Vol.2020-IOT-51 No.7
Vol.2020-SPT-39 No.7
2020/9/4

多段ssh接続を管理するsshrの改良

城後 明慈[1,a)]　長田 智和[2,b)]

概要：目的のサーバにアクセスするために外部からサーバへのアクセスへは踏み台サーバからのアクセスが必要である．踏み台サーバでは外部から第三者に乗っ取られることや不特定多数の攻撃の中継地点を防止するためのものである．しかし，利用している踏み台サーバが停止すると学内での復旧作業が必要になってしまうそこで，新規踏み台サーバとして sshr を提案する．sshr は鶴田博文氏が作成したものである．sshr はシステム管理者が組み込み可能なフック関数を用いてシステム変化に追従できる ssh プロキシサーバである．本稿では sshr を改良し弊学へ利用できるように実装をした．

1. はじめに

琉球大学知能情報コースでは教育情報システムが存在する．外部からのアクセスは踏み台サーバを通って目的のサーバに ssh する．教育情報システムのサーバの役割として，基幹システムとして動作するサーバ，または，学生が研究または実験のために使用するためのサーバである．この中でも基幹システムは学生のユーザ情報を管理を行う ldap サーバ，学科のウェブサイト，および学生個人のサイトを配信する Web サーバ，DNS サーバなどが存在する．この基幹システムは琉球大学知能情報コースの学生が主体となって運用，開発，保守を行なっており，システム管理チームが行なっている，システム管理チームは基幹システムの障害などに対応しており，学内のネットワーク内で復旧作業を行う．天候不良などで学内での作業が難しく外部からの作業を行うときがある．その場合，上記の踏み台サーバを通して復旧作業を行う．しかし，現在教育情報システムに踏み台サーバは一つしか存在せず，仮に学外からの作業を余儀なくされる場合で踏み台サーバが何らかの障害で停止してしまった場合，直接学校で作業を行う必要性がある．さらに，琉球大学では計画停電や，台風による急な停電が起こることがある．その場合，システム管理チームでは停電が起こりうる可能性がある時，事前に教育情報システムを停止させる必要性がある．停止させる場合も大学で作業を行う必要性がある．本論文では，ユーザに用いられるクライアントツールの制限や変更を要求せず，システム管理者が自由に組み込み可能なフック関数を用いてシステム変化に追従できる SSH プロキシサーバである sshr を利用し踏み台サーバを構築する．sshr はユーザ名から接続先サーバを決定するためのフック関数をシステム管理者が自由に実装，導入ができる SSH プロキシサーバである．この sshr サーバを琉球大学知能情報コースで保有されているさくらクラウド上で構築を行うことで停電による踏み台サーバの停止を防ぐことができる．

1 琉球大学大学院理工学研究科情報工学専攻
2 琉球大学工学部工学科知能情報コース
a) k198576@ie.u-ryukyu.ac.jp
b) nagayan@ie.u-ryukyu.ac.jp

2. Ubuntu

今回の環境は Ubuntu18.04 を採用した．sshr のサンプルのコードは CentOS であったが教育情報システムで試したのは Ubuntu を洗濯した，知能情報コースで利用されている教育情報システムは CentOS をベースで構築されている．しかし，教育情報システムは 5 年に一度のシステム更新がありその中で今回は CentOS ではなく Ubuntu を利用することを想定しているため，今回の sshr の環境も Ubuntu で行なった．

3. ssh

ssh とは Secure Shell の略でネットワークに接続されたサーバなどに接続するために用いるものである．琉球大学知能情報コースでも大学内の教育情報システムであるオンプレミスのシステムや KVM 上の VM などに接続するために ssh を用いて接続を行う．教育情報システムは学内からは ssh を行う時には基幹システムに Web サーバや DNS サーバなどに直接アクセスすることができる．第三者は学内のネットワーク利用しなければアクセスはできないため，第三者からの攻撃を回避することができる．学外からのアクセスは教育情報システムにある踏み台サーバを通してアクセスすることで各基幹システムにアクセスができるよう

になっている．現在，教育情報システムには踏み台サーバが一つだけ存在しており，また，オンプレミス上の VM で稼働しているため，停電時にはアクセスが不可能になる．さらに．踏み台サーバは一つだけのため，VM が停止しした場合学内での作業を余儀無くされる．そこで，システム管理者．ユーザが利用しやすい SSH プロキシサーバである sshr の構築を行う．

4. sshr

sshr とは鶴田博文氏が作成した ssh プロキシサーバである．sshr はユーザ名から接続先サーバからを決定するためのフック関数を用いてシステム変化に追従できるものである．これによりユーザ名を用いて透過的に目的のサーバに ssh 接続できるため，接続先の IP アドレス，ホスト名の変更があったとしてもユーザが意識せず変更の追従を行う必要性がない．また，システム管理者が自由に組み込みが可能なフック関数を用いてシステム変化に追従が可能な SSH プロキシサーバである SSH でサーバに接続する場合，ユーザが目的のサーバに接続する権限を持っているかどうかを確認するためにユーザの認証を行う必要がある．sshr ではユーザ認証をパスワード認証，公開鍵認証をサポートしている．sshr サーバを介してのパスワード認証はクライアント側から受けた認証をそのままリクエスト側に送り，目的のサーバ側で認証を行う．また，公開鍵認証ではプロキシサーバ側とクライアント側での公開鍵認証を行い, プロキシサーバ側で保持される秘密鍵を用いて目的のサーバ側へと認証を行う．

4.1 sshr で使用できる関数

sshr はシステム管理者が自由に組み込むことができる SSH プロキシサーバである．自由に組み込みができる関数が 3 つあるのでそれについて紹介する．FindUpStreamHook は接続先の決定に用いる関数である．次に FetchAuthorizedKeysHook は公開鍵認証の際に sshr サーバで公開鍵を検索を行う関数である最後に FetchPrivateKeyHook は秘密鍵を参照する処理である以上の 3 つが用意されている関数である．今回使用する関数は FindUpStreamHook 関数を用いて実装を行なった．接続先については基幹システムの一部である Web システムが稼働しているサーバへと接続を行う想定である．

4.2 sshr サーバへの公開鍵登録方法

sshr サーバで公開鍵認証を行うためにはクライアント側を認証する公開鍵が必要になる．そのため，今回は公開鍵情報を登録するためにコマンド (sshr-copy-id) を作成した．sshr を作成した鶴田博文氏のデモでは DB に公開鍵，ホスト名，ユーザ名などの情報操作は WebAPI を介して行っていた．また，sshr を介した ssh 接続を行う時は WebAPI を介してユーザ情報を取得していた．本論文で紹介している sshr は基幹システム上で利用されることを前提に sshr-copy-id コマンドを作成した.. コマンドを採用した理由は ldap サーバなどは基幹システム内にあり，登録のため，Web サイトを公開することで第三者が不正に登録する可能性があるからである．また，Web サイトをプライベートな環境で公開すると VPN 接続などを強いられるため今回はコマンドを利用した. そこで，基幹システム内でのコマンドを操作で登録を行うことで最小限に第三者からの不正登録を防ぐごとができる．また，基幹システムのユーザ情報は ldap サーバを用いて管理を行なっている．そこで，さらに安全性を高めるため ldap サーバに問い合わせを行う．これにより事前にユーザ情報が存在するかの確認を行うことで不正なユーザが登録されることを防いだ. ここで，sshr-copy-id の図について図 1 に示す

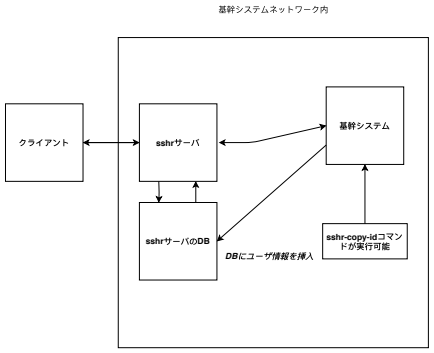

図 1: sshr-copy-id の図

sshr-copy-id は golang で実装を行なった．golang を採用したのは，sshr も golang で作られており使用している OSS と言語を合わせ統一性を保つためである．また，コマンドラインツールを作成を支援する cobra を用いてコマンドの開発を行なった．これは，コマンド作成を支援するツールを使うことでのちにコマンドの改良をしやすくするためや，可読性を高めるためである．sshr-copy-id の実行コマンドを下記に示す.

```
sshr-copy-id -l user -k id_rsa.pub
```

Code 1: sshr-copy-id コマンド

sshr-copy-id はユーザの登録として，ユーザ名と登録したい公開鍵を指定することで sshr サーバに登録することができる．次に sshr-copy-id の ldap サーバへのユーザ検索の実装について示す.

```
var keyCmd = &cobra.Command{
  Use: "key",
  Short: "sshr-copy-id␣command",
  Long: ‘sshr-copy-id is a command that registers a
       public key with the sshr server.‘,
```

```
  RunE: func(cmd *cobra.Command, args []string) error
        {

    if err := ldapCon(); err != nil {
      return err
    }

    exec.Command("ssh-keygen", ,"-m", "PEM", "-t", "
        rsa", "-f"+o.optname).Output()

    fileName := o.optname + ".pub"
    bytes, err := ioutil.ReadFile(fileName)
    if err != nil {
      return err
    }
    secretKey := (string(bytes))
    insertInfo(secretKey)

    return nil
  },
}


func init() {
  rootCmd.AddCommand(keyCmd)
  keyCmd.Flags().StringVarP(&o.optid, "optid", "l", "
      ", "student␣option")
  keyCmd.Flags().StringVarP(&o.optname, "optname", "f
      ", "", "file␣option")
}
```

Code 2: ldap サーバへの検索

上記コードは sshr-copy-id の main となるコードである．鍵の作成は ssh-keygen コマンドを用いて鍵を作成する．今回鍵を作成する際に option を付属して PEM と作成される鍵を指定している．その後，作成した公開鍵を sshr サーバで稼働している DB のサーバに登録を行う．次に，下記は ldap サーバへの接続とユーザの検索についてのコードである．

```
func ldapCon() error {

  l, err := ldap.DialURL("LDAPSERVER")
  if err != nil {
    return fmt.Errorf("ldap␣connect␣miss␣%W", err)
  }

  if err = l.StartTLS(&tls.Config{InsecureSkipVerify:
      true}); err != nil {
    return fmt.Errorf("TLS␣miss␣%W", err)
  }

  if err = l.Bind("cn=Manager,ou=ie,o=u-ryukyu,c=jp",
      LDAPPASSWORD); err != nil {
    return fmt.Errorf("Bind␣miss␣%W", err)
  }

  searchRequest := ldap.NewSearchRequest(
    CN,
    ldap.ScopeWholeSubtree, ldap.NeverDerefAliases, 0,
        0, false,
    fmt.Sprintf("(&(uid=%s))", o.optid),
    []string{"dn", "uid"},
    nil,
  )

  sr, err := l.Search(searchRequest)
  if err != nil {
    return fmt.Errorf("search␣error␣%w", err)
  }

  if len(sr.Entries) != 1 {
    return fmt.Errorf("User␣does␣not␣exist␣or␣too␣many
        ␣entries␣returned")
  }
  entry := sr.Entries[0]

  fmt.Printf("%s:␣%v\n", entry.DN, entry.
      GetAttributeValue("uid"))

  return nil
}
```

Code 3: sshr-copy-id のコマンド部分

sshr-copy-id はコマンドで渡されたユーザ名を基幹情報システムの一部である，ldap サーバに接続しユーザ名の検索を行う．また，ユーザ名と公開鍵はそれぞれオプションとして-l と-f で渡すことで sshr サーバに存在する DB にユーザ情報，公開鍵を格納している．

4.3 IPMI

Intelligent Platform Management Interface (IPMI) は標準化されたメッセージベースのハードウェア管理インターフェースである．IPMI の中核にはベースボード管理コントローラーまたは管理コントロート呼ばれるハードウェアチップが存在する．IPMI の機能として，電源のオンオフ，シリアルコンソーリの操作，ブートデバイスの変更などの機能を備えている．さらに，OS とは完全に独立しているのでハングアップしてしまった際でも実行可能である．今回 sshr を用いてクラウド上に構築することを想定している．停電時はバックアップのために基幹システムがシャットダウンを行う．

5. sshr の実装

この章では実際に基幹システム内で使用した sshr をどのように作成したかなどについて示す．また，琉球大学知能情報コースで利用されることから，本論文では琉球大学知能情報コースで利用するために作成した sshr のことを ie-sshr と記す．ie-sshr の実装は用意されているフック関数を用いて実装を行なった．実装内容としては用意されたフック関数を用いたパスワード認証，鍵認証のシステムの実装についてである

5.1 ie-sshr の利用方法

最初に今回構築する，ie-sshr についての想定している利用方法について解説する．

- 基幹システム内の踏み台サーバに接続を行う
- 接続後，sshr-copy-id を用いて登録を行う
- 実行後公開鍵と秘密鍵が発行されるので，秘密鍵をローカルに落としておく
- その後ローカルから sshr に対して ssh を行うことで接続したいサーバへと接続される．

5.2 ie-sshr の実装コード

ie-sshr の実装について記述する．最初に今回の ie-sshr の図を図 2 に示す．

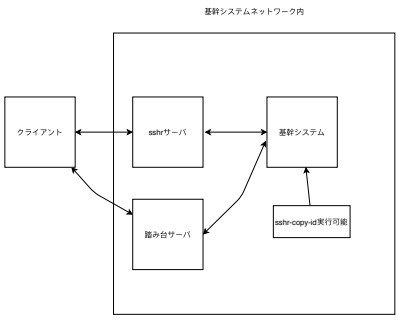

図 2: sshr の概要図

となり，ユーザはこの手順を通して ie-sshr を利用できる．今回の ie-sshr の実装部分は以下のようになる．

```
func main() {
  confFile := "./ie-sshr.toml"

  db, err := sqlx.Open("mysql", "USER:PASS@tcp(HOST:
      PORT)/sshr")
  if err != nil {
    log.Fatal(err)
  }
  defer db.Close()

  sshServer, err := sshr.NewSSHServer(confFile)
  if err != nil {
    logrus.Fatal(err)
  }

  sshServer.ProxyConfig.FindUpstreamHook =
      FindUpstreamByUsername
  sshServer.ProxyConfig.FetchAuthorizedKeysHook =
      FetchAuthorizedKeysMyHook
  sshServer.ProxyConfig.FetchPrivateKeyHook =
      FetchPrivateKeysMyHook
  if err := sshServer.Run(); err != nil {
    logrus.Fatal(err)
  }
}
```

Code 4: ie-sshr の main コード

上記は ie-sshr の main となるコードである．ie-sshr の main では confFile を読み込む必要がある．この confFile は IP アドレスやポート番号などの sshr サーバがプロキシサーバの情報について記述するものである．そのほかにも公開鍵認証で使用する sshr サーバから目的のサーバへの認証時に使用する秘密鍵を指定する．これらの情報でパスワード認証および公開鍵認証をする必要な情報を設定できる．

5.3 FindUpstreamHook

```
func FindUpstreamByUsername(username string) (string,
      error) {
  db, err := sqlx.Open("mysql", "USER:PASS@tcp(HOST:
      PORT)/sshr")
  if err != nil {
    log.Fatal(err)
  }

  row := db.QueryRow("SELECT␣*␣FROM␣StudentInfo␣where
      ␣studentNum=?", username)
  if err := row.Scan(&username); err != nil {
    return "ssht", nil
  } else {
    return "", errors.New(username + "'s␣host␣is␣not␣
        found!")
  }
}
```

Code 5: ie-sshr のホスト検索コード

続いて FindUpStreamHook ではユーザの ssh したい Host 先を返す関数である．今回の ie-sshr ではユーザ情報が DB 内に存在していれば Host 先情報を返すというものである．今回は用意した VM の sshd というサーバへ接続するように Host 情報を返すように行なった．パスワード認証のみであれば FindUpStreamHook というフック関数のみで行うことができる．

5.4 FetchAuthorizedKeysHook

```
func FetchAuthorizedKeysMyHook(username, host string)
      ([]byte, error) {
  db, err := sqlx.Open("mysql", "USER:PASS@tcp(HOST:
      PORT)/sshr")
  if err != nil {
    log.Fatal(err)
  }

  var user User
  err = db.Get(&user, "SELECT␣*␣FROM␣StudentInfo␣
      where␣StudentNum=?␣LIMIT␣1", "ie-user")
  if err != nil {
    return []byte("0"), err
  }
  log.Println(user);
  return []byte(user.KEY), nil
}
```

Code 6: ie-sshr の公開鍵検索コード

次に FetchAuthorizedKeysHook はクライアント側と sshr サーバ間で公開鍵認証を行うためのものである．今回はユーザ情報を元に DB からユーザと一致する公開鍵を返すように行なったものである．

5.5 FetchPrivateKeyHook

```
func FetchPrivateKeysMyHook(username string) ([]byte,
      error) {
  f, err := os.Open("master")
  if err != nil {
    fmt.Println("error")
  }

  defer f.Close()

  buf, err := ioutil.ReadAll(f);
  if err != nil {
    return []byte("0"), err
  }
  fmt.Println(buf)
  return buf, nil
}%
```

Code 7: ie-sshr の秘密鍵検索コード

最後に ie-sshr サーバから目的のホストへと公開鍵認証を行うために秘密鍵を返すものである．以上のコードが今回の ie-sshr で使用するために記述したものである．これで分かるようにフック関数を用いることでシステム管理者が構築しやすいものとなっている．

5.6 コードに対しての評価

sshr を用いて基幹情報システムで利用しやすいように用意されたフック関数を用いて構築を行なった．今回使用した関数は少ない行数で実装することができた．また，フック関数も利用のしやすいもので必要な情報を取得さえできれば良いためコードもシンプルになり後継が改良しやすいと考えられる．

5.7 セキュリティについて

sshr は踏み台サーバとして動作することを前提にされている故に，外部に公開するにあたってセキュリティ対策を行う必要がある．認証方式については今まで通り踏み台サーバでのパスワード認証と公開鍵認証方式を行うことができるものとする．公開鍵認証に関しては公開鍵の登録とユーザの登録が必要であるが基幹システム上で登録のみを行うことを想定している．また，第三者からの攻撃として ie-sshr サーバに対して攻撃を行われる可能性がある．そこで，ie-sshr サーバへの ssh を行うことができるのは学内のプライベートネットワークのみへのアクセスとして制限を行うことでふせぐことができる．さらに，システム管理者以外が sshr サーバに接続することを防ぐため ldap でのシステム管理者のユーザ権限が振られているユーザのみ ssh できるように行うことで攻撃を防ぐことができる．また，踏み台サーバとして外部に公開することは果実であるため第三者から ssh をされることがある．今回の実装では 2222 番ポートを使用しているが臨機にポートを変えるなどして変更を加えることで第三者からの不正にアクセスすることを回避できる．

6. 性能比較

今回構築した ie-sshr サーバと類似した動きができるようなものと比較を行い性能の比較を行う．sshpiper は鶴田博文氏の論文で比較を出されているため本論文では ssh の ProxyJump と比較を行なった．

6.1 ssh の ProxyJump との比較

ssh の ProxyJump は ssh の config ファイルに記述することで多段 ssh をすることができる．ssh の ProxyJump を行うときの例を下記に示す

```
Host host1
 HostName host.ie.u-ryukyu.ac.jp
 Port 22
 User ie-user
 IdentityFile ~/.ssh/ie-host

Host host2
 HostName host2.ie.u-ryukyu.ac.jp
 Port 22
 User ie-user
 IdentityFile ~/.ssh/ie-host2
 ProxyCommand ssh -W %h:%p host1
```

Code 8: ie-sshr の秘密鍵検索コード

上のように設定を行うことで Host1 を経由して Host2 へと接続される．しかし，ssh の ProxyJump はユーザが個人で設定を行うものである．ゆえにシステム管理者側がホスト名の変更や IP アドレスの変更などがある場合，ユーザ側がシステム変更に追従必要となってしまう．ssh の ProxyJump は少ないユーザが利用する場合や, 複雑な構成ではないシステムなどでは利用しやすい.sshr は多くのユーザかつ，複雑な構成のシステムであれば向いているため今回の ie-sshr のほうが適していると考えられる．

7. 今後の課題

実装を行なった ie-sshr に対しての課題を上げていく．今回実装を行なった ie-sshr は以下の課題を解決することで実際に運用を行うとすれば必要と感じた課題である．

7.1 OpenSSH 形式のパスフレーズ付き秘密鍵への対応

今回使用しいてる sshr ではパッケージの一つとして x/crypto/ssh が使用されている．この x/crypto/ssh パッケージは ssh 関連のパッケージである．しかし，OpenSSH のバージョン 7 .8 からパスワフレーズで保護を行うと 7.8 以前では秘密鍵の最後に RSA PRIVATE KEY と作成されるが，7.8 からは OPEN SSH PRIVATE KEY と末尾に作成されるようになった．これにより 7.8 以前の version では問題ないが，7,8 以降では秘密鍵として認識されずパスワード認証方式になる．よって，x/crypto/ssh パッケージを OpenSSH7.8 以上の秘密鍵を読み込みができるようにしなければらない. 現在，x/crypto/ssh パッケージは GitHub の Issue として上がっているが対応されていないため，リポジトリを fork して c/crypto/ssh を読み込めるように行うなどの改良が必要となってくる.

7.2 ユーザごとの公開鍵情報

学内で設置される ie-sshr は各学生がブログを公開できる Web サーバの役割を担っているところへと接続することを想定している．しかし，教育情報システムでは学生の実験，学習のための VM の貸し出しをシステム管理者が行っている．VM の貸し出しは，貸し出しをしたいユーザが学科の Web サービスより選択し，VM が作成され自動で IP アドレスが振られる．しかし，生徒が貸し出された VM に入るには踏み台サーバを経由しなければならないため ie-sshr を利用する場合はパスワード認証のみでしか対応することができない. よって柔軟にユーザが ssh をするためにも改善が必要である.

7.3 Web 上での鍵登録

鶴田博文氏の論文では sshr の登録は Web 上で鍵を登録する想定になっていた．今回,ie-sshr ではコマンドでユーザ情報の登録を行なっている．これは, 不正な第三者からの登録を防ぐために，今回は基幹システムのみで利用できるコマンドを作成した．しかし, ユーザ側が利用するために基幹システムに一度は接続をしなければならないのは手間となる．そこで，知能情報コースに所属している学生は Google の Gsuite でアカウントの管理を行なっている．Oauth 認証を用いて公開鍵登録を行う Web サイトを構築することで第三者の不正アクセスも検知できる．さらに，教育情報コースでは VM の貸し出し，IP アドレスと MAC アドレスの紐付けなどのサービスを担っている Akatsuki と呼ばれている教育情報システムでの Web サービスがある．その Web サービスと統合することによってユーザも使いやすい環境になると考えられる.

7.4 IPMI の実装

ie-sshr では SSH プロキシサーバとしての機能を実装している．ここで，知能情報コースで年に数回の計画停電や台風などの災害などにより緊急で教育情報システムが停止してしまうことがある．計画停電の場合は事前に周知されるためシステム管理者が基幹システムの停止を行うなどをして対策を行うことができる．しかし，台風などの場合システム管理者が危険を避けるために不要に外出を行うことができない．そこで,ie-sshr にその機能を組み込むことを提案する．実際に機能を作成する場合は SSH とプロトコルが違うため golang で IPMI を受け取ることができるパッケージを自作することで実現ができると考えられる.

8. まとめ

本論文では，ユーザに用いるクラアントツールに制限や変更を要求せず，システム管理者が自由に組み込むことができる sshr サーバを改良し基幹システム上で検証を行なった．実際に知能情報コースの基幹システム上で使用してみたが sshr で用意されている関数を使うことで実装をすることができた. 知能情報コースのシステムは 2020 年を迎え，現行のシステムから 5 年経過したことによりシステム更新を行う. よって，sshr の利点でであるクライアント側が IP アドレスの変更，ホスト名の変更などがあったとしてもクライアント側は意識をしなくてよい．ie-sshr はクラウド上での構築を行うこと想定しているため，システム更新で教育情報システムのサービスを利用したい学生にも対応することが可能であると考えられる．今回 ie-sshr の課題でも上げられた公開鍵認証方式は大学の基幹システムを第三者からの攻撃から防ぐのに有効であると考えられるため早期に取り掛かる必要がある．ie-sshr の今後の展開としては一つはクラウド上での構築を行い学生が利用できる環境にすること. 二つめに教育情報システムでは VM の貸し出しを行なっている. 今回の検証では ie-sshr は特定のサーバに対して接続するような形になっている. そこで，学生が借りた VM でも ie-sshr でホストの情報を登録することで SSH プロキシサーバへと接続され任意のホストに接続を行うことなどが実現できると考えられる．そこで golang の cui ツールを作るためのパッケージを用いることで接続したい VM を選択するなども可能である.

参考文献

[1] 鶴田博文，松本亮介：sshr: ユーザに変更を要求せずに システム変化に追従可能な SSH プロキシサーバ，*IOTS 2019* (2019).
[2] 田中大海，城戸翔大，長田智和，谷口裕治：情報系学科における教育研究用情報システムの運用管理に関する取り組み，*IOTS 2017* (2017).
[3] sshr: *https://github.com/tsurubee/sshr*.
[4] sshr.crypto: *https://github.com/tsurubee/sshr.crypto*.
[5] sqlx: *https://github.com/jmoiron/sqlx*.
[6] rfc4252: *https://tools.ietf.org/html/rfc4252*.