co z innymi?

Przykład 1, c.d.

- Ping to dość prosty program ale co z innymi?
- Ustawienia zapisane w xattr przypisane są do inoda. W czasie aktualizacji systemu zostana utracone. Jak sobie z tym radzić?

Przykład 1, c.d.

- Ping to dość prosty program ale co z innymi?
- Ustawienia zapisane w xattr przypisane są do inoda. W czasie aktualizacji systemu zostana utracone. Jak sobie z tym radzić?
- Skąd wiedzieć jakie uprawnienia nadać aplikacjom? Trzy metody - strace, SystemTap, kprobes.

Przykład 1, c.d.

- Ping to dość prosty program ale co z innymi?
- Ustawienia zapisane w xattr przypisane są do inoda. W czasie aktualizacji systemu zostana utracone. Jak sobie z tym radzić?
- Skąd wiedzieć jakie uprawnienia nadać aplikacjom? Trzy metody - strace, SystemTap, kprobes.
- Uwaga - czasem nadanie zbyt małej ilości uprawnień może być groźne - Sendmail Capabilities Bug.

Przykład 2, libcap

- Pierwsza i główna biblioteka do zarządzania Capabilities w Linuksie.

Przykład 2, libcap

- Pierwsza i główna biblioteka do zarządzania Capabilities w Linuksie.
- *cap_from_text()*, *cap_to_text()*, *cap_clear()*, *cap_get_proc()*, *cap_set_proc()*, *cap_get_file()*, *cap_set_file()*, ...

Przykład 2, libcap

- Pierwsza i główna biblioteka do zarządzania Capabilities w Linuksie.
- *cap_from_text()*, *cap_to_text()*, *cap_clear()*, *cap_get_proc()*, *cap_set_proc()*, *cap_get_file()*, *cap_set_file()*, ...
- Dość łatwo jest sprawdzić jakie mamy uprawnienia, ustawić je zgodnie z podaną specyfikacją w postaci tekstowej itp, trudniej jednak operować na pojedynczych uprawnieniach, sterować zachowaniem podczas dziedzicenia czy ustawiać zbiór X.

Przykład 2, libcap

- Pierwsza i główna biblioteka do zarządzania Capabilities w Linuksie.
- *cap_from_text()*, *cap_to_text()*, *cap_clear()*, *cap_get_proc()*, *cap_set_proc()*, *cap_get_file()*, *cap_set_file()*, ...
- Dość łatwo jest sprawdzić jakie mamy uprawnienia, ustawić je zgodnie z podaną specyfikacją w postaci tekstowej itp, trudniej jednak operować na pojedynczych uprawnieniach, sterować zachowaniem podczas dziedzicenia czy ustawiać zbiór X.
- Pakiet dostarcza podstawowych narzędzi takich jak: *setcap*, *getcap*, *getpcap*, *capsh*.

Przykład 2, libcap-ng

- Napisany z myślą o znaczym ułatwieniu operowania na Capabilities, w szczególności by poprawić wspomniane słabości libcap.

Przykład 2, libcap-ng

- Napisany z myślą o znaczym ułatwieniu operowania na Capabilities, w szczególności by poprawić wspomniane słabości libcap.
- Większość typowych operacji wykonuje się tu w 2-3 liniach kodu, np:

Przykład 2, libcap-ng

- Napisany z myślą o znaczym ułatwieniu operowania na Capabilities, w szczególności by poprawić wspomniane słabości libcap.
- Większość typowych operacji wykonuje się tu w 2-3 liniach kodu, np:
  - Zrzucenie wszystkich uprawnień:

```
capng_clear(CAPNG_SELECT_BOTH);
capng_apply(CAPNG_SELECT_BOTH);
```

Przykład 2, libcap-ng

- Napisany z myślą o znaczym ułatwieniu operowania na Capabilities, w szczególności by poprawić wspomniane słabości libcap.
- Większość typowych operacji wykonuje się tu w 2-3 liniach kodu, np:
  - Zrzucenie wszystkich uprawnień:
    ```
    capng_clear(CAPNG_SELECT_BOTH);
    capng_apply(CAPNG_SELECT_BOTH);
    ```
  - Ustawienie tylko kilku uprawnień:
    ```
    capng_clear(CAPNG_SELECT_BOTH);
    capng_updatev(CAPNG_ADD, CAPNG_EFFECTIVE|CAPNG_PERMITTED,
    CAP_SETUID, CAP_SETGID, -1);
    capng_apply(CAPNG_SELECT_BOTH);
    ```
- Istnieją dobre bindingi do pythona.

Przykład 2, libcap-ng

- Napisany z myślą o znaczym ułatwieniu operowania na Capabilities, w szczególności by poprawić wspomniane słabości libcap.
- Większość typowych operacji wykonuje się tu w 2-3 liniach kodu, np:
  - Zrzucenie wszystkich uprawnień:
    ```
    capng_clear(CAPNG_SELECT_BOTH);
    capng_apply(CAPNG_SELECT_BOTH);
    ```
  - Ustawienie tylko kilku uprawnień:
    ```
    capng_clear(CAPNG_SELECT_BOTH);
    capng_updatev(CAPNG_ADD, CAPNG_EFFECTIVE|CAPNG_PERMITTED,
    CAP_SETUID, CAP_SETGID, -1);
    capng_apply(CAPNG_SELECT_BOTH);
    ```
- Istnieją dobre bindingi do pythona.
- Pakiet dostarcza dodatkowych narzędzi takich jak: *pscap*, *netcap*, *filecap*.

Przykład 3, DAC_OVERRIDE

- Domyślnie większość plików i katalogów w systemie ma ustawione prawa do zapisu i odczytu dla roota (choć nie są potrzebne bo root ma CAP_DAC_OVERRIDE).

Przykład 3, DAC_OVERRIDE

- Domyślnie większość plików i katalogów w systemie ma ustawione prawa do zapisu i odczytu dla roota (choć nie są potrzebne bo root ma CAP_DAC_OVERRIDE).
- Jeśli ktoś zdobędzie EUID=0, nie potrzebuje mieć żadnych dodatkowych uprawnień żeby namieszać nam w systemie.

Przykład 3, DAC_OVERRIDE

- Domyślnie większość plików i katalogów w systemie ma ustawione prawa do zapisu i odczytu dla roota (choć nie są potrzebne bo root ma CAP_DAC_OVERRIDE).
- Jeśli ktoś zdobędzie EUID=0, nie potrzebuje mieć żadnych dodatkowych uprawnień żeby namieszać nam w systemie.
- Pomysł: zabrać CAP_DAC_OVERRIDE wszystkim, którzy go nie wymagają (szczególnie usługom sieciowym).

Przykład 3, DAC_OVERRIDE

- Domyślnie większość plików i katalogów w systemie ma ustawione prawa do zapisu i odczytu dla roota (choć nie są potrzebne bo root ma CAP_DAC_OVERRIDE).
- Jeśli ktoś zdobędzie EUID=0, nie potrzebuje mieć żadnych dodatkowych uprawnień żeby namieszać nam w systemie.
- Pomysł: zabrać CAP_DAC_OVERRIDE wszystkim, którzy go nie wymagają (szczególnie usługom sieciowym).
- Kompatybilność z oczekiwanym zachowaniem nie zmieni się - zalogowany root i tak ma wszystkie prawa jednak utrudni to działanie napastnikom.

Przykład 3, DAC_OVERRIDE

- Domyślnie większość plików i katalogów w systemie ma ustawione prawa do zapisu i odczytu dla roota (choć nie są potrzebne bo root ma CAP_DAC_OVERRIDE).
- Jeśli ktoś zdobędzie EUID=0, nie potrzebuje mieć żadnych dodatkowych uprawnień żeby namieszać nam w systemie.
- Pomysł: zabrać CAP_DAC_OVERRIDE wszystkim, którzy go nie wymagają (szczególnie usługom sieciowym).
- Kompatybilność z oczekiwanym zachowaniem nie zmieni się - zalogowany root i tak ma wszystkie prawa jednak utrudni to działanie napastnikom.
- Zostało z powodzeniem wprowadzone w Fedora 12, większość katalogów systemowych ma tu uprawnienia 555 a ważne pliki (jak /etc/passwd) mają często nawet 000. I wszystko działa!

Tzw. Sendmail Capabilities Bug

- Sendmail uruchamiany był jako aplikacja SUID-0.

Tzw. Sendmail Capabilities Bug

- Sendmail uruchamiany był jako aplikacja SUID-0.
- Po zbindowaniu portu 25, wykonywane było wywołanie systemowe setuid() zmieniając UID na wartość niezerową by pozbyć się niepotrzebnych już dodatkowych uprawnień.

Tzw. Sendmail Capabilities Bug

- Sendmail uruchamiany był jako aplikacja SUID-0.
- Po zbindowaniu portu 25, wykonywane było wywołanie systemowe setuid() zmieniając UID na wartość niezerową by pozbyć się niepotrzebnych już dodatkowych uprawnień.
- Ponieważ kiedyś wyłołanie to nie mogło się nie powieść, nie były wykonywane żadne akcje sprawdzające więc program umożliwiał dalsze działanie na prawach roota.

Tzw. Sendmail Capabilities Bug

- Sendmail uruchamiany był jako aplikacja SUID-0.
- Po zbindowaniu portu 25, wykonywane było wywołanie systemowe setuid() zmieniając UID na wartość niezerową by pozbyć się niepotrzebnych już dodatkowych uprawnień.
- Ponieważ kiedyś wyłołanie to nie mogło się nie powieść, nie były wykonywane żadne akcje sprawdzające więc program umożliwiał dalsze działanie na prawach roota.
- Nie wzięto pod uwagę iż proces może nie mieć uprawnienia CAP_SETUID.

Tzw. Sendmail Capabilities Bug

- Sendmail uruchamiany był jako aplikacja SUID-0.
- Po zbindowaniu portu 25, wykonywane było wywołanie systemowe setuid() zmieniając UID na wartość niezerową by pozbyć się niepotrzebnych już dodatkowych uprawnień.
- Ponieważ kiedyś wyłołanie to nie mogło się nie powieść, nie były wykonywane żadne akcje sprawdzające więc program umożliwiał dalsze działanie na prawach roota.
- Nie wzięto pod uwagę iż proces może nie mieć uprawnienia CAP_SETUID.
- Problemem nie był jednak sendmail a kernel...

Tzw. Sendmail Capabilities Bug c.d.

Reguły emulacji SUID-0 dla kernela <2.2.16

```
if (uid==0 or file-to-exec-is-setuid)
   fE = fI = ¬0;
   fP = 0;
 else
   fI = fP = fE = 0;
```

Tzw. Sendmail Capabilities Bug c.d.

Reguły emulacji SUID-0 dla kernela <2.2.16

```
if (uid==0 or file-to-exec-is-setuid)
   fE = fI = ¬0;
   fP = 0;
 else
   fI = fP = fE = 0;
```

- Problem w tym, że każdy użytkownik ma prawo do usunięcia dowolnego bitu z *pI*.

Tzw. Sendmail Capabilities Bug c.d.

Reguły emulacji SUID-0 dla kernela <2.2.16

```
if (uid==0 or file-to-exec-is-setuid)
   fE = fI = ¬0;
   fP = 0;
 else
   fI = fP = fE = 0;
```

- Problem w tym, że każdy użytkownik ma prawo do usunięcia dowolnego bitu z *pI*.
- $pP' = (pI \ \& \ fI) \mid (fP \ \& \ X)$

Tzw. Sendmail Capabilities Bug c.d.

Reguły emulacji SUID-0 dla kernela <2.2.16

```
if (uid==0 or file-to-exec-is-setuid)
   fE = fI = ¬0;
   fP = 0;
 else
   fI = fP = fE = 0;
```

- Problem w tym, że każdy użytkownik ma prawo do usunięcia dowolnego bitu z *pI*.
- $pP' = (pI \ \& \ fI) \mid (fP \ \& \ X)$
- Wniosek: każdy, nawet nieuprzywilejowany użytkownik, mógł zabrać CAP_SETUID z *pI* jednocześnie powodując, ze sendmail mimo iż uruchomiony z SUID-0, nie miał tego uprawnienia więc nie mógł zrzuć praw roota.

Tzw. Sendmail Capabilities Bug c.d.

Reguły emulacji SUID-0 dla kernela >2.2.15

```
if (uid==0 or file-to-exec-is-setuid)
   fE = fP = ¬0;
   fI = 0;
 else
   fI = fP = fE = 0;
```

Tzw. Sendmail Capabilities Bug c.d.

Reguły emulacji SUID-0 dla kernela >2.2.15

```
if (uid==0 or file-to-exec-is-setuid)
   fE = fP = ¬0;
   fI = 0;
 else
   fI = fP = fE = 0;
```

- Błąd został łatwo załatany ale "niesmak" co do POSIX Capabilities pozostał na lata.

Wady Capabilities.

Wady Capabilities.

- Źle dobrane uprawnienia.

Wady Capabilities.

- Źle dobrane uprawnienia.
- Trzymanie informacji o uprawnieniach przypisanych do plików w xattr co powoduje możliwość desychronizacji (podobnie jak w SELinux).

Wady Capabilities.

- Źle dobrane uprawnienia.
- Trzymanie informacji o uprawnieniach przypisanych do plików w xattr co powoduje możliwość desychronizacji (podobnie jak w SELinux).
- Niektóre aplikacje napisane są w taki sposób, że nie jest możliwe użycie Capabilites zamiast SUID-0.

Możliwości rozwoju

Możliwości rozwoju

- Początkowo zbiory uprawnień trzymane były w 32-bitowych zmiennych. Z czasem jednak ilość uprawnień przekroczyła 32 i potrzebne było rozszerzenie.

Możliwości rozwoju

- Początkowo zbiory uprawnień trzymane były w 32-bitowych zmiennych. Z czasem jednak ilość uprawnień przekroczyła 32 i potrzebne było rozszerzenie.
- Zamiast użyć początkowo proponowanych zmiennych 64-bitowych, zastosowano inne podejście - tablicę składającą się ze strutur zawierających 32-bitowe zbiory.

Możliwości rozwoju

- Początkowo zbiory uprawnień trzymane były w 32-bitowych zmiennych. Z czasem jednak ilość uprawnień przekroczyła 32 i potrzebne było rozszerzenie.
- Zamiast użyć początkowo proponowanych zmiennych 64-bitowych, zastosowano inne podejście - tablicę składającą się ze strutur zawierających 32-bitowe zbiory.
- W podobny sposób można rozszerzyć zbiór w przyszłości nie powodując braku kompatybilności wstecznej.

Możliwości rozwoju

- Początkowo zbiory uprawnień trzymane były w 32-bitowych zmiennych. Z czasem jednak ilość uprawnień przekroczyła 32 i potrzebne było rozszerzenie.
- Zamiast użyć początkowo proponowanych zmiennych 64-bitowych, zastosowano inne podejście - tablicę składającą się ze strutur zawierających 32-bitowe zbiory.
- W podobny sposób można rozszerzyć zbiór w przyszłości nie powodując braku kompatybilności wstecznej.
- Ale tylko jeśli nie usuniemy starych uprawnień a więc wprowadzenie mniejszych uprawnień wymagałoby dublowania niektórych.

Możliwości rozwoju

- Początkowo zbiory uprawnień trzymane były w 32-bitowych zmiennych. Z czasem jednak ilość uprawnień przekroczyła 32 i potrzebne było rozszerzenie.
- Zamiast użyć początkowo proponowanych zmiennych 64-bitowych, zastosowano inne podejście - tablicę składającą się ze strutur zawierających 32-bitowe zbiory.
- W podobny sposób można rozszerzyć zbiór w przyszłości nie powodując braku kompatybilności wstecznej.
- Ale tylko jeśli nie usuniemy starych uprawnień a więc wprowadzenie mniejszych uprawnień wymagałoby dublowania niektórych.
- Niektórzy chcieliby mieć możliwość kontrolowania nawet każdego wywołania systemowego, jednak inni twierdzą, że do takich rzeczy lepiej nadają się mechanizmy typu SELinux.

Bibliografia

- capabilities(7)
- /usr/include/linux/capability.h
- "POSIX file capabilities: Parceling the power of root" na IBM developerworks.
- "A bid to resurrect Linux capabilities" na LWN.net.
- "Restricting root with per-process securebits" na LWN.net.
- "Fixing CAP_SETPCAP" na LWN.net.
- `http://userweb.kernel.org/~morgan/sendmail-capabilities-war-story.html`
- `http://www.friedhoff.org/posixfilecaps.html`

Koniec

Dziękuję za uwagę.