KARADENİZ TEKNİK ÜNİVERSİTESİ
MÜHENDİSLİK FAKÜLTESİ
BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ

GÜVENLİ E-POSTA GEÇİDİ

TASARIM PROJESİ

Gizem İSKENDEROĞLU
Halit ALPTEKİN

2015-2016 GÜZ DÖNEMİ

KARADENİZ TEKNİK ÜNİVERSİTESİ
MÜHENDİSLİK FAKÜLTESİ
BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ

GÜVENLİ E-POSTA GEÇİDİ

TASARIM PROJESİ

Gizem İSKENDEROĞLU
Halit ALPTEKİN

Bu projenin teslim edilmesi ve sunulması tarafımca uygundur.

Danışman : Öğr.Gör.Selçuk CEVHER .....

2015-2016 GÜZ DÖNEMİ

IEEE Etik Kuralları
IEEE Code of Ethics

Mesleğime karşı şahsi sorumluluğumu kabul ederek, hizmet ettiğim toplumlara ve üyelerine en yüksek etik ve mesleki davranışta bulunmaya söz verdiğimi ve aşağıdaki etik kurallarını kabul ettiğimi ifade ederim:

1. Kamu güvenliği, sağlığı ve refahı ile uyumlu kararlar vermenin sorumluluğunu kabul etmek ve kamu veya çevreyi tehdit edebilecek faktörleri derhal açıklamak;
2. Mümkün olabilecek çıkar çatışması, ister gerçekten var olması isterse sadece algı olması, durumlarından kaçınmak. Çıkar çatışması olması durumunda, etkilenen taraflara durumu bildirmek;
3. Mevcut verilere dayalı tahminlerde ve fikir beyan etmelerde gerçekçi ve dürüst olmak;
4. Her türlü rüşveti reddetmek;
5. Mütenasip uygulamalarını ve muhtemel sonuçlarını gözeterek teknoloji anlayışını geliştirmek;
6. Teknik yeterliliklerimizi sürdürmek ve geliştirmek, yeterli eğitim veya tecrübe olması veya işin zorluk sınırları ifade edilmesi durumunda ancak başkaları için teknolojik sorumlulukları üstlenmek;
7. Teknik bir çalışma hakkında yansız bir eleştiri için uğraşmak, eleştiriyi kabul etmek ve eleştiriyi yapmak; hatları kabul etmek ve düzeltmek; diğer katkı sunanların emeklerini ifade etmek;
8. Bütün kişilere adilane davranmak; ırk, din, cinsiyet, yaş, milliyet, cinsi tercih, cinsiyet kimliği, veya cinsiyet ifadesi üzerinden ayırımcılık yapma durumuna girişmemek;
9. Yanlış veya kötü amaçlı eylemler sonucu kimsenin yaralanması, mülklerinin zarar görmesi, itibarlarının veya istihdamlarının zedelenmesi durumlarının oluşmasından kaçınmak;
10. Meslektaşlara ve yardımcı personele mesleki gelişimlerinde yardımcı olmak ve onları desteklemek.

IEEE Yönetim Kurulu tarafından Ağustos 1990'da onaylanmıştır.

ÖNSÖZ

Bu proje ile kurumsal firmaların yüzbinlerce dolar ödediği çözümlerin eşdeğerlerinin çok ufak miktarlara da başarıyla gerçekleştirilebileceğini göstermiş olduk. Saldırı engelleme sistemlerinin ilk ve en önemli aşaması olan saldırı tespit kısmını başarıyla tamamladık. Bu kapsamda diğer çözümler içerisinde olmayan veri kaçaklarını önleme, e-posta sunucularına karşı yapılan saldırı engelleme ve siber tehditlerden erken haberdar olma özelliklerini ekleyerek farklı bir çalışma ortaya koyduk. Sadece teorik çalışmalarda kalmayan, güncel tehditlere karşı da başarılı olan bu çalışmamıza, yaptığımız projenin amacıyla uygun olarak Türkçe bir kelime olan *KORGAVUS* ismini seçtik. Umarız benzer proje geliştirmek isteyen tüm arkadaşlara yardımcı olacak bir çalışma ortaya çıkarmışızdır.

Gizem İSKENDEROĞLU
Halit ALPTEKİN
Trabzon 2015

İÇİNDEKİLER

ÖZET

Proje kapsamı içerisinde *Güvenli Mail Geçidi* gerçekleştirilmiştir. Gerçekleştirilen sistemin test edilebilmesi için kurumsal bir ağ yapısı tasarımı yapılmıştır. Buna uygun olarak kurban ve saldırgan makineler kurulmuştur. Kurban sunucusundan çıkan tüm trafik gerçekleştirilen geçit üzerinden geçmektedir. İlgili geçit içerisinde sadece e-posta trafiği toplanmaktadır. Toplanan e-posta trafiği daha sonradan analiz işlemine tabi tutulmaktadır.

Analiz edilen ve analiz edilmeyi bekleyen e-postalar bir panel aracılığı ile kullanıcıya gösterilmektedir. Bu panel içerisinden analiz edilen e-postanın herhangi bir tehdit içerip içermediği görülebilmektedir. Aynı şekilde kullanıcının tanımlayabileceği kurallar ile de e-posta trafiğinin uyuşup uyuşmadığı görülebilmektedir.

Geçit içerisinde en çok işlem yükü getiren paket toplama ve analiz aşamalarında işletim sisteminin sağladığı en alt araçlardan faydalanılmıştır. Bu derece alt seviye ile iletişime geçilmesinden dolayı hem performans hem de esneklik kazanılmıştır. Üst seviye araçlar ile elde edilemeyecek daha fazla veriye erişim sağlanmıştır.

Sistem üzerinden geçen e-posta trafiğine herhangi bir şekilde müdahale etmemektedir. Bu tarz bir müdahalenin gerçekleştirilmesi için öncelikle çok iyi bir tespit sistemine ihtiyaç duyulmaktadır. Proje kapsamında böyle bir tespit sistemi gerçekleştirilmiştir. Bu sistem aracılığı ile korunmak istenen ağa gelen e-posta trafiği analiz edilip içerisinde herhangi bir siber tehdit barındırıp barındırmadığının kontrolü yapılmaktadır. Burada bahsedilen siber tehditler zararlı dosya indiren site adresleri, dünya üzerinde yer alan saldırgan IP adresleri, spam e-posta gönderen adresler olabileceği gibi henüz tespit edilmemiş açıklıkları sömüren kod parçaları olabilmektedir. Örneğin yakın bir zamanda ortaya çıkan ve e-posta sunucuları hedef alan açıklığın sömürülmesi, gerçekleştirilen sistem tarafından başarıyla tespit edilmiştir.

Projenin amaçlarından bir diğeri de ağ içerisinden dışarı gönderilen e-postalar içerisinde önemli veri bulunan postaların tespitidir. Bu şekilde veri kaçaklarından da haberdar olunabilmektedir.

Sistem sadece içerisinde tanımlı olan imzalar aracılığı ile tespit yapmamaktadır. İmzaların yanında çeşitli kurallar da bulunmaktadır. Gerçekleştirilen geçit içerisine yeni bir kural eklemek çok kolaydır. Haliyle geliştirilmeye çok açık durumdadır. Aynı zamanda içerisinde bulunan imzalar da düzenli aralıklar ile güncelleştirilmektedir. Haliyle son siber tehditlerden hızlı bir şekilde haberdar olabilmektedir.

Piyasada bulunan ve sadece kurumsal firmalar tarafından güvenli e-posta geçitlerine farklı ve ucuz bir yorum getirilmiştir. Bu kapsamda *Raspberry PI* gibi ucuz bir donanım ile de benzer bir yapının kurulabileceği gösterilmiştir. Aynı zamanda kullanılan teknolojiler ile de *özgür ve açık kaynak* yazılımların bu tarz projeler için ne kadar uygun olduğu belirtilmiştir.

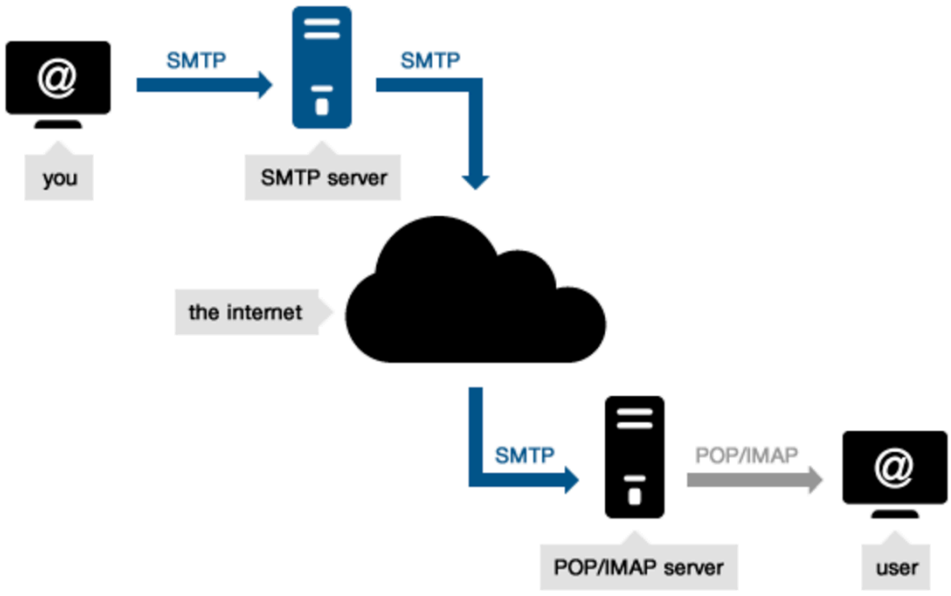
1. GENEL BİLGİLER

1.1. Giriş

E-Postaların gönderimi ve alımı çıktığı ilk günden bu yana büyük önem taşımaktadır. Birçok firma e-posta üzerinden işlerini yürütebilmektedir. Bu yüzden postaların ticari boyutları da olabilmektedir. Bir kişi farklı bir kişiyle iletişim kurmak için e-posta gönderebileceği gibi dosya paylaşımı için de gönderim yapabilmektedir. Bu derece yüksek öneme sahip bu yapı haliyle saldırganlar için de büyük önem arz etmektedir.

Bir kişinin e-posta gönderimi yapabilmesi için öncelikle bir e-posta hesabına sahip olması gerekmektedir. Kullanıcılar bu hesapları e-posta servis sağlayıcılarından alabilmektedir. Ücretsiz olarak sunulan hizmetlerin yanında ücretli çözümler de bulunmaktadır. Günümüzde kurumların çoğu kendi e-posta sunucularını kurmakta ve bu sunucular üzerinden internet ile e-posta trafiği oluşturmaktadır. Çünkü kurumun çalıştığı sektöre göre ücretsiz e-posta servisleri yeterli olmamaktadır. Örneğin savunma sanayisi firması bu tarz ücretsiz bir sistem kullanırsa üzerinde çalıştığı projelerin başka kimseler tarafından ele geçirilmesine de olanak tanır. Çünkü ücretsiz servislerin kullanıcı sayıları yüksektir ve saldırganlar tarafından cezbedici gelmektedir. Bu sayıların ötesinde böyle ücretsiz sistemlerin kurulma sebepleri başında bilgi toplama da gelmektedir. Ücretsiz sistemleri çok sayıda kullanıcı kullanır, çok sayıda kullanıcı da çok sayıda e-posta trafiği üretir. Böyle bir trafik büyük oyuncular için vazgeçilmezdir.

Bir e-posta hesabından bir diğer e-posta hesabına gönderim yapılabilmesi için SMTP[1] protokolüne ihtiyaç duyulmaktadır. Bu protokol aracılığı ile e-posta bir sunucudan diğerine gönderilebilir. Ancak böyle bir gönderim tek başına yeterli değildir. Kullanıcıların da kendi hesaplarını kontrol edebilmeleri gerekmektedir. İşte bu sorunun çözümü için de POP[2] veya IMAP[3] protokolleri kullanılmaktadır.

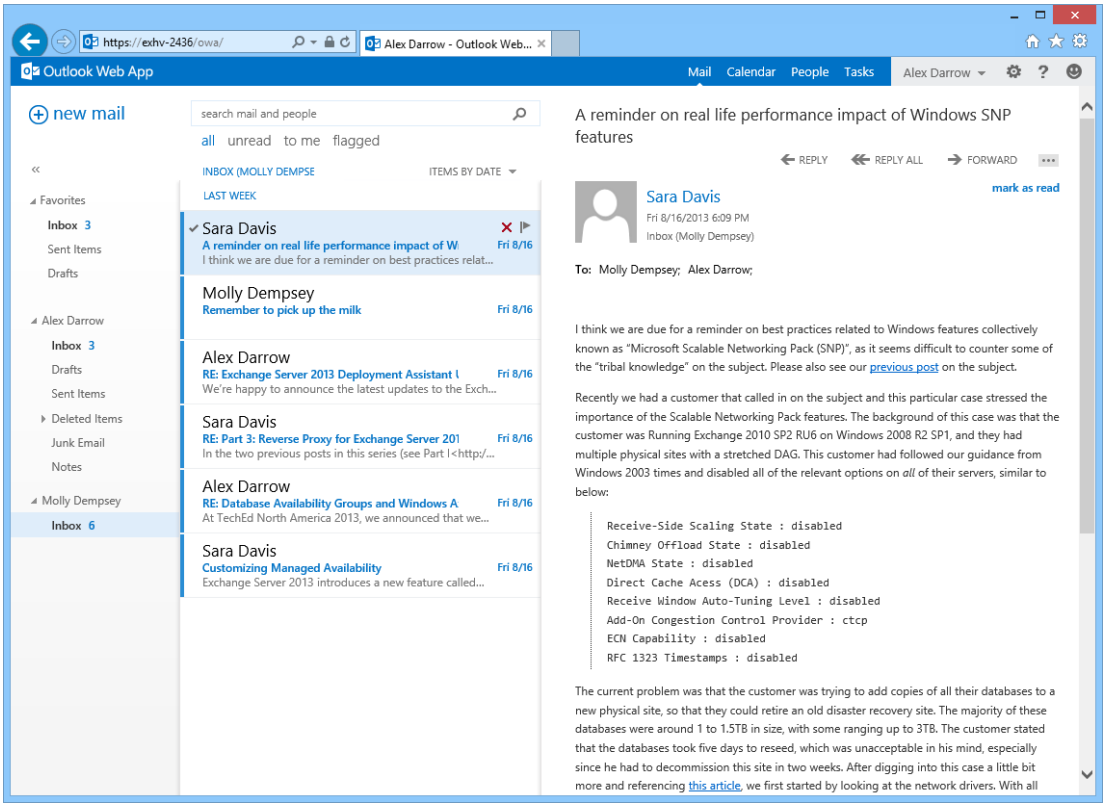


Şekil 1.1. E-Posta trafiği içerisinde kullanılan protokoller.

Bir e-posta gönderimi için ilk olarak bilgisayarımızda kurulu olan istemci aracılığı ile posta oluşturulur. Daha sonra bu posta hesap sağlayıcısı olan e-posta sunucusuna SMTP protokolü aracılığı ile gider. Postayı alan sunucu hedef sunucuyu ve IP adresini tespit eder. Daha sonra SMTP protokolü aracılığı ile sunucuya iletir. Burada hedef sunucu kendisine gelen e-postayı içerisinde tutar. Kendisine gelen postayı okumak isteyen kullanıcı ise IMAP veya POP protokollerinden birisini kullanarak bilgisayarında kurulu olan istemciye bu postayı indirir. Son olarak da istemci aracılığı ile posta içeriğini okur. Tüm yapı Şekil 1.1'de görülebilir.

E-posta trafiğinin öneminin arttığı günümüzde e-posta sunucu yazılımlarının da önemi artmış durumdadır. Şu anda GNU/Linux dünyasında en popüler mail sunucu çözümleri Postfix[4] ve Dovecot[5] yazılımlarıdır. Benzer şekilde Windows dünyasında ise Exchange[6] yazılımı gayet popüler durumdadır. Bu çözümlerin dışında kullanıcıların web arayüzü üzerinden e-postalarını okuma imkanı veren Zimbra[7] ve Roundcube[8] gibi yazılımlarda bulunmaktadır

Sunucular e-posta trafiğinin oluşturulması için kurulmaktadır. Ancak gelen postalara erişim için istemci uygulamalardan faydalanılmaktadır. İstemci uygulamaların en ünlüsü Thunderbird[9] yazılımıdır. Özgür ve açık kaynak kodlu olan bu yazılım aracılığı istenilen protokol kullanılarak e-posta alımı ve gönderimi yapılabilmektedir. Aynı zamanda IMAP protokolünün getirdiği klasör yapısı, filtreleme gibi özellikler de bu istemciler aracılığı ile kullanılabilmektedir.



Şekil 1.2. Web arayüzüne sahip e-posta istemcisi örneği.

Saldırganlar e-posta sunucularına saldırı gerçekleştirmek için herhangi bir ağ katmanını kullanabilmektedir. SMTP protokolü ile çalışan bir uygulama kaynaklı saldırı yapılabileceği gibi sunucu içerisinde çalışan e-posta ile hiç alakası olmayan başka bir servis aracılığı ile de saldırı gerçekleştirilebilir. Bu yüzden e-posta sunucusuna gelen trafik büyük önem taşımaktadır. Benzer şekilde bir şekilde saldırganlar tarafından ele geçirilmiş sunuculardan internete doğru oluşan trafik de inceleme açısından önemlidir. Saldırganlar çoğu zaman ele geçirdikleri e-posta sunucuları üzerinden tüm internete zararlı yazılım dağıtımı yapmaktadır. Haliyle istem dışı olarak da olsa suça ortak olunabilmektedir.

Proje içerisinde gerçekleştirilen sistem ile e-posta sunucusuna yönelik gelebilecek saldırıların tespiti, e-posta aracılığı ile gelebilecek siber tehditlerin tespiti ve e-posta aracılığı ile dışarıya çıkarılabilecek hassas verinin tespiti amaçlanmıştır. Bu yüzden gerçekleştirilen donanım üzerinden e-posta trafiği geçecek şekilde ağa dahil edilmelidir. Trafik analizinin önemi için ilk olarak saldırı tespit sistemlerinin önemi iyi kavranmalıdır.

1.2. Saldırı Tespit Sistemleri

Bilişim sistemlerinin korunması amacıyla kullanılan en önemli araçların başında saldırı tespit sistemleri(intrusion detection system)[10] gelmektedir. Bir siber saldırıyı engelleyebilmek için ilk olarak onu tespit etmek gereklidir. Bu yüzden engelleme sistemlerinin en önemli ve karmaşık kısmı tespit aşamasının yapıldığı yerdir. Buranın gelişmiş olması büyük önem taşımaktadır. Gerçekleştirilen proje de buradan yola çıkarak saldırı tespit sistemi olarak tasarlanmıştır. İyi bir saldırı tespit sistemi ortaya koyulduktan sonraki saldırı engelleme aşaması çok daha kolay olmaktadır. Örneğin ağ düzeyinde düşünülürse, bir saldırının kaynağı tespit edilirse engellemek için bu kaynaktan gelen tüm paketler düşürülebilir. Haliyle saldırı engellenmiş olunur. Burada saldırının engellenme aşamasının sadece alt seviyedeki işletim sistemi araçlarına belirli bir kural girmek olduğu anlaşılabilir.

Saldırı tespit sistemleri belirli protokoller üzerinde çalışabileceği gibi tüm ağ trafiği ile de ilgilenebilir. Örneğin özgür ve açık kaynaklı bir saldırı tespit sistemi olan Snort[11] tüm protokoller ile ilgilenmektedir. Bu sistemler genellikle imza tabanlı çalışmaktadır. İmza tabanlı sistemlerin getirdiği çeşitli zorluklar bulunmaktadır. Özellikle henüz herkes tarafından paylaşılmamış açıklıkların tespiti konusunda bu tarz sistemler çok zayıf kalmaktadır. Aynı zamanda imza veritabanının boyutunun zamanla artması da analiz işleminin süresinin artmasına sebep olmaktadır. Tüm bunlardan dolayı imza tabanlı analiz işlemleri yerini daha akıllı sistemlere bırakmaya başlamıştır. Proje çerçevesinde gerçekleştirilen geçit içerisinde hem imza veritabanı hem de çeşitli kurallar bulunmaktadır. İmza tabanlı sistemlerin zayıf özellikleri bir nebze engellenmiştir. Özellikle sürekli güncel tutma özelliği ile gelişim sağlanmıştır. Benzer şekilde kurallar ile de sadece imza tabanlı bir analiz işlemi yapılmaması sağlanmıştır.

1.3. Siber Tehditler

Günümüzde internet kullanımı arttıkça siber suçlular tarafından da yaratılan siber tehditler artmaktadır. Herhangi bir siber tehdit ile ne zaman karşılaşılacağı bilinemektedir. Bu yüzden sürekli olarak önlem alınması gerekmektedir.

Saldırganların türüne göre motivasyonları da değişmektedir. Haliyle siber suçun oluşum şekli de bu noktada farklılaşmaktadır. Örneğin oltalama[12] saldırısı gerçekleştirmek isteyen saldırgan e-posta içeriğinde gerçek bir sitenin sahtesini oluşturup adresini koymaktadır. Ancak e-posta attığı kişinin bilgisayarına erişim sağlamak isteyen kişi e-posta içerisinde ek olarak zararlı yazılım göndermektedir. Benzer olarak e-posta sunucusu üzerinde yer alan bir açıklık ile kurumsal firmanın iç ağına erişim sağlamak isteyen saldırgan, belki de hiç e-posta göndermeden sadece e-posta sunucusu ile iletişime geçmektedir. Tüm bu farklılaşmalardan dolayı siber tehditlerin engellenmesinde imza tabanlı engelleme tek başına yeterli olmamaktadır.

Proje kapsamında incelenen siber tehditler Tablo 1.1'de yer almaktadır. Buna göre her tehditin süre kapsamı bulunmaktadır. Örneğin bir saldırgan IP adresi geçicidir. Çünkü saldırganlar, saldırılarını gerçekleştirdikleri sunucuları sürekli olarak tutmakta zorlanırlar. Herhangi bir saldırıdan sonra erişim engellenmesi ile karşılaşabilirler. Bu yüzden saldırgan IP adreslerini sürekli olarak aynı şekilde değerlendirmek doğru değildir. Ancak bir zararlı dosya yıllar geçse de zararlı olarak kalacaktır. Doğru bir imza ile sürekli olarak tespit edilebilir.

Tehdit Adı	**Açıklama**	**Tehdit Süresi**
Botnet IP	Zombie IP adresleri	Geçici
Saldırgan IP	Saldırgan VPN veya kişisel IP adresleri	Geçici
Spam Sunucu	Spam e-posta gönderilen sunucular	Geçici
Spam E-Posta Adresi	Spam e-posta gönderen hesaplar	Kalıcı
Zararlı Yazılım	Zararlı dosyalar	Kalıcı
Oltalama Siteleri	Bilgi çalmaya yönelik sahte siteler	Kalıcı
Ele Geçirilmiş Siteler	Amacı dışında kullanılan siteler	Geçici

Tablo 1.1. Siber tehdit sınıflandırılması.

Siber tehditlerin proje içerisinde kullanılabilmesi için çeşitli tehdit paylaşım platformlarından faydalanılmıştır. Örneğin çok ünlü bir botnet olan *Zeus* yazılımının takibi için açılmış olan websitesi aracılığı ile açık durumdaki IP adresleri paylaşılmaktadır. Hem bu zararlı yazılımın paylaşıldığı site isimleri hem de saldırganın kurbanları yönetmek için kullandığı sunucu IP adresleri bilgiler içerisinde yer almaktadır. Burada dikkat edilmesi gereken bir nokta saldırganın kurbanları yönetmek dışında da ilgili IP adresleri üzerinden saldırılar gerçekleştirebileceğidir. Bu yüzden hem site isimleri hem de IP adresleri birlikte incelenmelidir. Örnek bir botnet bilgileri Şekil 1.3 ile görülebilir.

Dateadded	Malware	Host	IP address	Level	Status	Files Online	SBL	Country	AS number	Uptime
2015-12-21	KINS	allterrainadventures.co.uk	185.116.212.119	2	online	2	SBL280302		AS20860	13:38:45
2015-12-09	VMZeuS	emreerdem.com.tr	85.95.237.78	2	online	2	SBL278986		AS42926	303:18:21
2015-12-08	ZeuS	beautifulmindsinc.com	184.168.56.1	4	online	1	Not listed		AS26496	326:01:03
2015-10-09	VMZeuS	balcaolocal.com	131.72.52.210	2	online	2	Not listed		AS61718	838:59:59
2014-11-15	ZeuS	championbft.com	104.207.130.93	4	online	1	Not listed		AS20473	838:59:59
2014-05-29	Citadel	www.loongweed.com	202.142.215.16	2	online	1	SBL223782		AS7654	838:59:59

Şekil 1.3. Botnet IP adresleri.

1.4. Güvenli E-posta Geçidi

Siber tehditlerin yayılma türleri arasında e-postalar önemli bir yer elde etmektedir. Proje kapsamı içerisinde de sadece e-postalar üzerinden yayılan siber tehditlerin engellenmesi amaçlanmaktadır. Piyasada yer alan güvenli e-posta geçitleri genellikle spam üzerine yoğunlaşmaktadır. Özellikle büyük oyuncular sırf bu sektörü kaçırmamak için çeşitli ürünler üretmektedir. Ancak gerçek bir tehdit tespit ve engelleme sistemi sayısı ciddi derecede azdır. Güvenli e-posta geçitlerinin karşılaştırılması Şekil 1.4 ile görülebilir. Bu şekilden görüldüğü üzere Proofpoint ve Cisco sektörün büyük oyuncularıdır. Proje kapsamında incelemeler yapıldığında, bu çözümlerin yapılmak istenen güvenli geçit kavramından farklı olduğu görülmüştür. Piyasa içerisinde yer alan geçitler genellikle spam veya veri kaçakları üzerine yoğunlaşmıştır. Ancak proje içerisinde yoğunlaşılan kavram siber tehditlerdir.

Şekil 1.4. Güvenli e-posta geçidi karşılaştırmaları.

1.5. Kullanılan Teknolojiler

Proje içerisinde olabildiğince farklı ve fazla sayıda modern teknoloji kullanılmıştır. Bu derece farklı bir ortamda en büyük ortaklık tüm teknolojilerin özgür ve açık kaynaklı olmasıdır. Kullanılan donanımdan yazılıma kadar tüm birimler bu kavrama uymaktadır. Aynı zamanda teknoloji çöplüğü haline getirmek yerine sadece gerekli ve sade teknolojiler seçilmiştir. Bu yüzden hem performans hem de geliştirilebilirlik proje içerisinde çok yüksektir. Tüm bu teknolojilerin sınıflandırılması Tablo 1.2 ile görülebilir.

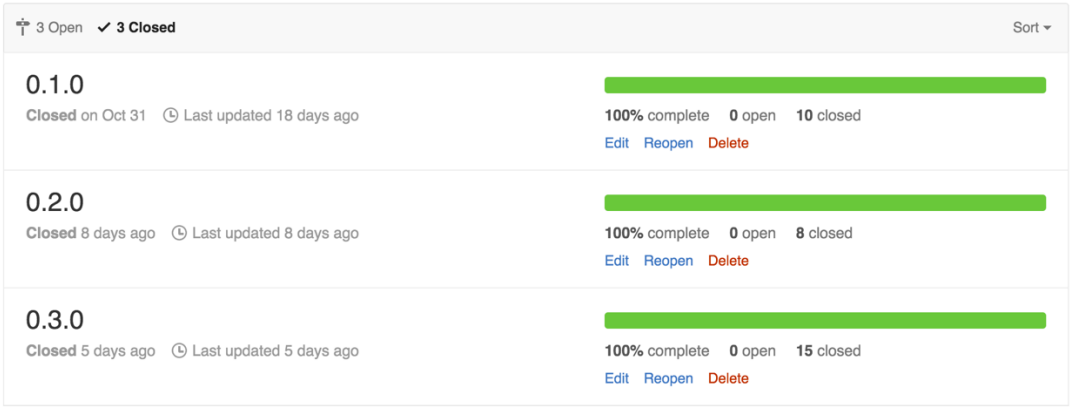
Kısım	Teknolojiler	Türü
Paket Toplayıcı	C, libnids, libuuid, libjansson	Yazılım
Güvenlik Duvarı	Pfsense	Yazılım
Takipçi	Python, pyinotify, celery	Yazılım
E-Posta Geçidi	Raspberry PI, Arch Linux	Donanım
Analiz Motorları	Python, yara, celery, rabbitmq	Yazılım
Panel	Python, django, postgresql, celery	Yazılım
Tehdit Kaynağı Arayüzü	Python, flask, mongodb, rsync	Yazılım

Tablo 1.2. Proje içerisinde kullanılan teknolojilerin sınıflandırılması.

1.6. Geliştirme Metodolojisi

Projenin geliştirilme aşamasında modern metodolojiler kullanılmıştır. Bunların başında *git*, *git flow*, *semver* ve *github* yer almaktadır. Uygulamalar geliştirilirken sürekli olarak *develop* dalı kullanılmaktadır. Geliştirilen uygulama belirli olgunluğa erişirse yeni bir sürüm *release* olarak yayınlanır ve içerik *master* dalına dahil edilir. Geliştirme sırasında kod içerisinde herhangi bir hata çözülmek isteniyorsa *hotfix* dalına geçilir ve burada çözüm gerçekleştirildikten sonra *develop* dalına tekrardan dahil edilir.

Geliştirilen yazılım belirli olgunluğa ulaştığında sürümler olarak yayına alınmaktadır. Bunun için en başta *milestone* tanımlanmaktadır. Bu tanım içerisinde çok sayıda açık *issue* bulunmaktadır. Bu konular kapatıldıkça *milestone* da son olarak kapatılmaktadır. Haliyle yeni bir sürüm yayına çıkmaktadır. Kapatılan *milestone* örnekleri Şekil 1.5 ile görülebilir.



Şekil 1.5. Kapatılan sürüm örnekleri.

Proje içerisinde sürümleme sisteminin daha rahat anlaşılabilmesi için *semver* kullanılmaktadır[13]. Haliyle eklenen yeni özellikler geçmiş ile uyumlu ise minör sürüm numarasında artış, uyumsuz ise majör sürüm numarasında bir artış gerçekleştirilmektedir. Eğer hata kapatılması ise son kısımda artış yapılır. Örneğin 0.1.0 sürümünden 0.2.0 sürümüne çıkmak yeni bir özelliği gösterirken, 0.3.0 dan 0.3.1 sürümüne geçiş bir hatanın çözüldüğünü göstermektedir.

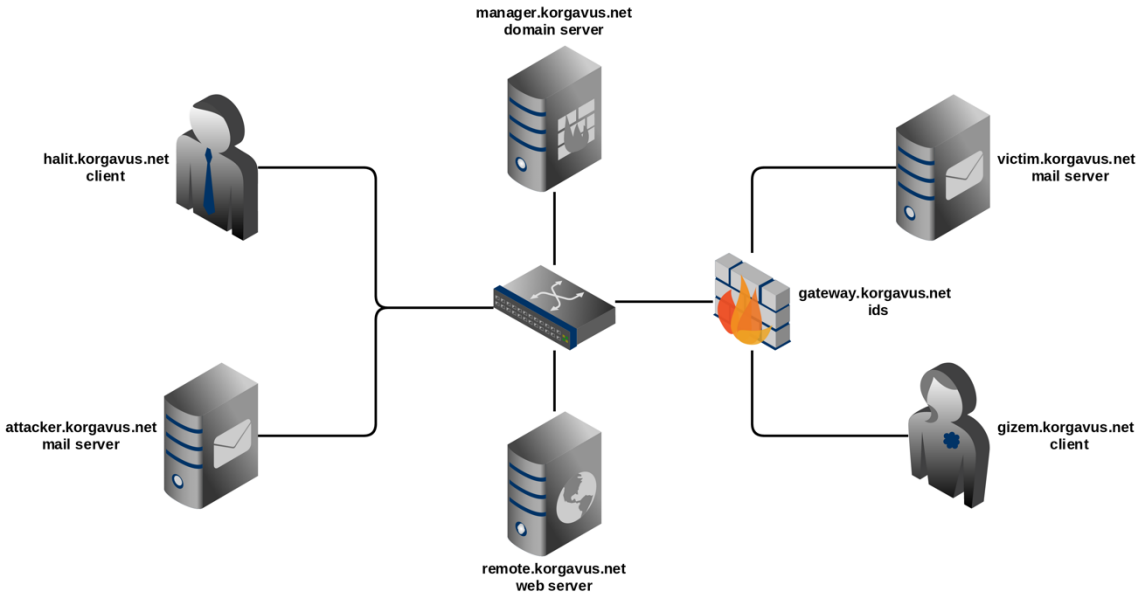
2. YAPILAN ÇALIŞMALAR

2.1 Ağ Kurulumu

Gerçekleştirilen ağ geçidinin test edilebilmesi için uygun bir ağ ortamına ihtiyaç duyulmaktadır. Bu yüzden kurumsal bir ağ yapısı kurgulanıp proje içerisinde kullanılmıştır. Ağın kurulumu *Zyxel* marka switch ile gerçekleştirilmiştir. Tek başına switch tasarlanan ağın ihtiyaçlarına cevap vermediği için yazılımsal bir çözüm olan *Pfsense* güvenlik duvarından faydalanılmıştır[14].

2.1.1 Kurumsal Ağ Tasarımı

Projenin daha gerçekçi bir ortamda test edilmesi için kurumsal bir ağ tasarlanmıştır. Bu yapı Şekil 2.1'de görülebilir.



Şekil 2.1. Tasarlanan kurumsal ağ yapısı.

Bu yapı içerisinde gerçekleştirilen geçit dışında 4 adet daha sunucuya ihtiyaç duyulduğu görülebilir. Aynı zamanda 2 adet de kullanıcı bilgisayarı ağa dahil olacaktır. Burada amaçlanan nokta saldırgan kişi kişisel bilgisayarından kendi e-posta sunucusu aracılığı ile kurban kişinin e-posta hesabına göndereceği postanın başarıyla gitmesidir. Kurban kişi de e-posta hesabını kendi kişisel bilgisayarında kurulu olan istemci aracılığı ile okuyabilir durumdadır. Burada kurban e-posta sunucusu ile kurban kişisel bilgisayarı ayrı bir ağ içerisindeymiş gibi düşünülebilir. Tamamen farklı bir ağ kurmak yerine sanal bir yapılandırma tercih edilmiştir.

Kullanılan yardımcı sunucular sanal makine oldukları için ağ adaptörleri köprü modu ile birlikte kullanılmaktadır. Şema içerisinde doğrudan switch'e bağlanan çizgiler bu yüzden tam bir kabloyu ifade etmemektedir.

Host Adı	Görevi	Açıklama
manager	Domain yöneticisi, yönlendirici	Pfsense güvenlik duvarı
gateway	Güvenli mail geçidi	Raspberry PI cihazı
attacker	Saldırgan e-posta sunucusu	Ubuntu server
victim	Kurban e-posta sunucusu	Ubuntu server
remote	Siber tehdit kaynağı sunucusu	Ubuntu server
halit	Saldırgan kişisel bilgisayarı	Macbook pro
gizem	Kurban kişisel bilgisayarı	Ubuntu

Tablo 2.1. Ağ içerisinde yer alan cihazlar.

Tasarlanan ağ içerisine dahil olacak cihazlar Tablo 2.1'de görülebilir. Bu tabloya göre 2 adet gerçek kişisel bilgisayar, 4 adet sanal makine ve 1 adet gömülü cihaz kullanılmaktadır. Sanal makinelerin hepsi Virtualbox[15] yazılımı aracılığı ile oluşturulmaktadır.

2.1.2 Güvenlik Duvarı Kurulumu

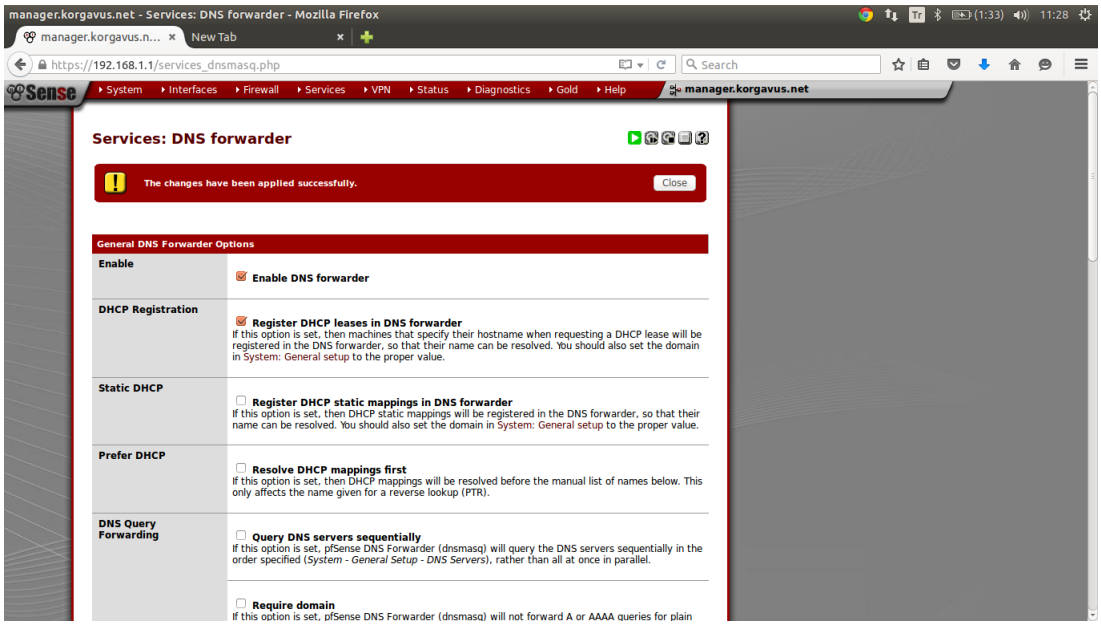
Ağ içerisinde trafiğin akışı ve internet bağlantısının gerçekleştirilmesi için *DHCP* server'a ihtiyaç duyulmaktadır[16]. Aynı zamanda gerçek bir kurumsal ağ yapısına benzemesi için domain tanımlamasının *DNS* üzerinde gerçekleştirilmesi gerekmektedir[17]. Bu ihtiyaçların sağlanması aracılığı ile ayrı bir *linux server* kurulumu yapıp tüm bu servislerin teker teker yapılandırılması gerekmektedir. Ancak proje içerisinde kolaylık sağlaması için Pfsense çözümü kullanılmıştır. Bu sayede bahsedilen tüm servisler kolayca arayüz içerisinden yapılandırılabilir olmuştur. Aynı zamanda internet bağlantısı için gerekli yönlendirmelerin ve *NAT* yapılarının da otomatik olarak gerçekleştirilmesi sağlanmıştır. Bu kurulum EK 1 içerisinde gösterilmiştir.

2.1.3 Ağ Yapılandırmaları

Kurulan ağ içerisinde alan adlarının çözümü için DNS Server yapılandırılması yapılmıştır. Buna göre DNS Server, DNS Yölendirici olarak çalıştırılmaktadır. Haliyle ağ içerisinde yapılan DNS istekleri başka bir DNS sunucusuna yönlendirilmektedir. Burada yönlendirilen sunucu olarak Google'ın kendi DNS Serverları olan 8.8.8.8 ve 8.8.4.4 ip adresleri seçilmiştir.

DNS Server'ın varlığının yanında ek olarak DHCP tarafından atanan IP ve Hostname bilgilerinin de DNS tarafından çözülmesi seçeneği aktif edilmiştir. Bu sayede ileride kullanılacak olan E-Posta sunucuları için ön bir hazırlık yapılmıştır. Çünkü doğrudan ip adresi kullanılacak olsaydı E-Postaları doğrudan IP'lere atılması veya E-Posta sunucuları üzerinde değişiklik yapılması gerekecekti. Bunun çözümü için domain isimlerinin DNS sunucusu tarafından da çözülmesi planlanmıştır.

Bu yapılandırmayı sağlamak Pfsense içerisinden gayet kolaydır. Şekil 2.2. bu durumu göstermektedir. Seçilen özellikler DNS Server'i yeniden başlattıktan sonra hemen işlenmektedir.



Şekil 2.2. DNS server yapılandırılması.

Tüm ağ yapılandırmaları sağlandıktan sonra sunucuların birbirlerine domain isimleri ile ulaşıp ulaşılmadıkları test edilmiştir. Bu durum da Şekil 2.3. üzerinden rahatlıkla görülebilir. Şekil içerisinde *pwnpi* host adına sahip sunucuya istek atılmıştır ve DNS server tarafından bu istek *pwnpi.korgavus.net* host adına çevirilip *192.168.1.102* IP adresine dönüştürülmüştür.

```
gizem@gizem-Inspiron-5521:~$ ping pwnpi
PING pwnpi.korgavus.net (192.168.1.102) 56(84) bytes of data.
64 bytes from pwnpi.korgavus.net (192.168.1.102): icmp_seq=1 ttl=64 time=0.551 m
s
64 bytes from pwnpi.korgavus.net (192.168.1.102): icmp_seq=2 ttl=64 time=0.317 m
s
64 bytes from pwnpi.korgavus.net (192.168.1.102): icmp_seq=3 ttl=64 time=0.306 m
s
64 bytes from pwnpi.korgavus.net (192.168.1.102): icmp_seq=4 ttl=64 time=0.270 m
s
64 bytes from pwnpi.korgavus.net (192.168.1.102): icmp_seq=5 ttl=64 time=0.273 m
s
64 bytes from pwnpi.korgavus.net (192.168.1.102): icmp_seq=6 ttl=64 time=0.281 m
s
64 bytes from pwnpi.korgavus.net (192.168.1.102): icmp_seq=7 ttl=64 time=0.328 m
s
```

Şekil 2.3. DNS server yapılandırmalarının sonucu.

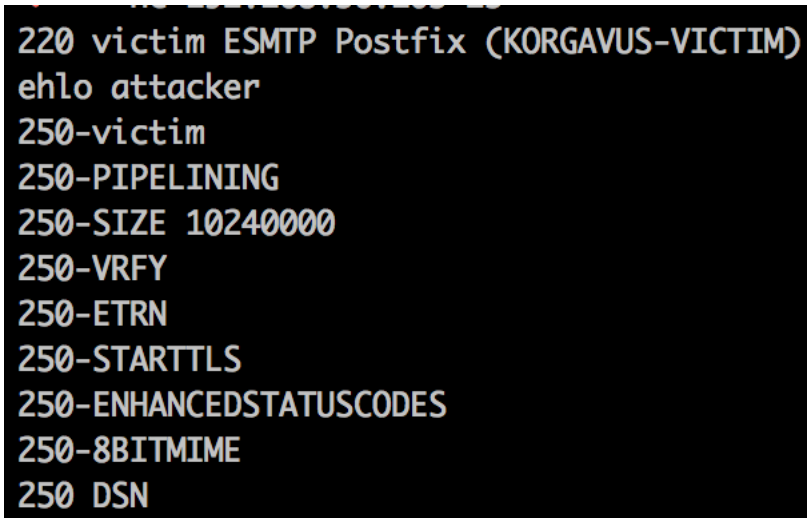
2.2 E-Posta Sunucuları

Ağ kurulumları tamamlanıp sunucuların birbirleriyle haberleşmesi kontrol edildikten sonra e-posta sunucularının kurulumuna geçilmiştir. Bu kısım sonunda amaçlanan sonuç iki e-posta sunucu arasında e-posta trafiğini başarıyla oluşturabilmektir.

2.2.1 Kurban Makinesi Kurulumu

İlk kurulan e-posta sunucusu *victim.korgavus.net* domain adına sahip kurban makinesidir. Amaçlanan yapıda bu sunucu iç kurumsal ağ tarafında kalmaktadır. Yani kurumsal bir firmanın e-posta sunucusunu nitelendirmektedir.

Kurulum sırasında *Ubuntu Server 14.04* işletim sistemi seçilmiştir[18]. Bu işletim sistemi üzerinde çalışacak şekilde *Postfix* ve *Dovecot* e-posta uygulamaları çalıştırılmaktadır. Proje içerisinde kullanılan sunucuların hepsi aynı işletim sistemine sahip olmalarına rağmen farklı konfigürasyonlar içermektedir. Bu yüzden oluşturulan sanal makine kalıbı kullanılarak diğer sunucular oluşturulmuştur. Sunucuların kurulumu Ek 2'de gösterilmektedir. Tüm bu kurulumlardan sonra sunucunun çalışır hali Şekil 2.4'ten görülebilir. Burada 25 portuna *netcat* aracı ile bağlantı gerçekleştirilmiştir[19]. *ehlo* isteğine gelen cevaba göre sunucunun başarıyla kurulduğu anlaşılabilir.



Şekil 2.4. Kurban sunucusu kurulumunun tamamlanması.

2.2.2 Saldırgan Makinesi Kurulumu

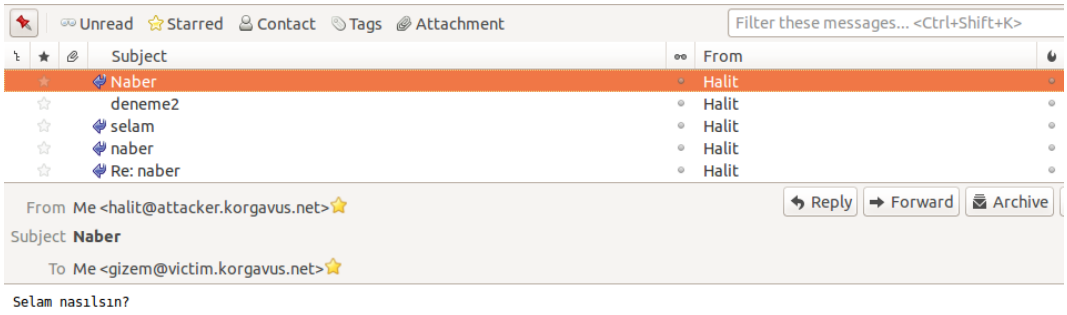
Saldırgan makinesi de Kurban makinesine benzer şekilde Ek 2'ye dayanılarak oluşturulmuştur. Sadece sunuculara göre ufak bazı ayarlamalar yapılmıştır. Örneğin e-postaların kabulü için /etc/postfix/main.cf içerisinde Şekilde 2.5'te görülen hedef değişiklikleri yapılmıştır.

Yapılan tüm ayarlamalardan sonra artık ağ içerisinde iki adet çalışır durumda e-posta sunucusu elde edilmiştir. Bu sunucuların testi için *Thunderbird* istemci uygulaması kullanılmıştır. Bir e-posta adresinden diğer adrese gönderim yapılıp test edilmiştir. E-posta hesaplarının oluşturulması için de kolaylık olması için sunucu içerisinde yer alan kullanıcılar seçilmiştir.

Yani halit@attacker.korgavus.net e-posta adresi için *attacker.korgavus.net* sunucusu içerisinde *halit* isimli bir kullanıcı oluşturulmuştur.

```
# See /usr/share/doc/postfix/TLS_README.gz in the postfix-doc package for
# information on enabling SSL in the smtp client.

smtpd_relay_restrictions = permit_mynetworks permit_sasl_authenticated defer_una
uth_destination
myhostname = victim
alias_maps = hash:/etc/aliases
alias_database = hash:/etc/aliases
myorigin = /etc/mailname
mydestination = victim.korgavus.net, victim, localhost.localdomain, localhost
relayhost =
mynetworks = 127.0.0.0/8 [::ffff:127.0.0.0]/104 [::1]/128, 192.168.1.0/24
mailbox_command = procmail -a "$EXTENSION"
mailbox_size_limit = 0
recipient_delimiter = +
inet_interfaces = all
inet_protocols = all
victim@victim:/etc/postfix$
```

Şekil 2.5. E-posta sunucu ayarlamaları.



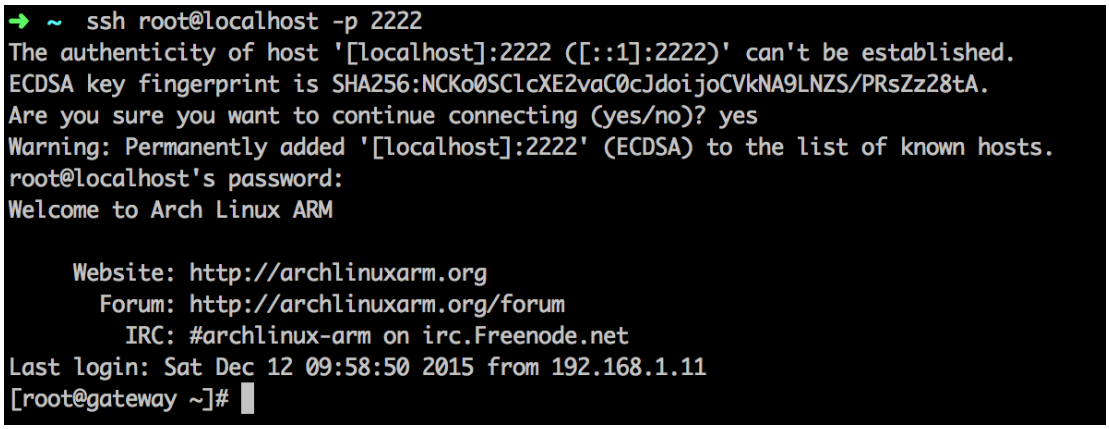
Şekil 2.6. E-posta gönderim testi.

2.3 Geçit Kurulumu

Proje içerisinde güvenli ağ geçidi gerçekleştirilmesi planlanmaktadır. Bu kapsamda cihazlar arasında en önemli yeri geçit elde etmektedir. Bu geçit trafiği bir arayüzünden alıp diğer arayüzüne aktaracaktır. Aktardığı trafik üzerinden de sadece e-postaları toplayacak ve analiz edecektir.

2.3.1 İşletim Sistemi Kurulumu

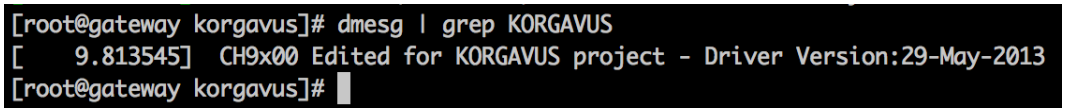
Geçit projenin en çok işlem yapan birimi olduğu için kurulumlarda en çok performans gerektiren kısımdır. Bu yüzden işletim sistemi olarak *Arch Linux ARM* seçilmiştir[20]. Bu sayede olabilecek en sade sistem elde edilmiştir. Paket yöneticisinin stabil çalışması ve paket çeşitliliği ile de projenin başarı olmasında büyük rol oynamıştır. İşletim sisteminin tüm kurulum aşaması EK 3 kısmında gösterilmiştir. Kurulan sisteme yapılan ilk *SSH bağlantısı* da Şekil 2.7'de gözükmektedir. Burada bağlantı *localhost* üzerine yapılmış gibi anlaşılabilir. Ancak bu durum ağ içerisindeki başka bir sunucu üzerinden *SSH tüneli* yapılmış olmasındandır. Çünkü geliştirilme aşamasında bazen tüm ağ kurulmadan da erişişim yapılma gerekmiştir. 2222 portuna gelen istekler geçidin 22 portuna yönlendirilmektedir.



Şekil 2.7. Geçide yapılan ilk SSH bağlantısı.

2.3.2 Trafik Köprüleme İşlemi

İşletim sistemi kurulan geçit üzerinden trafiğin yönlendirilebilmesi için harici bir ethernet adaptörüne ihtiyaç duyulmuştur. Bunun çözümü için internetten ucuz bir şekilde temin edilebilen *usb2ethernet* aracı kullanılmıştır. Haliyle bu araç geçit içerisine ikincil ağ adaptörü olarak dahil edilmiştir. İşletim sisteminin bu ağ adaptörünü tanıması için başka bir cihaz için gerçekleştirilmiş sürücü kullanılmıştır. Bu kullanım için ilk olarak *Linux* çekirdeğinin başlık bilgileri indirilmiştir. Daha sonra gerekli bazı kütüphaneler kurulmuştur. Tüm bu yapılanlardan sonra ethernet sürücüsü başarıyla kurulmuştur. Kurulumun sistem logları içerisinde yer alması Şekil 2.8. ile görülebilir.



Şekil 2.8. Ethernet sürücüsünün başarıyla kurulumu.

İkincil ağ adaptörü başarıyla kurulup IP adresi alması sağlandıktan sonraki aşama trafiği köprülemektir. Bunun için *bridge-utils* isimli araç kurulmuştur[21]. Bu sayede köprülenmek istenen trafik işlemi kolaylaşmıştır. İlk olarak *bridge0* isimli köprü arayüzü oluşturulmuştur. Daha sonra *eth0* ve *eth1* arayüzleri bu köprüye eklenmiştir. Haliyle bir arayüzden gelen paketler diğer arayüze geçebilecektir. Son olarak da köprü arayüzüne *DHCP* üzerinden *IP* atanması yapılmıştır.

```
brctl addbr bridge0
brctl addif bridge0 eth0
brctl addif bridge0 eth1
ifconfig eth0 0.0.0.0
ifconfig eth1 0.0.0.0
ifconfig bridge0 up
dhcpcd bridge0
```

Tablo 2.2. Trafik köprüleme işlemi için kullanılan Linux komutları.

2.4 Paket Toplayıcı

Geçidin, üzerinden geçen trafiği köprülemesinden sonra yapılması gereken bu trafiğin toplanmasıdır. Bu amaçla paket toplayıcı yazılımı geliştirilmiştir. Bu kısım yüksek hesaplama ve fazla kütüphane çağrısı gerektirdiğinden *C* programlama dili ve *libnids* kütüphanesi ile birlikte geliştirilmiştir[22].

2.4.1 Yardımcıların Oluşturulması

Paket toplayıcı uygulamasının gerçekleştirilmesine başlamadan önce ilk olarak derleme işlemlerini kolaylaştıran *makefile* dosyası oluşturulmuştur. Bu dosya sayesinde hem hızlı bir şekilde derleme hem de gerekli kütüphanelerin bağlanması kolaylaşmıştır. Şekil 2.9'dan görüldüğü gibi *libnids*, *libjansson* ve *libuuid* kütüphaneleri araç içerisinde kullanılmaktadır[23][24].

```
TARGET = sniffer

SRC_EXT = .c
OBJ_EXT = .o
HDR_EXT = .h

SRC_DIR = src

COMPILER = ${CC}
COMPILER_FLAGS = ${CFLAGS} -lnids -ljansson -luuid -Wall

SRC = $(shell ls ${SRC_DIR}/*${SRC_EXT})
HDR = $(shell ls ${SRC_DIR}/*${HDR_EXT})
OBJ = $(shell ls ${SRC_DIR}/*${SRC_EXT} | sed s\/${SRC_EXT}$$\/${OBJ_EXT}\/)

all: ${OBJ}
    ${COMPILER} -o ${TARGET} ${OBJ} ${COMPILER_FLAGS}

%${OBJ_EXT}: %${SRC_EXT}
    ${COMPILER} ${COMPILER_FLAGS} -c $< -o $@

clean:
    @if `test -e ${TARGET}`; then \
      rm ${TARGET}; \
    fi

    @for file in ${OBJ}; do \
      if `test -e $$file`; \
        then rm $$file; \
      fi \
    done
```

Şekil 2.9. Yardımcı derleme aracı makefile içeriği.

Paket toplayıcının başlık bilgileri için *korgavus.h* isimli bir dosya oluşturulmuştur. Bu dosya hem paket toplayıcının kullandığı bağımlılıkları hem de analiz işlemi için gerekli olacak veri yapılarının tanımlamalarını içermektedir. Aynı zamanda paket toplama işini yapacak fonksiyon prototipleri de bu dosya içerisinde bulunmaktadır. Kirlilik yaratmamak adına dosyanın tüm içeriği gösterilmemiştir. Ancak bağımlılıkların eklenme kısmı Şekil 2.10 ile görülebilir.

```c
/*
 * Include requirement dependencies
 */
#include <stdlib.h>
#include <stdio.h>
#include <uuid/uuid.h>
#include <jansson.h>
#include <string.h>

/*
 * Define constants
 */
static const char* USAGE_TEXT = "Usage: korgavus [t]\n";
static const char* MAIL_DIR = "/korgavus/mails/queue";
static const int DEFAULT_STATE_TABLE_SIZE = 4096;
typedef enum {MEMORY_ALLOCATION, NULL_OBJECT_DESTROY, FILE_OPEN_ERROR} error_t;
```

Şekil 2.10. Gerekli bağımlılıklar ve sabitlerin tanımı.

Geliştirilen paket toplayıcının çalışma yapısı gereği bazı veri yapıları tanımlanmıştır. Bunlardan ilki mail içeriği tutacak olan *mail_t* ve durum tablosunun elemanlarını tutacak olan *state_t* yapılarıdır. Paket toplayıcı üzerinden geçen trafiği analiz ederek e-postaları ayıklamaktadır. Ayıkladığı e-postaların içeriğini ve durum tablosunu ayrıca tutmaktadır. Bu sayede üst kısımda yapılabilecek detaylı analizler için alt yapı oluşturmaktadır. Örneğin üst kısımda yer alacak analiz aracı artık e-postayı bir saldırganın mı yoksa normal bir kullanıcının mı göndermiş olduğunu durum tablosuna bakarak söyleyebilecektir. Bu veri yapıları Şekil 2.11 ile gösterilmiştir.

```c
/*
 * State table item
 *
 * @stream_data: user input data
 * @stream_size: size of data
 */
typedef struct{
    char* stream_data;
    unsigned long stream_size;
}state_t;

/*
 * Korgavus mail file format
 * State table hold all of the user inputs
 *
 * @source_port: source port number
 * @dest_port: destination port number
 * @source_ip: source ip address
 * @dest_ip: destination ip address
 * @state_table: state table of user inputs
 * @state_table_size: size of state table
 * @max_state_table_Size: maximum size of the state table
 * @created_time: first packet time
 * @destroyed_time: last packet time
 */
typedef struct{
    unsigned short source_port;
    unsigned short dest_port;

    unsigned long source_ip;
    unsigned long dest_ip;

    state_t** state_table;
    unsigned long state_table_size;
    unsigned long max_state_table_size;
}mail_t;
```

Şekil 2.11. E-Posta ve durum tablosu veri yapıları tanımlamaları.

Şekil 2.11'den de görüldüğü üzere durum tablosu *state_table* değişkeni ile dinamik bir dizi olarak tanımlanmıştır. Haliyle e-posta sunucusuna gönderilen *SMTP* trafiğinin boyutu sınırsız olabilmektedir. Eğer burası statik olarak tanımlanmış olursa saldırganlar ilgili statik boyutun fazlası kadar istek oluşturup güvenlik sistemini atlatabilecektir. Ancak dinamik tanımlanması bu durumu ortadan kaldırmaktadır.

2.4.2 Paket Toplayıcının Gerçekleştirilmesi

Yardımcı kısımların tamamlanmasından sonra asıl paket toplama işini yapan *sniffer.c* kodu yazılmıştır. Bu kod içerisinde tüm paket toplayıcının *main* fonksiyonu bulunmaktadır. Haliyle paket toplamanın başlangıcı buradadır.

Paket toplama işlemi içerisinde çeşitli tanımlamalar yapılmıştır. *libnids* kütüphanesinin parametreleri ile sadece 25 nolu port ve daha önce köprülenmiş olan ağ arayüzü üzerinde paket toplaması yapılması belirtilmiştir. Haliyle projenin bu kısmı sadece *SMTP* paketleri ile ilgilenecektir. Haliyle geçit üzerinden geçen diğer trafik ile ilgilenmemektedir. Bu tanımlamalar Şekil 2.12 ile görülebilir.

```
/*
 * Close, port scanning feature
 * Set, pcap filter to SMTP
 * Set, device to bridge0
 */
nids_params.scan_num_hosts  = 0;
nids_params.tcp_workarounds = 1;
nids_params.pcap_filter     = "tcp port 25";
nids_params.device          = "bridge0";

/*
 * Init libnids
 */
if (!nids_init()){
    fprintf(stderr,"%s\n", nids_errbuf);
    return 1;
}

/*
 * Don't calculate checksum
 */
ctl.netaddr = 0;
ctl.mask    = 0;
ctl.action  = NIDS_DONT_CHKSUM;
nids_register_chksum_ctl(&ctl, 1);

/*
 * Register callback function and enter to main loop
 */
nids_register_tcp(tcp_callback);
nids_run();
```

Şekil 2.12. Libnids kütüphanesinin çalıştırılması ve hataların düzeltilmesi.

İlgili kütüphanenin eski ve örnek kod bulmanın zorluğundan kaynaklı olarak çeşitli zorluklar ile karşılaşılmıştır. Bunların en başında internet üzerinde çalışan kod parçası bulamamaktır. Dikkatli bir şekilde kodlar incelendiğinde ve çeşitli yazılar üzerinden araştırma yapıldığında bunun değişen ağ teknolojileri ile ilgili olduğu görülmüştür. Sorunun çözümü için *libnids* kütüphanesinin kontrol parametresine *NIDS_DONT_CHKSUM* yani paketler üzerinde herhangi bir doğrulama işlemi yapma eklenmiştir.

```
case NIDS_DATA:
    mail_p = *mail_save_p;
    struct half_stream* hlf;

    /*
     * If we received new server data
     * Create new state table item
     * Then append new item to table
     */
    if (a_tcp->server.count_new){
        hlf = &a_tcp->server;
        state_t* new_state = init_state_item(hlf->count_new+1);
        strncpy(new_state->stream_data, hlf->data, hlf->count_new);
        append_state(mail_p, new_state);
    }
    break;
```

Şekil 2.13. Paket toplama aşamasında geçit üzerine gelen ilk pakedin işlenmesi.

Geçit üzerinden geçen paketler toplanırken *TCP* protokolünün durumlarına göre analiz edilmektedir. İlk paketin atılmasından en son pakedin atılmasına kadar geçen tüm süreç incelenebilmektedir.

E-posta geçit üzerinden geçerken ilk olarak bağlantı kurma talebinde bulunmaktadır. Bu talebi alan paket toplayıcı bellek üzerinde durum tablosunu ve diğer veri yapılarını oluşturmaktadır. Bağlantının kurulmasından sonra sunucuya veri iletimi gerçekleştirmektedir. Bu iletimin bitmesi sırasında ise bellekte toplanan veriler veri yapıları içerisine eklenmektedir. Tüm bu durumlar Şekil 2.13 ile görülebilir.

Veri iletim işlemi bittiğinde ise paket toplayıcı artık bu e-posta için hiçbir şekilde benzeri olmayacak biricik bir dosya yolu üretmektedir. Bu dosya yolu içerisine de mail *json* veri tipi ile çıkartılmaktadır. Aynı zamanda sistem dinamik oluşturulan tüm bellek alanlarını işletim sistemine iade etmektedir. Haliyle bellek kaçakları oluşmamaktadır. Bu kısım Şekil 2.14 ile görülebilir.

```
case NIDS_CLOSE:
    mail_p = *mail_save_p;

    /*
     * Create unique path and write mail object to it
     */
    char* unique_path = create_unique_path();
    save_to_file(unique_path, mail_p);

    /*
     * Destroy allocated memory regions
     */
    destroy_mail_type(mail_p);
    free(unique_path);
    break;
```

Şekilde 2.14. E-postanın analiz için kaydedilmesi.

Paket toplayıcısı kendisine belirtilen yola biricik bir adres ile birlikte e-postayı *json* olarak çıkarttıktan sonra görevini analiz aracına bırakmaktadır[25]. Tüm işlemler sonucunda örnek olarak dosyaya kaydedilmiş bir e-posta içeriği Şekil 2.15 ile görülebilir.


```
{'dest_ip': 1795270848,
 'dest_port': 25,
 'source_ip': 1761716416,
 'source_port': 45210,
 'state_table': [{'stream_data': 'EHLO victim\r\n', 'stream_size': 14},
                 {'stream_data': 'MAIL FROM:<gizem@victim.korgavus.net> SIZE=566\r\nRCPT
TO:<halit@attacker.korgavus.net> ORCPT=rfc822;halit@attacker.korgavus.net\r\nDATA\r\n',
                  'stream_size': 135},
                 {'stream_data': 'Received: from [192.168.1.104] (gizem.korgavus.net [192
.168.1.104])\r\n\tby victim (Postfix) with ESMTP id 1F2762208B\r\n\tfor <halit@attacker.k
orgavus.net>; Tue, 15 Dec 2015 13:02:06 +0200 (EET)\r\nTo: Halit <halit@attacker.korgavus
.net>\r\nFrom: Gizem <gizem@victim.korgavus.net>\r\nSubject: jj\r\nMessage-ID: <566FF32F.
9080604@victim.korgavus.net>\r\nDate: Tue, 15 Dec 2015 13:02:07 +0200\r\nUser-Agent: Mozi
lla/5.0 (X11; Linux x86_64; rv:38.0) Gecko/20100101\r\n Thunderbird/38.4.0\r\nMIME-Versio
n: 1.0\r\nContent-Type: text/plain; charset=utf-8\r\nContent-Transfer-Encoding: 7bit\r\n\
r\njj\r\n.\r\nQUIT\r\n',
                  'stream_size': 576}]}
```


Şekil 2.15. Örnek e-posta içeriği.

2.5 Analiz Aracı

Geçit üzerinden geçen paketler birer *json* dosyası olarak kaydedilmektedir. Paket toplayıcı bu görevini tamamladıktan sonra ilgili e-postaların analiz edilmesi gerekmektedir. Bu analiz işlemi hem kural tabanlı hem de imza tabanlı olarak gerçekleştirilmektedir. Aynı zamanda yeni bir analiz motorunun eklenmesi yapısı gereği çok kolaydır.

2.5.1 Analiz Motorlarının Gerçekleştirilmesi

Analiz işlemi kısmından *yara* kütüphanesinden faydalanılmıştır[26]. Bu sayede imza tabanlı analiz işlemi büyük oranda kolaylaşmıştır. İlgili kütüphanenin alt seviye araçlar ile gerçekleştirilmesi ve paralellik konusunda büyük gelişimler içermesi projenin bu kısmının performanslı olmasına sebep olmuştur. Örnek bir yara kuralı şekil 2.16 ile görülebilir. Bu kural içerik içerisinde ilgili IP adresi ile karşılaştığı zaman geriye tespit edilme bilgisini dönecektir.

```
rule ip_signature_156
{
    meta:
        description = "malicious"

    strings:
        $ip = "94.103.36.55"

    condition:
        $ip
}
```

Şekil 2.16. Örnek yara kuralı.

Analiz işlemi iş kuyrukları tarafından yönetilecek olmasına rağmen çalıştırılan her analiz motoru paralel olarak çalıştırılmaktadır. Bu yüzden *celery* kütüphanesinin *multiprocess* işlemler için oluşturulan *billiard* modülünden faydalanılmıştır[27]. Tüm analiz motorları bir işletim sistemi süreci olarak oluşturulmaktadır. Bu işlemler için oluşturulan ana sınıf Şekil 2.17 ile görülebilir. Kirlilik yaşanmaması için tüm kod bloğu gösterilmemiştir. Ancak tüm analiz işleminin başlatıldığı *run* metodu gösterilmiştir.

```
class Engine(billiard.Process):
    """
    Analyze Engine Base Class
    """
    NAME = ''

    def __init__(self, rule_db, content, queue, name, field):
        """
        Constructor method

        :param rule_db: file path of yara rule db
        :param content: analysis data
        :param queue: thread-safe queue
        :return: None
        """
        billiard.Process.__init__(self)

        self.rule_db = rule_db
        self.content = content
        self.queue = queue
        self.name = name
        self.field = field

    def run(self):
        """
        Runner method of process

        :return: result in the queue
        """
        yara_result = self.rule_db.match(data=self.content)

        if yara_result:
            yara_result = yara_result[0]
            engine_result = {'name': self.name, 'status': True, 'rule': yara_result.rule,
                             'meta': yara_result.meta, 'field': self.field}
        else:
            engine_result = {'name': self.name, 'status': False, 'field': self.field}

        self.queue.put(engine_result)
```

Şekil 2.17. Analiz motoru taban sınıfı.

Analiz işlemleri projenin en can alıcı noktasını oluşturmaktadır. Bu yüzden dikkatlice geliştirilmiştir. Geliştirilebilirliğe açık olması için sınıfların miraslar yoluyla türetilmesinden faydalanılmıştır. Yani sistem içerisine yeni bir analiz motoru eklemek sadece var olan sınıfları türetip, ilgili metodlar üzerinde yeni işlemler tanımlama olarak düşünülebilir. Burada analiz motoru diye kastedilen sadece arka planda yara imzaları çalıştıran kısımlar değildir. Bu imzalar dışında çeşitli kuralların da kontrolü gene benzer şekilde gerçekleştirilmektedir. Analiz motorlarının yönetimi içerisinde hangi motorun çağrıldığı bilinebilmektedir. Aynı zamanda sonuçlar da benzer içerikte geriye dönmektedir.

```python
class StateTableAnomalyPolicy(Engine):
    """
    State Table Anomaly Policy
    """
    NAME = 'state_table_anomaly_policy'

    def run(self):
        """
        Runner method of process
        :return:
        """
        unvalids = filter(lambda x: not x['is_valid'], self.content)
        if unvalids:
            engine_result = {'name': self.name, 'status': True, 'field': self.field}
        else:
            engine_result = {'name': self.name, 'status': False, 'field': self.field}
        self.queue.put(engine_result)
```

Şekil 2.18. Örnek analiz kuralı.

Örnek bir analiz kuralı Şekil 2.18 ile görülebilir. Burada dikkat edilmesi gereken nokta arka planda herhangi bir *yara* imzasının çağrılmamasıdır. Haliyle bu analiz motoru artık bir analiz kuralı işlemektedir. Örnekteki kural, durum tablosu içerisinde geçerli olmayan bir *SMTP* komutunun olup olmadığını kontrol etmektedir. Eğer böyle bir durum ile karşılaşılırsa geriye dönen cevap içerisinde *status* kısmı ile bu belirtilmektedir. Görüldüğü üzere yeni bir analiz motoru geliştirmek ciddi derecede kolaydır. Yapılması gereken sadece yeni bir sınıf türetmek ve bu sınıfın *run* metodunu yeniden yazmaktır.

Analiz motorları bahsedildiği üzere türetilerek oluşturulmaktadır. Ancak bu motorların takipçi kısmı tarafından kolayca çağrılması ve asenkron bir şekilde çalıştırılabilmesi analiz motoru yöneticisi oluşturulmuştur. Çağrılmak istenen analiz motorları ve içerikler bu yöneticiye verilmekte ve daha sonra sonuçların oluşması beklenilmektedir. Haliyle asenkron çalışan işlemlerin sonuçları beklenilmektedir.

Tüm analiz motorları asenkron olarak çalışmaya başlamaktadır. Aynı zamanda her bir analiz motoru içerisinde *yara* imzaları da kendi içerisinde birçok süreç çalıştırmaktadır. Haliyle ulaşabilecek en iyi hıza erişmek için gerekli tüm çalışmalar yapılmıştır.

Analiz motorları çalışmaya başlarken içerik, analiz motoru ismi ve özel bilgi almaktadır. Daha sonra analiz motoru yöneticisi motor ismine bakarak ilgili sınıftan bir örnek oluşturmakta ve içerik bilgisini bu sınıfın *run* metoduna vermektedir. Burada ek olarak verilen özel bilgi ilgili motorun bir imza mı yoksa kural mı olduğunu belirtmektedir. Çünkü imza tabanlı motorlar arka planda *yara* uygulamasını çalıştırırken, kural tabanlı motorlar ise normal *python* kodları ile işlemlerini gerçekleştirmektedir[28]. Aynı zamanda kuralların da içerisinde *yara* kuralları çalıştırılabilmektedir. Bu amaçla da metoda verilen son *boolean* değeri ilgili kural motorunun bir imza veritabanı kullanıp kullanmadığını belirtmektedir.

```
def run(self, name, content, scan_type, scan_field, policy_db=True):
    """
    Run engine as a process with the content

    :param name: engine name
    :param content: analysis data
    :param scan_type: scan type
    :param scan_field: scan field
    :param policy_db: policy db
    :return: None
    """
    engine_cls = self.engines[name]

    if scan_type == 'S':
        rules_path = korgavus_config['analyzer']['signature_path']
    else:
        rules_path = korgavus_config['analyzer']['policy_path']

    if policy_db:
        rule_path = "{}.compiled".format(os.path.join(rules_path, name))
        engine_rules = yara.load(rule_path)
    else:
        engine_rules = None

    engine_ins = engine_cls(engine_rules, content, self.queue, name, scan_field)
    engine_ins.start()

    self.active_engines.append(engine_ins)
```

Şekil 2.19. Analiz motoru yöneticisi run metodu.

Analiz motoru yöneticisinin sadece *run* metodu Şekil 2.19 ile görülebilir. Bu metodun tanımlanmasından da anlaşıldığı üzere çeşitli parametrelere göre sınıflar seçilmekte ve bunların *start* metodları çağrılmaktadır. Burada *start* metodunun çağrılma sebebi ilgili sınıfların birer işletim sistemi süreci olarak değerlendirilmesidir.

Analiz motorlarının çalışırken kullanması gereken bazı ayar değerlerine rahat erişim sağlanabilmesi için ayar yönetici sınıfı oluşturulmuştur. Bu sınıf aracılığı ile ayarlara erişim kolay bir hal almıştır. İlgili sınıfın bir kısmı Şekil 2.20'de yer almaktadır.

```
class ConfigManager(collections.MutableMapping):
    """
    Korgavus config manager
    """

    def __init__(self, config_file=KORGAVUS_CONFIG_FILE, *args, **kwargs):
        """
        Constructor method of ``ConfigManager``

        :param config_path: config file path
        """
        self.store = {}
        self.config_file = config_file
        self.update(dict(*args, **kwargs))
        self.parser = SafeConfigParser()
        self.parser.read(self.config_file)

        for section in self.parser.sections():
            self.update({section: dict(self.parser.items(section))})

    def __getitem__(self, key):
        """
        Get the item from store
        """
        return self.store[key]

    def __setitem__(self, key, value):
        """
        Set the item to store
        """
        self.store[key] = value

    def __delitem__(self, key):
        """
        Del the item from store
        """
        del self.store[key]
```

Şekil 2.20. Ayar yöneticisi sınıfı.

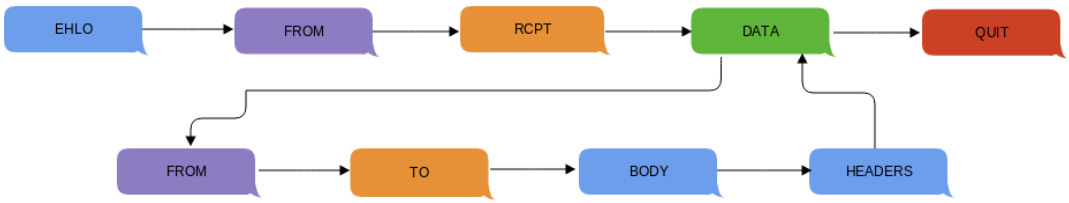
2.5.2 E-Posta Ayrıştırıcısı

Paket toplayıcı topladığı e-posta trafiğini dosya olarak dışarı çıkartmaktadır. Analiz motorları da bu dosya içeriğini okuyarak analiz işlemi yapmaktadır. Ancak analiz motorlarının analiz işlemini yapabilmesi için ilgili paket trafiğinin ayrıştırılması gerekmektedir. Çünkü paket toplama işlemi sırasında sadece tüm trafik ham olarak alınmaktadır. Bu ham trafiği *SMTP* trafiği haline dönüştürmek için ayrıştırıcı yazılmıştır. Ayrıştırıcının ufak bir kısmı Şekil 2.21'de görülebilir.

```
state_list = []
last_state = 'START'

for line in splitted_text:
    line_splitted = line.split()
    len_line_splitted = len(line_splitted)
    if len_line_splitted:
        verb = line_splitted[0].upper()
        args = line_splitted[1:] if len_line_splitted > 1 else ''

        if last_state == 'DATA' and verb != 'QUIT':
            continue

        is_valid = verb in STATE_TABLE

        state_list.append({'verb': verb, 'args': args, 'is_valid': is_valid})
        last_state = verb

data = R_DATA.findall(full_text)
if data:
    data = data[0]

parsed = mime.from_string(str(data))

result = {'state_list': state_list, 'mail': parsed}
other = {key: self.stream[key] for key in ('source_port', 'dest_port', 'source_ip', 'dest_ip')}
other['source_ip'] = int_to_ip(other['source_ip'])
other['dest_ip'] = int_to_ip(other['dest_ip'])
result.update(other)
```

Şekil 2.21. Ayrıştırma işlemi yapan fonksiyon içeriği.

Ayrıştırma işlemi sırasında düzenli ifadelerden faydalanılmaktadır. Bu düzenli ifadeler paketin geçerli bir *SMTP* trafiği olup olmadığının kontrolünü de gerçekleştirmektedir. Bu kontrolü geçen paket içerikleri daha sonra durum tablolarına dönüştürülmektedir. Dönüşüm işlemi sırasında da düzenli ifadelerden faydalanılmaktadır. Gerçekleştiren işlem analiz işlemi için büyük önem taşımaktadır. Özellikle durum tablosunun üst kısımda yer alan analiz motorlarına ulaştırılması daha karmaşık analiz işlemlerinin yapılmasına olanak sağlamaktadır. Olması beklenen geçerli bir e-posta trafiği Şekil 2.22 gibi gösterilebilir.



Şekil 2.22. Geçerli bir e-posta trafiği.

2.6 Takipçi

Proje içerisinde tüm kısımlar birbirlerinden tamamen farklı şekilde modüler geliştirilmiştir. Örneğin paket toplama kısmı sadece paket toplamaktadır ve analiz ile ilgili herhangi bir iş gerçekleştirmemektedir. Analiz kısmı da benzer şekilde elindeki veriyi analiz edip sonucu panele iletmektedir.

Paket toplayıcı topladığı paketleri *json* dosyası olarak dışarıya çıkartmaktadır. Bu dosyaların takibi için takipçi uygulaması gerçekleştirilmiştir. Bu kısım belirtilen dizin içerisine yeni bir dosyanın eklenip eklenmediğini takibini gerçekleştirmektedir. Eğer eklenme olursa analiz işlemini de başlatmaktadır. Haliyle sürekli olarak çalışması gereken kritik bir noktadır.

2.6.1 Takipçi Uygulamasının Gerçekleştirilmesi

Takipçi uygulaması *pyinotify* kütüphanesi aracılığı ile dizin içerisine yeni bir paket gelip gelmediğinin kontrolünü yapmaktadır[29]. Yeni bir paket geldiği anda iş kuyruğuna analiz işlemi ile birlikte gönderilmektedir. Bu durum Şekil 2.23 ile görülebilir. Burada işletim sisteminin çağrıları aracılığı ile ilgili dizin içerisinde yeni bir dosyanın oluşturulduğu tespit edilmektedir. Hangi dizinin takip edileceği bilgisi de ayar yöneticisi aracılığı ile ayar dosyası içerisinden alınmaktadır.

Gerçekleştirilen uygulama sürekli olarak bir döngü içerisinde dizini takip etmektedir. Takip ettiği anda yeni bir dosya oluşursa *parse_and_analyze* isimli iş çalıştırılacaktır. Bu iş *celery* aracılığı ile *rabbitmq* iş kuyruğu üzerinde asenkron olarak gerçekleştirilecektir. Bu kuyruk türünün seçimindeki en önemli faktör, herhangi bir durumda sunucu kapanırsa kuyruğa itilen işlerin kaldığı yerden devam edebilmesidir. Özellikle proje içerisinde e-posta analizi yapıldığı için bir e-postanın kaybedilmesi büyük risk taşımaktadır. Bu yüzden olası tüm durumlar kontrol edilmektedir.

```
class EventHandler(pyinotify.ProcessEvent):
    def process_IN_CREATE(self, event):
        changed_file = event.pathname
        parse_and_analyze.delay(changed_file)

if __name__ == '__main__':
    wm = pyinotify.WatchManager()
    mask = pyinotify.IN_CREATE

    handler = EventHandler()
    notifier = pyinotify.Notifier(wm, handler)

    wm.add_watch(korgavus_config['watcher']['queue_mail'], mask)
    notifier.loop()
```

Şekil 2.23. Yeni paketlerin takibi.

```
try:
    analyze_manager.run('ip_engine', parse_result['source_ip'], 'S', 'source_ip')
    analyze_manager.run('ip_engine', parse_result['dest_ip'], 'S', 'dest_ip')

    received_s = mail.headers.get('Received', '')
    if received_s:
        analyze_manager.run('ip_engine', received_s, 'S', 'received_header')

    from_s = mail.headers.get('From', '')
    if from_s:
        analyze_manager.run('mail_engine', from_s, 'S', 'from_header')
        analyze_manager.run('spoof_mail_policy', (from_s, from_mail), 'P', 'spoof_mail', False)

    to_s = mail.headers.get('To', '')
    if to_s:
        analyze_manager.run('mail_engine', to_s, 'S', 'to_header')

    analyze_manager.run('mail_engine', from_mail, 'S', 'from_mail')
    analyze_manager.run('mail_engine', to_mail, 'S', 'to_mail')
    analyze_manager.run('state_table_anomaly_policy', parse_result['state_list'], 'P', 'state_list', False)
    analyze_manager.run('mail_engine', mail.body, 'S', 'mail_body')
    analyze_manager.run('ip_engine', mail.body, 'S', 'ip_body')
    analyze_manager.run('domain_engine', mail.body, 'S', 'domain_body')

    analyze_result = analyze_manager.join()
```

Şekil 2.24. Takipçi içerisinde analiz işlemlerinin başlatılması.

Yeni bir dosyanın oluşmasından sonra çağrılan iş Şekil 2.24 ile görülebilir. İlgili fonksiyon çok büyük olduğu için sadece ufak bir kısmı gösterilmektedir. Burada analiz motoru yöneticisinin kullanıldığı görülmektedir. Aynı zamanda e-posta ayrıştırıcısı da e-posta içeriğini ayrıştırmaktadır. Ayrıştırılmış durumdaki veri üzerinde de analiz motorları çalıştırılmaktadır. Örneğin gelen pakedin içerisinde yer alan kaynak IP üzerinde *ip_engine* ismi verilen içerisinde zararlı IP adreslerinin bulunduğu imza veritabanı çalıştırılmaktadır. Ancak *From* başlık bilgisi üzerinde de e-postanın gerçekten ilgili kişi üzerinden gelip gelmediği kontrol edilmektedir. Yani bir kural çalıştırılmaktadır. Bu kısım işine başladığı anda analize başladığını panel uygulama arayüzüne bildirmektedir. Bildirimden sonra analiz işlemi başlar ve sonuçları tekrardan ilgili arayüze bildirir. İlk bildirim işlemi Şekil 2.25 ile görülebilir.

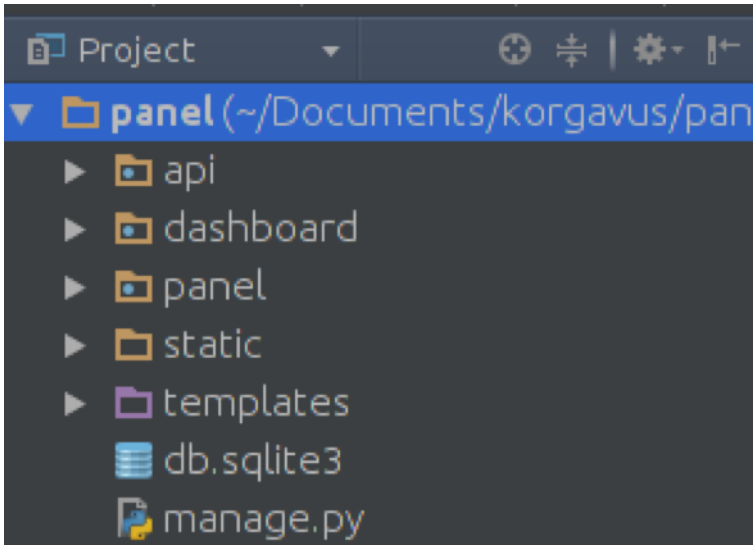
```
try:
    response = requests.post(korgavus_config['panel']['analyze_api'], data=json.dumps(request_data))
    if response.ok:
        result_data = response.json()
        response_status = result_data['status']

        if response_status:
            result_data = result_data['result']
        else:
            move_to_trash(file_name, 'analyze api response status error')
            return
    else:
        move_to_trash(file_name, 'analyze api response error')
        return
except Exception as e:
    move_to_trash(file_name, e)
    return
```

Şekil 2.25. Analiz işleminin başladığının panel'e gönderilmesi.

Analiz işlemi sırasında tüm hata durumları kontrol edilmektedir. Herhangi bir hata ile karşılaşıldığı anda ilgili e-posta içeriği çöp kutusuna gönderilmektedir. Bu sayede hiçbir e-posta kaçırılmamaktadır. Bu e-postaların panel arayüzü içerisinde gösterilmesi için gerekli ek kısımlarda hazırlanmıştır ancak eklenmemiştir.

2.7 Panel

Projenin arayüzünde *Django* iş çatışı kullanılarak bir web uygulaması oluşturulmuştur[30]. Bu kapsamda panel katmanı altında arayüz tasarımı gerçekleştirilmiştir. Şekil 2.26 Panel uygulamasının klasör yapısını göstermektedir. Görüldüğü üzere proje *api*, *dashboard*, *panel*, *static* ve *templates* olmak üzere 5 alt klasöre ayrılmıştır. Bu yapılandırma *Django* projelerinin tipik bir uygulamasıdır.

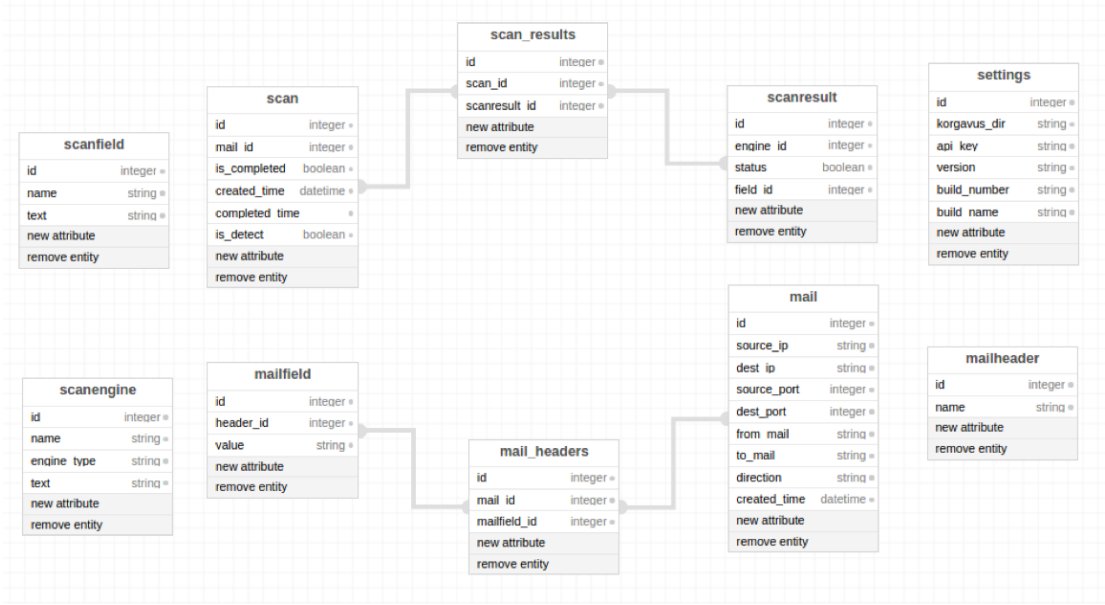


Şekil 2.26. Panel uygulaması klasör yapısı.

Klasörler içerisinden *api*, projenin analiz kısmı ile haberleşip istenilen verileri arayüze sunmak için oluşturulan bir uygulama arayüzüdür. Bu kısım alt seviye kodlar ile haberleşilip verileri arayüze aktaracak şekilde ayarlanmıştır. *Dashboard*, panelin arayüzünde kullanılacak temel yapıları içerir. *model* ve *view* katmanları bu yapı içerisindedir. *panel* klasörü içerisinde projenin çalışması için gerekli ayar dosyaları bulunmaktadır. *templates*, grafiksel arayüzün tasarımlarını içermektedir.

2.7.1 Model Katmanı

Arayüzde kullanılacak verilerin veritabanı ile iletişim halinde olması gerekmektedir. Bu kapsamda arayüzün veri tabanının tasarımı yapılmıştır. Şekil 2.27 bu veritabanının tasarımını göstermektedir. İstenen veritabanı tabloları ve kolonları da *models.py* içerisinde uygulamanın kendi model arayüzleri kullanılarak sınıflar şeklinde tanımlanmıştır. Bu tanımlamalar sayesinde proje veritabanı yönetim sistemi bağımsız hale gelmiştir. Yani projenin veritabanı *PostgreSQL* ise ve *MySQL* haline dönüştürülmek isteniyorsa, yapılması gereken tek şey ayar dosyasından ilgili kısmı değişmektir. Veritabanı tabloları ve kolonları otomatik olarak oluşturulacaktır. Hatta *Django* kütüphanesinin araçları ile veriler de yeni veritabanına eklenebilmektedir.



Şekil 2.27. Panel uygulaması veritabanı tasarımı.

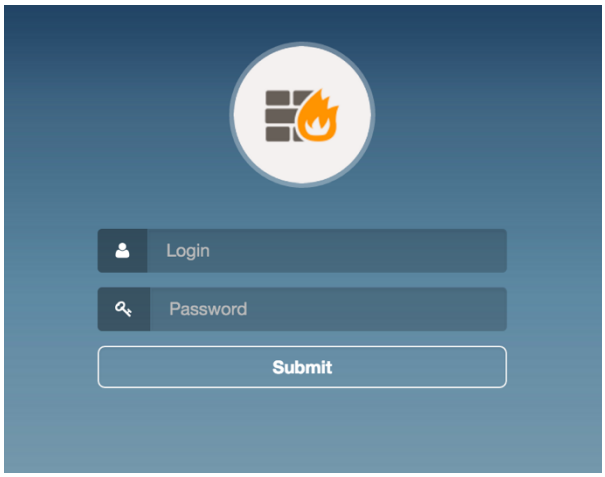
Veritabanı tasarımı şema üzerinde tasarlandıktan sonra ilgili tablolar sınıf yapısı şeklinde oluşturulmuştur. E-posta içeriklerini tutacak tablonun *Django ORM* hali Şekil 2.28 ile görülebilir. Geçit üzerinden geçen tüm e-posta trafiği bu formata dönüştürülüp veritabanına kaydedilmektedir. Daha sonra analiz işlemine tabi tutulan e-postanın sonucu beklenmektedir. Bu sonuç da gene benzer şekilde veritabanına kaydedilmektedir.

```
class Mail(models.Model):
    source_ip = models.GenericIPAddressField()
    dest_ip = models.GenericIPAddressField()
    source_port = models.IntegerField()
    dest_port = models.IntegerField()
    from_mail = models.EmailField()
    to_mail = models.EmailField()
    headers = models.ManyToManyField(MailField)
    direction = models.CharField(choices=DIRECTIONS, max_length=2)
    created_time = models.DateTimeField(auto_now_add=True)

    def __unicode__(self):
        return '{} -> {}, {}'.format(self.source_ip, self.dest_ip, self.direction)
```

Şekil 2.28. E-posta model yapısı.

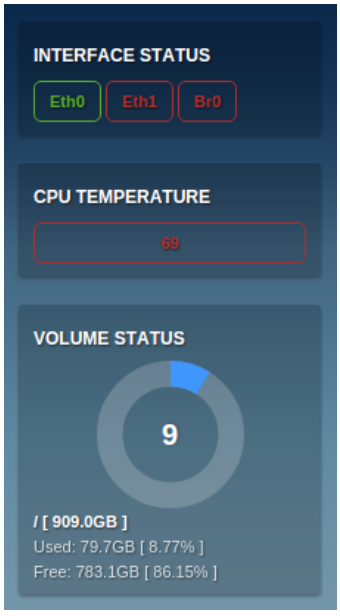
2.7.2 Görünüm

Her ne kadar arka plan süreçlerinde başarılı da olunsa grafiksel arayüz projenin dışardan bakıldığı andaki itibarı için çok önemlidir. Bu yüzden tasarımın modern bir yapıya sahip olması gerekmektedir. Bu yapı *Twitter Bootstrap* kütüphanesi kullanılarak *Jquery* ve kütüphaneleri ile birlikte gerçekleştirilmiştir[31][32]. Panelin arayüzü en başta *html, css* ve *javascript* olarak hazırlanmıştır. Bu hazırlamaların tamamlanmasından sonra *Django* uygulaması içerisine eklenmiştir. Örneğin Şekil 2.29 tasarımı yapılan panelin giriş kısmını göstermektedir.



Şekil 2.29. Panel giriş tasarımı.

Tasarlanan panelin tüm verileri *ajax* istekleri ile arayüz aracılığı ile aktarılması planlanmıştır. Yani sayfaya giren herhangi bir kullanıcı sayfaya herhangi bir yenileme yapmadan tüm değişimleri görmesi planlanmıştır. Bunun için de grafiksel eklentiler tasarlanmıştır. Şekil 2.30 bu tasarımı göstermektedir. Burada ağ adaptörlerin durumundan işlemci sıcaklığına kadar çeşitli sistem verileri anlık olarak değişecek şekilde ayarlanmıştır. Bu iş için sürekli olarak arka tarafa istek atacak *javascript* kodlarının ufak bir kısmı Şekil 2.31 ile görülebilir. Burada kısa süre arayla istek atılmakta ve veriler *jquery* seçicileri aracılığı ile güncellenmektedir.



Şekil 2.30. Eklentilerin tasarımı.

```
(function interface_worker() {
    $.ajax({
        url: '{% url 'status_device' %}',
        success: function (data) {
            $("#eth0").removeClass();
            $("#eth0").addClass("btn btn-sm btn-clean btn-" + data.eth0);
            $("#eth1").removeClass();
            $("#eth1").addClass("btn btn-sm btn-clean btn-" + data.eth1);
            $("#br0").removeClass();
            $("#br0").addClass("btn btn-sm btn-clean btn-" + data.br0);

            var totalS = "/ [ " + data.total + " ]";
            $("#total").text(totalS);

            var usedS = "Used: " + data.used + " [ " + data.used_p + "% ]";
            $("#used").text(usedS);

            var freeS = "Free: " + data.free + " [ " + data.free_p + "% ]";
            $("#free").text(freeS);

            $("#free_p").val(data.used_p).trigger('change');
```

Şekil 2.31. Ajax çağrısı atan kod bloğu örneği.

2.7.3 Kontrol Kısımları

Panel uygulamasının model katmanı ile tasarım bloğunu birbirine bağlamak için kontrol kısmı kullanılmaktadır. Burada uygulamaya yapılan istekler belirli kurallar çerçevesinde işlenerek veritabanı sorguları ve sonuçları oluşturulmaktadır. Daha sonra da daha önceden oluşturulmuş olan arayüz içerisine doldurulmaktadır. Örnek bir kontrol bloğu Şekil 2.32 ile görülebilir. Burada yer alan *dashboard_details* isimli kontrol fonksiyonu gelen istek içerisinden hangi tarama olduğunu bulmaktadır ve geriye bu sonuçları dönmektedir. Burada dikkat edildiği üzere gerçekleştirilen kontrol bloklarının tüm hepsinde *login_required* yardımcısı ile tüm sayfalara giriş yapılmadan girilmesi engellenmiştir.

Tüm verilerin listelendiği gibi sadece spesifik detayların da listelendiği kontrol blokları bulunmaktadır. Bu fonksiyonların hepsi *views.py* dosyası içerisinde tanımlanmaktadır.

```
def dashboard_details(request, pk):
    scan = get_object_or_404(Scan, id=pk)
    headers = scan.mail.headers.all()

    engines = scan.results.filter(engine__engine_type='E')
    policies = scan.results.filter(engine__engine_type='P')

    return render(request, 'dashboard/details.html', locals())


@login_required(login_url='dashboard_threats')
def dashboard_threats(request):
    threats_mails = Scan.objects.filter(is_detect=True).filter(is_completed=True)

    return render(request, 'dashboard/threats.html', locals())
```

Şekil 2.32. Örnek kontrol bloğu.

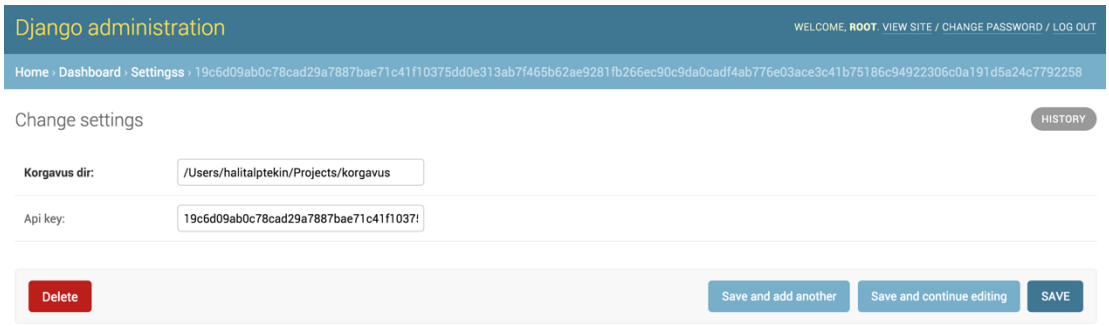
2.7.4 Uygulama Arayüzü

Analizi yapılan e-postanın sonuçlarının panel'e bildirilebilmesi için bir arayüze ihtiyaç duyulmuştur. Bu arayüz *json* istek alıp *json* istek dönecektir. Bu kısımda iki adet arayüz gerçekleştirilmiştir. Bunlardan ilki yakalanan e-postanın bildirilmesi ile ilgilenirken diğeri analiz işleminin sonucunu göndermektedir. Şekil 2.33'de yer alan kısım analiz sonucunu kabul eden fonksiyona aittir. Burada farkedildiği üzere gelen istekler filtrelenmektedir. Sadece istenen içeriğin olması sağlandıktan sonra isteği gönderen kişinin yetkili olup olmadığının kontrolü yapılmaktadır. Bu kontrol için *api_key* değeri kullanılmaktadır.

```
@csrf_exempt
def put_result(request):
    if request.method == 'POST':
        request_body = request.body
        try:
            request_json = json.loads(request_body)
            filter_result = [parameter for parameter in RESULT_PARAMETERS if parameter not in request_json.keys()]

            if not filter_result:
                scan = Scan.objects.filter(id=request_json['scan_id'])
                if scan:
                    scan = scan[0]
                    is_valid = True
                    is_detect = False
                    scan_results = []
                    for result in request_json['scan_results']:
                        engine = ScanEngine.objects.filter(name=result['name'])
                        field = ScanField.objects.filter(name=result['field'])
                        if engine and field:
                            engine = engine[0]
                            field = field[0]
                            scan_results.append({'engine': engine, 'status': result['status'], 'field': field})
                            if result['status']:
                                is_detect = True
                        else:
                            is_valid = False
```

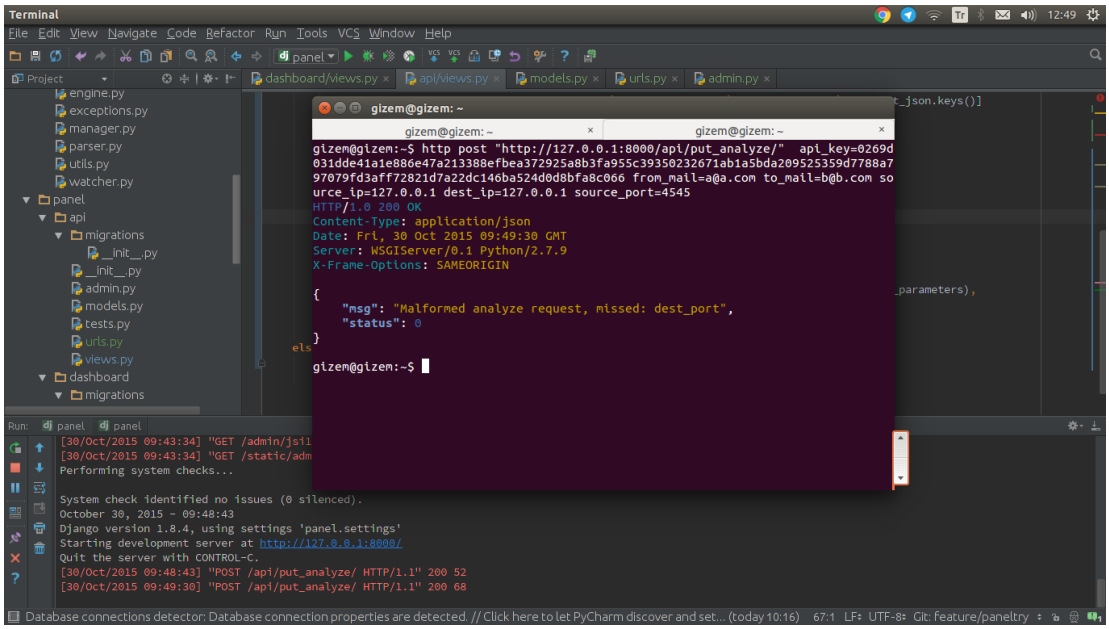
Şekil 2.33. Analiz sonucunu kabul eden uygulama arayüzü.

Panel uygulamasının kendi yönetim paneli bulunmaktadır. Normalde kullanıcının bu kısma girmesine gerek yoktur ancak çeşitli düzenlemeler için buraya da girebilmektedir. Bu panel içerisinde *api key* değerleri *settings* kısmı altında tutulmaktadır. Kullanıcı herhangi bir değer girmese dahi otomatik olarak oluşturulmaktadır. Bu yüzden kullanıcı etkileşimi beklememektedir. Şekil 2.34 yönetici paneli içerisinde bu kısmı göstermektedir.



Şekil 2.34. Api anahtarlarının yönetici paneli içerisindeki görünümü.

Uygulama arayüzünün tasarımı gereği tüm önlemler alınmaya çalışmıştır. Örneğin gönderilen istek içerisinde eksik bir değer varsa bu geriye bildirilmektedir. Benzer şekilde *status* değeri ile de uygulamalara gerçekleştirilen isteklerin sonucu belirtilmektedir. Örnek bir istek Şekil 2.35 ile görülebilir. Burada *put_analyze* linkine geçerli bir api anahtarı ile birlikte istek atılmıştır.



Şekil 2.35. Uygulama arayüzü istek denemesi.

2.8 Siber Tehdit Kaynak Arayüzü

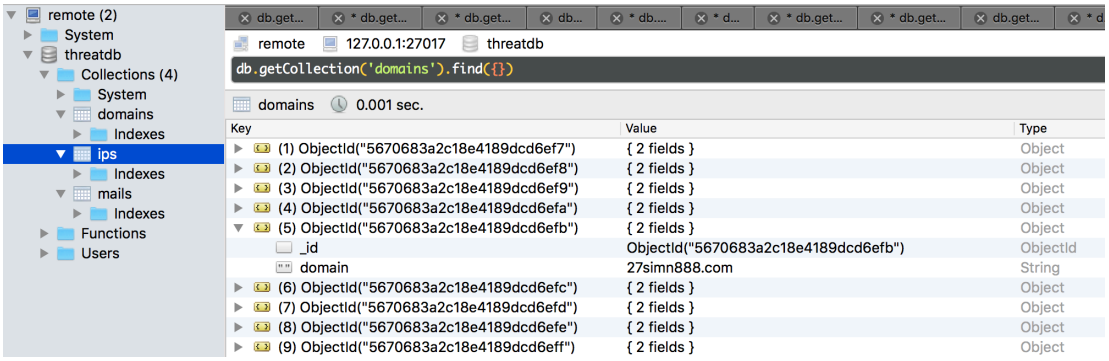
Üzerinden geçen trafiğin analiz edilmesi sırasında kullanılan kural ve imza veritabanlarının güncel tutulması gerekmektedir. Bu amaçla siber tehdit kaynak arayüzü gerçekleştirilmiştir. Burada sürekli olarak siber tehditler çeşitli kaynaklardan toplanarak geçerli kurallar topluluğuna dönüştürülmektedir.

2.8.1 Sunucu Kurulumu

Siber tehditlerin toplanması ve tutulması için uzak bir sunucu ihtiyacı bulunmaktadır. Bunun için EK 2 içerisinde anlatıldığı üzere *Ubuntu Server* kurulumu yapılmıştır. Host adı olarak da *remote* seçilmiştir.

2.8.2 Veritabanı

Siber tehditlerin tutulması için veritabanı olarak *mongodb* seçilmiştir[33]. Bu seçimin yapılmasında verilerin yüksek miktarlarda olması ve birbirleri arasında ilişki olmaması verilebilir. Bu yüzden veritabanı şemasız bir şekilde tasarlanmıştır. Bu tasarım içerisinde siber tehditler üzerinde çeşitli gruplamalar yapılmıştır. Örneğin hacker IP adresleri ile zararlı yazılım dağıtan IP adresleri temelde aynı *collection* içerisinde tutulmaktadır. Aynı zamanda bu veriler arasında herhangi bir ilişki bulunmamaktadır. Haliyle *mongodb* tasarlanan sistem için en uygun yapıya sahiptir.



Şekil 2.36. Siber tehdit veritabanı içeriği.

Şekil 2.36 siber tehdit veritabanına aittir. Burada veritabanı isminin *threatdb* olduğu ve *domains, ips* ve *mails* isminde *collection* yer aldığı görülebilir.

2.8.3 Tehdit Toplayıcıları

Tehdit toplayıcıları tehdit kaynakları içerisinden verileri alarak veritabanına ekleyen kısımdır. Bu kısım için gerçekleştirilen taban sınıf Şekil 2.37 ile görülebilir.

```
class ThreatCrawler(object):
    NAME = ''
    COLLECTION = ''
    FIELD = ''
    VERSION = ''
    URL = ''
    SOURCE_REGEX = ''
    BAD_CHARS = ['']
    BAD_CHARS_F = False

    def __init__(self):
        self.content = None
        self.result = None
        self.re_compiled = re.compile(self.SOURCE_REGEX)

    def run(self):
        try:
            response = requests.get(self.URL)
            if response.ok:
                self.content = response.content
                self.before_parse()
                self.parse()
                if self.BAD_CHARS_F:
                    for item in self.result:
                        for bad_char in self.BAD_CHARS:
                            item.replace(bad_char, '')
                self.after_parse()
                return self.result
            else:
                pass
        except requests.ConnectionError:
            pass

    def before_parse(self):
        self.result = self.result

    def parse(self):
        self.result = self.re_compiled.findall(self.content)

    def after_parse(self):
        self.result = self.result
```

Şekil 2.37. Tehdit toplayıcı taban sınıfı.

Tehdit toplayıcıların rahatlıkla gelişime açık olması için taban sınıf yapısı oluşturulmuştur. Aynı analiz motorlarında yapılan geliştirme benzeri bir yapı gerçekleştirilmiştir. Bu sayede yeni bir tehdit kaynağı eklemek isteyen geliştiriciler rahatlıkla ekleme yapabilmektedir. Yapılması gereken sadece taban sınıftan yeni bir sınıf türetmek ve kaynağa özgü değişiklikler yaparak tamamlamaktır.

```
class MalwareDomainsCrawler(ThreatCrawler):
    NAME = 'malware_domains'
    COLLECTION = 'domains'
    FIELD = 'domain'
    VERSION = '0.1.0'
    URL = 'http://mirror1.malwaredomains.com/files/justdomains'
    SOURCE_REGEX = '(.*)\n'

    def after_parse(self):
        self.result = self.result[1:]


class ZeusDomainsCrawler(ThreatCrawler):
    NAME = 'zeus_domains'
    COLLECTION = 'domains'
    FIELD = 'domain'
    VERSION = '0.1.0'
    URL = 'https://zeustracker.abuse.ch/blocklist.php?download=baddomains'
    SOURCE_REGEX = '(.*)\n'

    def after_parse(self):
        self.result = self.result[6:]


class ZeusIPCrawler(ThreatCrawler):
    NAME = 'zeus_ips'
    COLLECTION = 'ips'
    FIELD = 'ip'
    VERSION = '0.1.0'
    URL = 'https://zeustracker.abuse.ch/blocklist.php?download=badips'
    SOURCE_REGEX = '(.*)\n'

    def after_parse(self):
        self.result = self.result[6:]
```

Şekil 2.38. Örnek tehdit toplayıcılar.

Geliştirilen yapıya uygun olarak örneğin *Zeus* zararlısının paylaşıldığı site adresleri toplanmak istenmektedir. Bunun için yapılması gereken *ThreatCrawler* isimli taban sınıftan yeni bir sınıf türetmektir. Türetilen *ZeusDomainCrawler* sınıfı Şekil 2.38 ile görülebilir. Daha sonra *URL* değeri ile hangi kaynak adres üzerinden verinin alacağı tanımlanmıştır. İstek gerçekleştirildikten sonra dönen sonuç üzerinde düzenli ifade çalıştırılmaktadır. Bu da *SOURCE_REGEX* isimli değişken aracılığı ile yapılmaktadır. Son olarak da eğer son halin üzerinde bir değişiklik yapılacaksa *after_parse* fonksiyonu yeniden yazılmaktadır. Yeni bir tehdit toplayıcı eklemek bu kadar ile sonlanmaktadır. Sistem bir sonraki tarama işlemi sırasında bu toplayıcı da çalıştıracaktır.

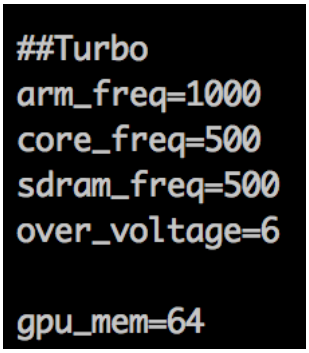
2.9 Yapılandırmalar

Proje içerisindeki yazılımların kodlanmasının tamamlanması projenin bittiği anlamına gelmemektedir. Bu yüzden hem yazılımların hem de işletim sistemlerinin yapılandırılması gerekmektedir. Yapılandırmalar bazen klasör yapısı olabilirken bazen de işletim sisteminin daha performanslı çalışması için yapılmaktadır.

2.9.1 Geçit Yapılandırmaları

Geçit donanımı projenin en kritik kısmını oluşturmaktadır. Bu yüzden geçit üzerindeki donanımın performanslı çalışması gerekmektedir.

Kullanılan *Raspberry PI* cihazı ilk alındığında *800 MHZ* ile çalışmaktadır[34]. Proje kapsamında bu saat frekansı *1000 MHZ'*ye çıkarılmıştır. Bunun için */boot/config.txt* dosyası Şekil 2.39 gibi düzenlenmiştir. Burada gözüktüğü gibi belleğin performansında da artış yapılmıştır. Tüm bu düzenlemelerden sonra performans arttırımı gözükse de sıcaklık değerinde artış yaşanmıştır. Bunun çözümü için işlemci üzerine soğutucu ile birlikte fan takılmıştır. Bu artış yaşanmadan önce 30-35 derece civarında olan işlemci sıcaklığı artıştan sonra 60-65 derece civarına çıkmıştır. Gerekli montajın yapılmasından sonra bu değerler 30-35 derece civarına tekrardan geri dönmüştür.



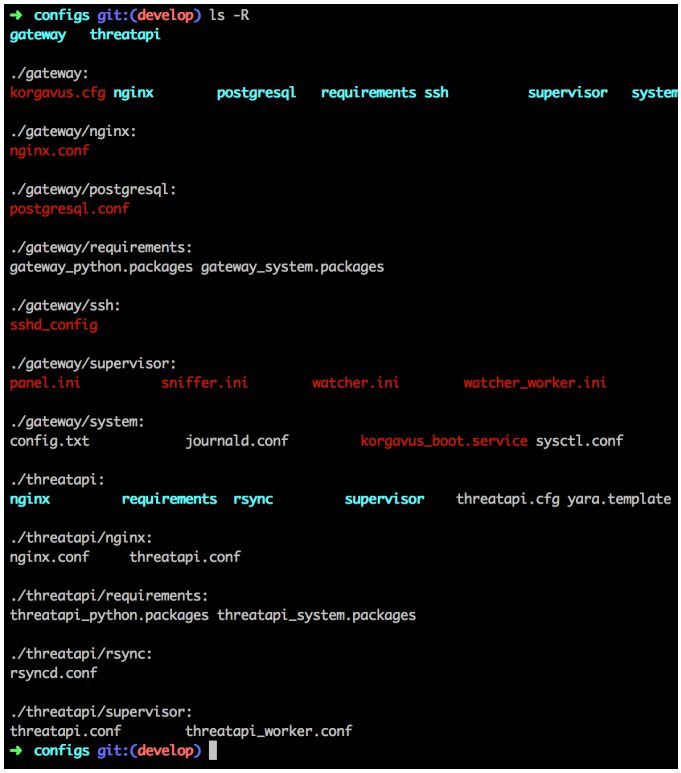
Şekil 2.39. Cihaz performans arttırımı.

Proje açıldığı zaman *PostgreSQL, sniffer, rabbitmq, celery, django, gunicorn, nginx* gibi bir çok uygulama da çalışır hale gelmektedir[35][36][37][38]. Normalde ciddi derecede bellek kullanımı beklenirken *Arch Linux ARM* başarılı bir bellek yönetimi yaparak bunu minimuma indirmektedir. Hiçbir uygulama çalışmıyorken *15 MB* bellek kullanımı yapan işletim sistemi, tüm proje tam performans çalışırken *121 MB* bellek kullanımı yapmaktadır. Bu durum Şekil 2.40 ile görülebilir.

```
[root@gateway korgavus]# free -m
              total        used        free      shared  buff/cache   available
Mem:            923         121         490          23         310         742
Swap:             0           0           0
[root@gateway korgavus]#
```

Şekil 2.40. Proje bellek kullanımları.

Proje yapılandırmaları çerçevesinde kullanılan ayar dosyalarının tümü *configs* klasörü altında gruplandırılmıştır. Bu gruplandırma işlemi sonucu oluşan son yapı Şekil 2.41 ile görülebilir. Örneğin *Django* uygulamasını çalıştıran ve sürekli açık tutan *supervisor* uygulamasının ayar dosyası *configs/gateway/supervisor/panel.ini* dizini altındadır[39]. Proje *deploy* edilirken bu ayar dosyası da olması gereken yere kopyalanmaktadır ve yeniden çalıştırılmaktadır. Burada kullanılan *supervisor* uygulaması proje içerisinde çalışan tüm servisleri ayakta tutma konusunda garanti vermektedir. Bu sayede herhangi bir durumda eğer servis kapanırsa bu uygulama tekrardan başlatmaktadır ve bunu kayıt altına almaktadır. Aynı zamanda servisin ekrana bastığı tüm içeriği de kayıt altına almaktadır.



Şekil 2.41. Ayar dosyaları dizini.

Proje geliştirilme sırasında sürekli olarak yeni özellik eklemesi yapılmıştır. Haliyle güncel halin gerçek ortamda da görülebilmesi için harici betiklerden faydalanılmıştır. Bu betiklerin yazımı için *Python* programlama diline ait *fabric* kütüphanesi kullanılmıştır. Şekil 2.42 bu amaç için yazılmış aracın ufak bir kısmını göstermektedir[40].

```
run('tar czvf /tmp/{}.tar.gz /korgavus/logs /var/log'.format(timestamp))
run('rm -rf /korfavus/logs')
run('mkdir -p /korgavus/logs/postgresql')
get('/tmp/{}.tar.gz'.format(timestamp), LOG_BACKUP_DIR)
run('rm -rf /tmp/{}.tar.gz'.format(timestamp))
time.sleep(2)

run('tar czvf /tmp/{}.tar.gz /korgavus/mails'.format(timestamp))
run('rm -rf /korgavus/mails')
run('mkdir -p /korgavus/mails/{archive,queue}')
get('/tmp/{}.tar.gz'.format(timestamp), MAIL_BACKUP_DIR)
run('rm -rf /tmp/{}.tar.gz'.format(timestamp))
time.sleep(2)

run('rm -rf /etc/systemd/journald.conf')
run('ln -s /korgavus/configs/gateway/system/journald.conf /etc/systemd/journald.conf')
run('systemctl restart systemd-journald')
time.sleep(2)

run('rm -rf /etc/sysctl.conf')
run('ln -s /korgavus/configs/gateway/system/sysctl.conf /etc/sysctl.conf')
run('sysctl -p /etc/sysctl.conf')
time.sleep(2)

run('systemctl restart systemd-timesyncd')

run('rm -rf /etc/systemd/system/korgavus_boot.service')
run('ln -s /korgavus/configs/gateway/system/korgavus_boot.service /etc/systemd/system/korgavus_boot.service')
```

Şekil 2.42. Yardımcı araç yazımı.

Geçit en başta açılırken gerekli bazı işlemler vardır. Bunlardan bazıları Ethernet sürücüsünün yüklenmesi ve arayüzlerin köprülenmesidir. Bu amaçla bir adet *bash* betiği oluşturulmuştur. İçeriği büyük olmasına rağmen ufak bir kısmı Şekil 2.43 ile görülebilir.

```
→ scripts git:(develop) cat boot.sh
#!/bin/bash

# Ethernet Driver
cd /korgavus/drivers/ethernet && make load
echo "Ethernet driver loaded"

# Bridge
brctl addbr bridge0
brctl addif bridge0 eth0
brctl addif bridge0 eth1
ifconfig eth0 0.0.0.0
ifconfig eth1 0.0.0.0
ifconfig bridge0 up
dhcpcd bridge0
echo "eth0 and eth1 bridged"
```

Şekil 2.43. Başlangıç betiği.

SSH bağlantısı gerçekleştirilirken kolaylık sağlaması ve güvenlik önlemi almak için özel anahtar ikilileri kullanılmıştır. Bu sayede özel anahtarı olmayan kimse sunucuya giriş yapamamaktadır. Ayrıca SSH servisine karşı yapılabilecek şifre kırma saldırılarının da önüne geçilmiştir. Anahtar ile birlikte bağlantı kurulması Şekil 2.44 ile görülebilir.

```
gizem@gizem:~/Documents/tasarim/keys$ ls -la
total 16
drwxrwxr-x 2 gizem gizem 4096 Ara 12 15:41 .
drwxrwxr-x 3 gizem gizem 4096 Ara 12 15:40 ..
-rw------- 1 gizem gizem 1766 Ara 12 15:41 korgavus_gizem
-rw-r--r-- 1 gizem gizem  393 Ara 12 15:41 korgavus_gizem.pub
gizem@gizem:~/Documents/tasarim/keys$ chmod 700 korgavus_gizem
gizem@gizem:~/Documents/tasarim/keys$ ssh root@gateway -i korgavus_gizem
Enter passphrase for key 'korgavus_gizem':
   __
  / /_____  _______ ____ __  ____ _____
 /  '_/ _ \/ __/ _ `/ _ `| |/ / // (_-<
/_/\_\\___/_/  \_, /\_,_/|___/\_,_/___/
              /___/

Last login: Sat Dec 12 13:43:59 2015 from 192.168.1.11
[root@gateway ~]#
```

Şekil 2.44. SSH bağlantısının özel anahtar ile kurulması.

2.9.2 Yedekleme ve Kayıt Tutma Yapılandırmaları

Proje içerisinde çalışan tüm servislerin kayıtları tek bir dizin altında toplanmaktadır. Bu şekilde hem hata tespiti kolay olmakta hem de yedek alınması kolaylaşmaktadır. Cihaz içerisinde */korgavus/logs* dizini bu amaç için ayrılmıştır. Şekil 2.45'de örnek bir hata kaydı ve hata kayıt dizini içeriği gözükmektedir.

```
➜  logs git:(develop) ls -la
total 2104
drwxr-xr-x  9 halitalptekin  staff     306 Dec 13 13:28 .
drwxr-xr-x  3 halitalptekin  staff     102 Dec 22 21:01 ..
-rw-r--r--  1 halitalptekin  staff  175544 Dec 16 16:50 panel_gunicorn.log
-rw-r--r--  1 halitalptekin  staff  781605 Dec 15 13:13 panel_nginx_access.log
-rw-r--r--  1 halitalptekin  staff   60998 Dec 15 13:13 panel_nginx_error.log
drwxr-xr-x  2 halitalptekin  staff      68 Dec 12 16:51 postgresql
-rw-r--r--  1 halitalptekin  staff       0 Dec 13 10:53 sniffer.log
-rw-r--r--  1 halitalptekin  staff       0 Dec 13 13:19 watcher.log
-rw-r--r--  1 halitalptekin  staff   56606 Dec 16 16:57 watcher_worker.log
➜  logs git:(develop) tail -f panel_nginx_error.log
2015/12/15 11:13:01 [error] 262#0: *903 open() "/korgavus/panel/static/admin/css/base.css
" failed (2: No such file or directory), client: 192.168.1.104, server: gateway, request:
 "GET /static/admin/css/base.css HTTP/1.1", host: "gateway.korgavus.net", referrer: "http
://gateway.korgavus.net/admin/dashboard/scanfield/"
2015/12/15 11:13:01 [error] 262#0: *892 open() "/korgavus/panel/static/admin/css/changeli
sts.css" failed (2: No such file or directory), client: 192.168.1.104, server: gateway, r
equest: "GET /static/admin/css/changelists.css HTTP/1.1", host: "gateway.korgavus.net", r
eferrer: "http://gateway.korgavus.net/admin/dashboard/scanfield/"
2015/12/15 11:13:01 [error] 262#0: *899 open() "/korgavus/panel/static/admin/js/core.js"
failed (2: No such file or directory), client: 192.168.1.104, server: gateway, request: "
GET /static/admin/js/core.js HTTP/1.1", host: "gateway.korgavus.net", referrer: "http://g
ateway.korgavus.net/admin/dashboard/scanfield/"
```

Şekil 2.45. Örnek hata kaydı ve hata kayıt dizini.

Çalışan uygulamaların yanında *supervisor* ile birlikte çalıştırılan servislerin hata kayıtları da aynı dizin altında toplanmaktadır. Örneğin Şekil 2.46 analiz işleminin yapıldığı iş kuyruğunun çalıştırılması için tanımlanan *supervisor* ayar dosyasını göstermektedir.

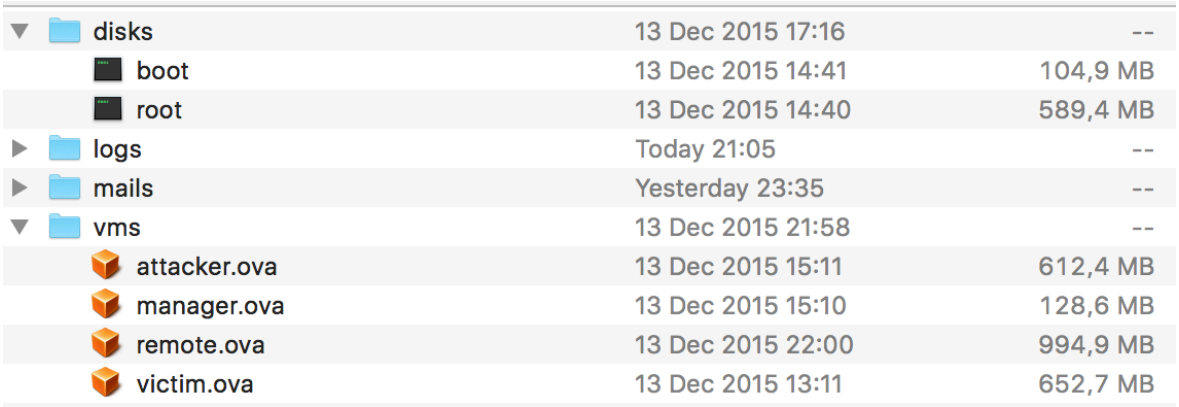
```
[program:watcher_worker]
command = celery -A watcher.app worker -Q watcher -l info
directory = /korgavus/analyzer
redirect_stderr = True
stdout_logfile = /korgavus/logs/watcher_worker.log
environment=C_FORCE_ROOT="yes"
```

Şekil 2.46. Hata kaydı için supervisor ayar dosyası içeriği.

Geçidin diskinin yedeklenmesi için *bootstrap* aracı içerisinde tanımlamalar yapılmıştır. Bu çerçevede *fsarchiver* aracı kullanılarak diskin tüm imajı alınmaktadır ve tekrardan oluşturulabilmektedir[41]. Bu aşamada kullanılan komutlar Tablo 2.3 ile görülebilir. Kullanılan araç gömülü cihazın yedeğini başarıyla aldıktan sonra bir dosyaya kaydetmektedir. Daha sonra herhangi bir şekilde diskte hasar oluşursa ilgili yedekten geriye dönebilmektedir.

```
fsarchiver savefs backup /dev/sdb2
fsarchiver restfs backup id=0,dest=/dev/sdb2
```

Tablo 2.3. Disk yedeği alınması ve tekrardan oluşturulması.



Şekil 2.47. Yedek klasörü içeriği.

Proje geliştirilirken herhangi bir aksi durumla karşılaşılmamak için düzenli olarak yedek alınmıştır. Bu durum sonunda oluşan dizin yapısı Şekil 2.47'den görülebilir. Bu derece düzenli yedek alınmasının faydası karşılaşılan bir durum ile görülmüştür. Cihaz içerisinde takılı olan harici disk aniden çalışamaz duruma gelmiştir. Ancak yedeği bulunduğundan birebir aynısı olacak şekilde geriye döndürülmüştür.

Siber tehdit arayüzü içerisinde yer alan güncel tehdit veritabanlarının geçit içerisine alınması *rsync* aracı ile gerçekleştirilmektedir[42]. Bu aracın getirdiği faydaların başında gereksiz indirme işlemi yapmaması gelmektedir. Örneğin daha önceden sunucudan çekilmiş olan 100MB'lik dosya bulunsun. Bir gün sonra güncelleme gelmektedir ve bu dosyanın boyutu 102MB olmaktadır. Eğer sistem normal bir şekilde tasarlanmış olsaydı ilgili dosya tekrardan indirilecekti. Yani iki gün içerisinde harcanan toplam band genişliği 202MB olacaktır. Ancak *rsync* aracı kullanıldığı için ikinci gün sadece 2MB indirme yapılacaktır. Bu da iki günün toplamını 102MB gibi bir rakama getirecektir. Bu denli büyük bir başarı elde edebilmek için *remote.korgavus.net* sunucusuna *rsync* sunucusu kurulmuştur. Bahsedilen rakamlar aylık veya yıllık düzeyde düşünüldüğünde devasa boyutlara ulaşmaktadır. *rsync* sunucusunun başarıyla kurulduğu Şekil 2.48 ile görülebilir.

```
(bootstrap)➜ bootstrap git:(develop) ✗ rsync 192.168.56.103::
threats         KORGAVUS - Last Threats
(bootstrap)➜ bootstrap git:(develop) ✗ rsync 192.168.56.103::threats
drwxr-xr-x        4096 2015/12/15 23:06:35 .
lrwxrwxrwx          26 2015/12/15 23:05:21 .threatapi_db
-rw-r--r--      240087 2015/12/15 23:06:29 korgavus_domains.rules.gz
-rw-r--r--        1820 2015/12/15 23:06:29 korgavus_ips.rules.gz
-rw-r--r--      135176 2015/12/15 23:06:35 korgavus_mails.rules.gz
-rw-rw-r--     8864579 2015/12/15 23:05:39 malware_engine.rules.gz
lrwxrwxrwx          44 2015/12/15 23:05:21 yara.template
(bootstrap)➜ bootstrap git:(develop) ✗
```

Şekil 2.48. Rsync kurulumunun tamamlanması ve test edilmesi.

Siber tehditlerin güncel olarak toplanması amacıyla periyodik görevler oluşturulmuştur. Bunlardan ilki internet üzerinden tüm tehditlerin toplanmasıdır. Bu işlem 3 saatte bir gerçekleşmektedir. Toplanan tehditleri kullanarak yeni kural oluşturulması gerekmektedir. Bu iş de 6 saat'de bir gerçekleşmektedir. Haliyle gün içerisinde 4 adet yeni güncelleme yapılmaktadır. Bu 4 güncelleme için 8 defa da siber tehdit toplanması gerçekleştirilmektedir. Kuralların derlenmesi işlemi yara imzalarının oluşturulmasıdır. Oluşturulan imzalar *rsync* ile uyumlu olacak şekilde sıkıştırılmaktadır. Bu işler Şekil 2.49 ile görülebilir. Periyodik imza oluşturmanın yanı sıra *clamav* antivirüs yazılımının imza veritabanı da belirli aralıkla *yara* imzalarına çevrilmektedir[43].

```
@periodic_task(run_every=crontab(minute=0, hour='*/3'))
def fetch_threats():
    for crawler in ACTIVE_CRAWLERS:
        create_threat.delay(crawler)


@periodic_task(run_every=crontab(minute=0, hour='*/6'))
def create_rules():
    client = MongoClient(config_parser.get('threatapi', 'db_host'),
                         int(config_parser.get('threatapi', 'db_port')))

    template = open(os.path.join(config_parser.get('threatapi', 'rule_path'), 'yara.template')).read()
    db_number = open(os.path.join(config_parser.get('threatapi', 'rule_path'), '.threatapi_db')).read()

    try:
        db_number = int(db_number)
    except ValueError:
        db_number = 0

    for crawler in THREATS:
        threat_rules = ''
        data = client[config_parser.get('threatapi', 'db_name')][crawler[0]].find()
        for i, item in enumerate(data):
            clean_data = item[crawler[1]]

            threat_rule = template.replace('{{name}}', crawler[1])
            threat_rule = threat_rule.replace('{{id}}', str(i))
            threat_rule = threat_rule.replace('{{description}}', 'malicious')
            threat_rule = threat_rule.replace('{{value}}', clean_data)
            threat_rule += '\n'

            threat_rules += threat_rule
        file_name = '/tmp/{}_engine.rules'.format(crawler[1])
        open(file_name, 'w').write(threat_rules)

        commands.getoutput('gzip --rsyncable {}'.format(file_name))
        commands.getoutput('mv {}.gz {}'.format(file_name, config_parser.get('threatapi', 'rule_path')))
    open(os.path.join(config_parser.get('threatapi', 'rule_path'), '.threatapi_db'), 'w').write(str(db_number + 1))
```

Şekil 2.49. Periyodik olarak kural oluşturulması ve siber tehdit araştırması yapılması.

2.9.3 Saldırı Engelleme Yapılandırmaları

Proje içerisinde *DoS* saldırılarının engellenmesi için herhangi bir üst seviye işlem yapılmamıştır[44]. Aynı zamanda *spam* e-posta atma hızı konusunda da üst seviye bir değişiklik yapılmamıştır. Tüm bu işlemler *iptables* aracı ile gerçekleştirilmiştir[45]. Cihazın açılışı sırasında bu kurallar sisteme dahil olmaktadır ve bilinen çoğu saldırı türüne karşı önlem almaktadır.

Örneğin geçide yönelik herhangi bir port taraması gerçekleştirme işlemi doğrudan engellenmektedir. Bu engel işlemi aynı zamanda kayıtlara da eklenmektedir. Belirli bir zaman sonra bu engel otomatik olarak ortadan kalkmaktadır. Benzer şekilde aynı anda açılabilecek bağlantı sayısı da kısıtlanmıştır. Bu sınır *ssh* ve *http* için 3'tür. Sisteme eklenen tüm kurallar Şekil 2.50 ile görülebilir.

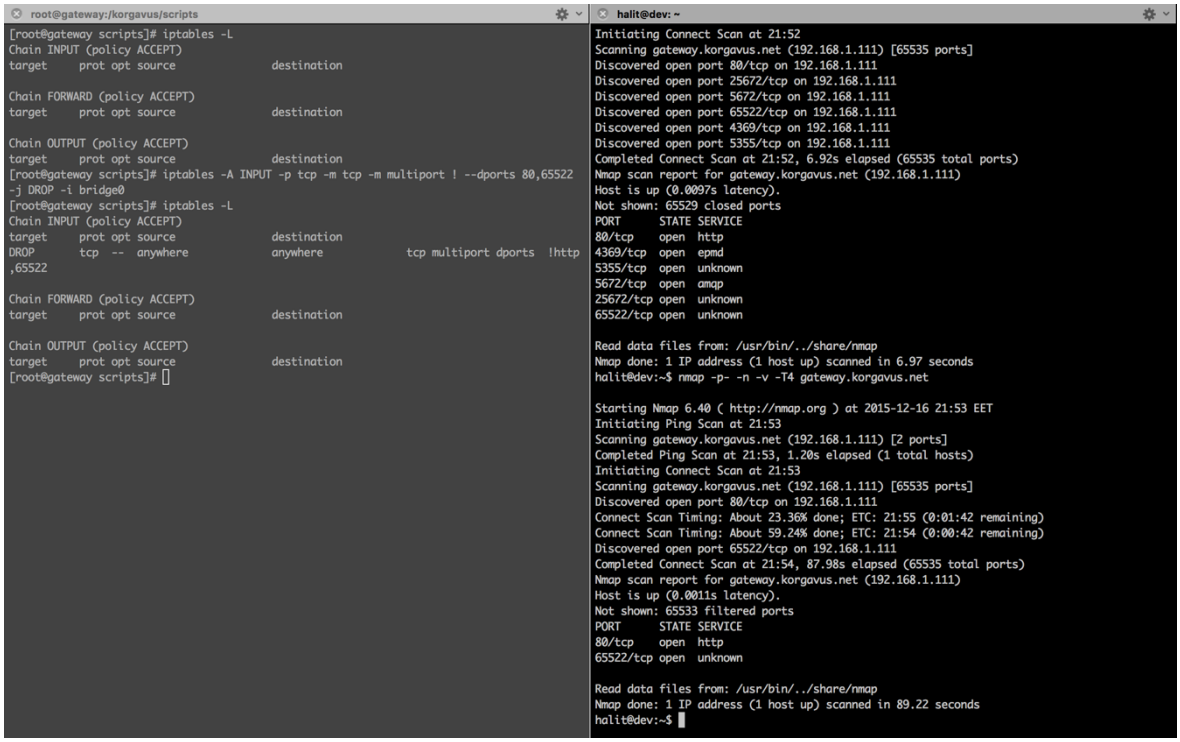
```
# IPTables rules
iptables -F
iptables -X
iptables -A INPUT -i lo -p all -j ACCEPT
iptables -A INPUT -p tcp -m tcp -m multiport ! --dports 80,65522 -j DROP
iptables -A INPUT -p tcp --syn --dport 65522 -m connlimit --connlimit-above 3 -j REJECT
iptables -A INPUT -p tcp --syn --dport 80 -m connlimit --connlimit-above 3 -j REJECT
iptables -A INPUT -p tcp -m state --state NEW -m limit --limit 5/second --limit-burst 5 -j ACCEPT
iptables -A INPUT -p tcp -m state --state NEW -j DROP
iptables -A INPUT -m state --state ESTABLISHED,RELATED -j ACCEPT
iptables -A INPUT -p tcp ! --syn -m state --state NEW -j DROP
iptables -A INPUT -f -j DROP
iptables -A INPUT -p tcp --tcp-flags ALL ALL -j DROP
iptables -A INPUT -p tcp --tcp-flags ALL NONE -j DROP
iptables -A INPUT -s 10.0.0.0/8 -j DROP
iptables -A INPUT -s 169.254.0.0/16 -j DROP
iptables -A INPUT -s 172.16.0.0/12 -j DROP
iptables -A INPUT -s 127.0.0.0/8 -j DROP
iptables -A INPUT -s 224.0.0.0/4 -j DROP
iptables -A INPUT -d 224.0.0.0/4 -j DROP
iptables -A INPUT -s 240.0.0.0/5 -j DROP
iptables -A INPUT -d 240.0.0.0/5 -j DROP
iptables -A INPUT -s 0.0.0.0/8 -j DROP
iptables -A INPUT -d 0.0.0.0/8 -j DROP
iptables -A INPUT -d 239.255.255.0/24 -j DROP
iptables -A INPUT -d 255.255.255.255 -j DROP
iptables -A INPUT -p icmp -m icmp --icmp-type address-mask-request -j DROP
iptables -A INPUT -p icmp -m icmp --icmp-type timestamp-request -j DROP
iptables -A INPUT -p tcp -m tcp --tcp-flags RST RST -m limit --limit 2/second --limit-burst 2 -j ACCEPT
iptables -A INPUT -m state --state INVALID -j DROP
iptables -A FORWARD -m state --state INVALID -j DROP
iptables -A OUTPUT -m state --state INVALID -j DROP
iptables -A INPUT -p tcp -m tcp --tcp-flags RST RST -m limit --limit 2/second --limit-burst 2 -j ACCEPT
iptables -A INPUT -m recent --name portscan --rcheck --seconds 86400 -j DROP
iptables -A FORWARD -m recent --name portscan --rcheck --seconds 86400 -j DROP
iptables -A INPUT -m recent --name portscan --remove
iptables -A FORWARD -m recent --name portscan --remove
iptables -A INPUT -p tcp -m tcp --dport 139 -m recent --name portscan --set -j LOG --log-prefix "portscan:"
iptables -A INPUT -p tcp -m tcp --dport 139 -m recent --name portscan --set -j DROP
iptables -A FORWARD -p tcp -m tcp --dport 139 -m recent --name portscan --set -j LOG --log-prefix "portscan:"
iptables -A FORWARD -p tcp -m tcp --dport 139 -m recent --name portscan --set -j DROP
iptables -A INPUT -p icmp -m icmp --icmp-type 8 -j DROP
echo "Iptables configured"
```

Şekil 2.50. Iptables kuralları.

Sunucunun kendisini korumak için alınan önlemlerden bir diğeri de portları dışarıya kapatmaktır. İçeriden dışarıya yönelik tüm trafiğe izin verilirken dışarıdan içeriye doğru sadece 80 ve 65522 portları açık durumdadır. Normalde açık olmasına rağmen port tarama sonucunda çıkmaması sağlanmıştır. Bu durumların öncesi ve sonrası Şekil 2.51 ile görülebilir.

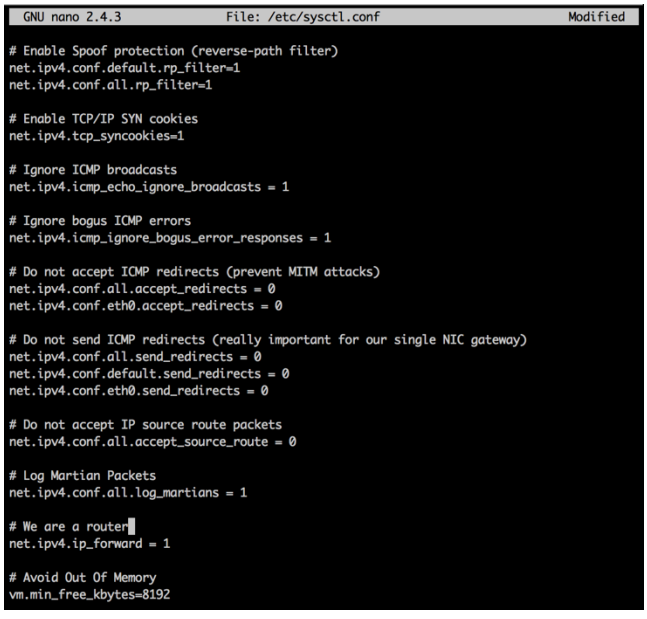
Geçit üzerinden geçen e-posta trafiği ile ilgilenmesine rağmen bu trafik ile ilgili de çeşitli kurallar içermektedir. Örneğin belirli bir süre içerisinde atılabilecek e-posta sayısı bağlantı sayısı ile sınırlandırılabilmektedir. Bu işlem üst kısımda yapılmak yerine alt seviye araç olan *iptables* ile gerçekleştirilmiştir.

Gerçekleştirilen projenin bir diğer önemli amacı da üzerinden geçen gereksiz trafiği ayıklamaktır. Bu yüzden arka tarafta korunan ağ içerisine gelen gereksiz ve bozuk türdeki paketler daha geçit aşamasında elenmektedir. Haliyle korunan ağ'a doğru temiz bir trafik akmaktadır. Bu şekilde olması hem güvenlik açısından hem de stabil durum açısından önemlidir.



Şekil 2.51. Port kapama işlemlerinin sonuçları.

Geçit üzerinde gerçekleştirilen güvenlik çalışmaları sadece paket tabanlı değildir. Ek olarak işletim sistemi çekirdeği üzerinde de bazı değişiklikler yapılmıştır. Bu sayede *iptables* kuralları içerisinden geçen paketlerin sisteme etkisi en aza indirilmeye çalışılmıştır. Örneğin *syn flood* diye bilinen en etkili *DoS* saldırı şeklinin önlemi *syn cookie* olarak adlandırılır[46]. Bunun önlemi hem *iptables* tarafında hem de çekirdek tarafında alınmıştır. Bir şekilde kurallar geçersiz kalırsa sistem bu saldırılara karşı korunmaya devam edecektir.



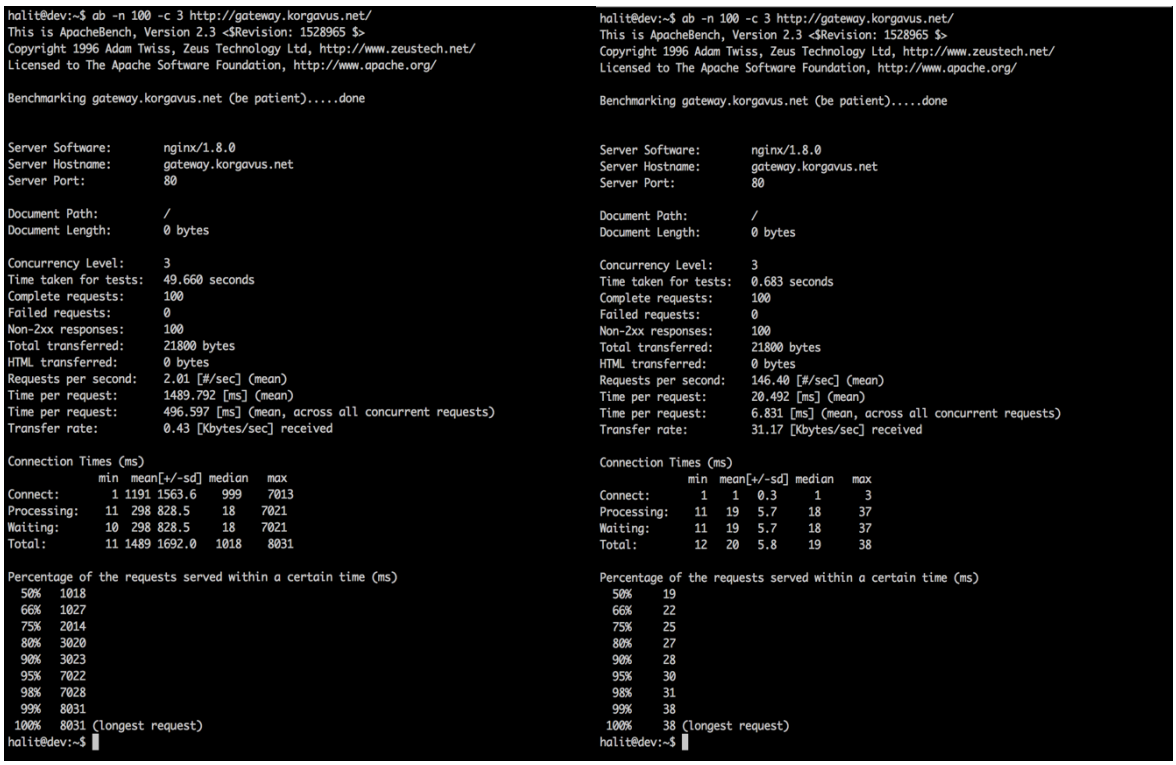
Şekil 2.52. Çekirdek parametreleri.

2.10 Testler

2.10.1 Sürdürebilirlik Testleri

Geçidin geliştirilmesi ve yapılandırmaları bittikten sonra sürdürebilirlik testleri gerçekleştirilmiştir. Bu testler içerisinde ani elektrik kayıplarının sistem üzerinde oluşturacağı etkilerden, bir şekilde diskin bozulması durumunda alınacak önlemler bulunmaktadır. Servislerin sürdürebilirliği *supervisor* aracına bırakılmıştır. Burada servislerin sürekli olarak çalışıp çalışmadığının kontrolünü gerçekleştirmektedir ve herhangi bir durumda duruma müdahale ederek sistemin çalışmaya devam etmesini sağlamaktadır.

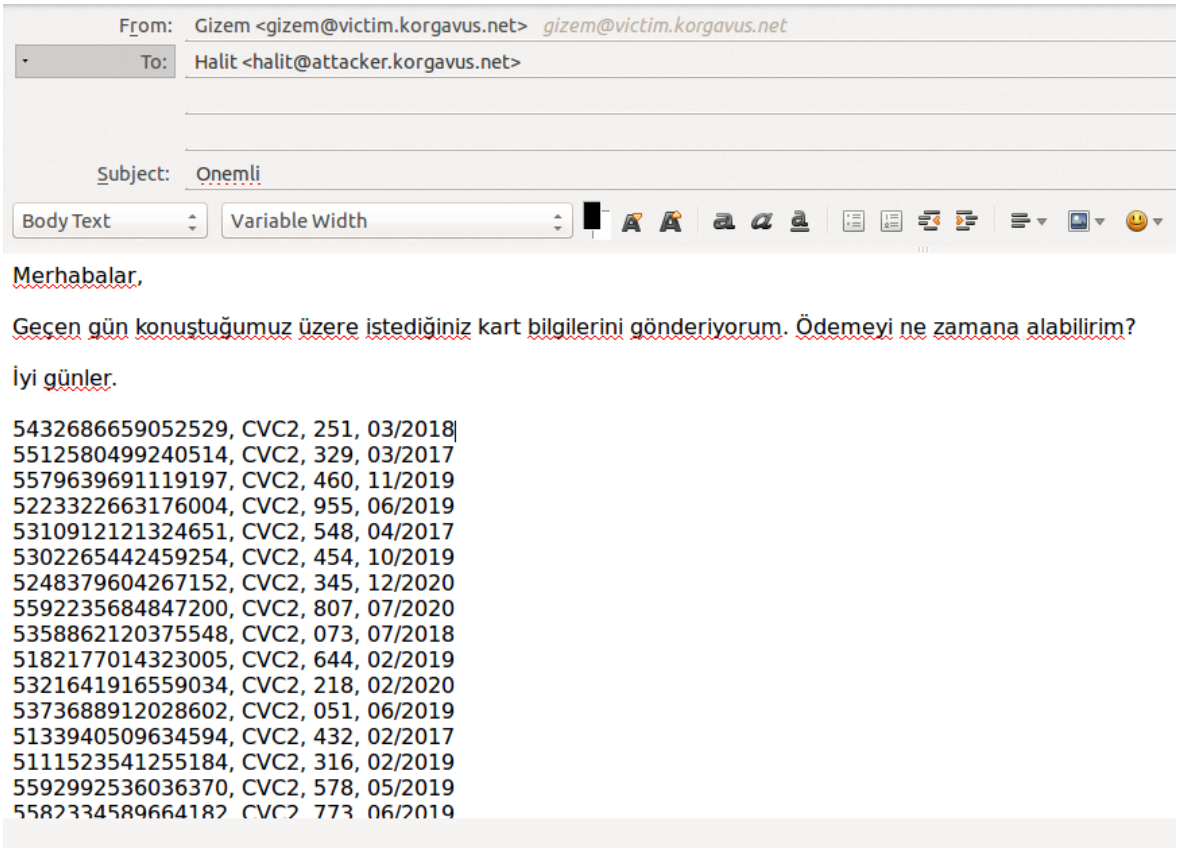
Panel için eklenen bağlantı kuralları *ab* isimli araç ile test edilmiştir. Buna göre aynı anda 3 ve toplamda 5 bağlantıya izin veren sistem ilk olarak *iptables* kuralı olmadan test edilmiştir. Bu durumda sistemden alınan en uzun cevap 38 ms olmaktadır. Daha sonra kurallar devre alınmıştır ve aynı parametreler yani anlık 3 toplamda 100 istek ile çalıştırılmıştır. Bu seferde alınan en geç cevabın süresi 8031 ms olmaktadır. Bu durum Şekil 2.53 ile görülebilir.



Şekil 2.53. Panel bağlantı testi.

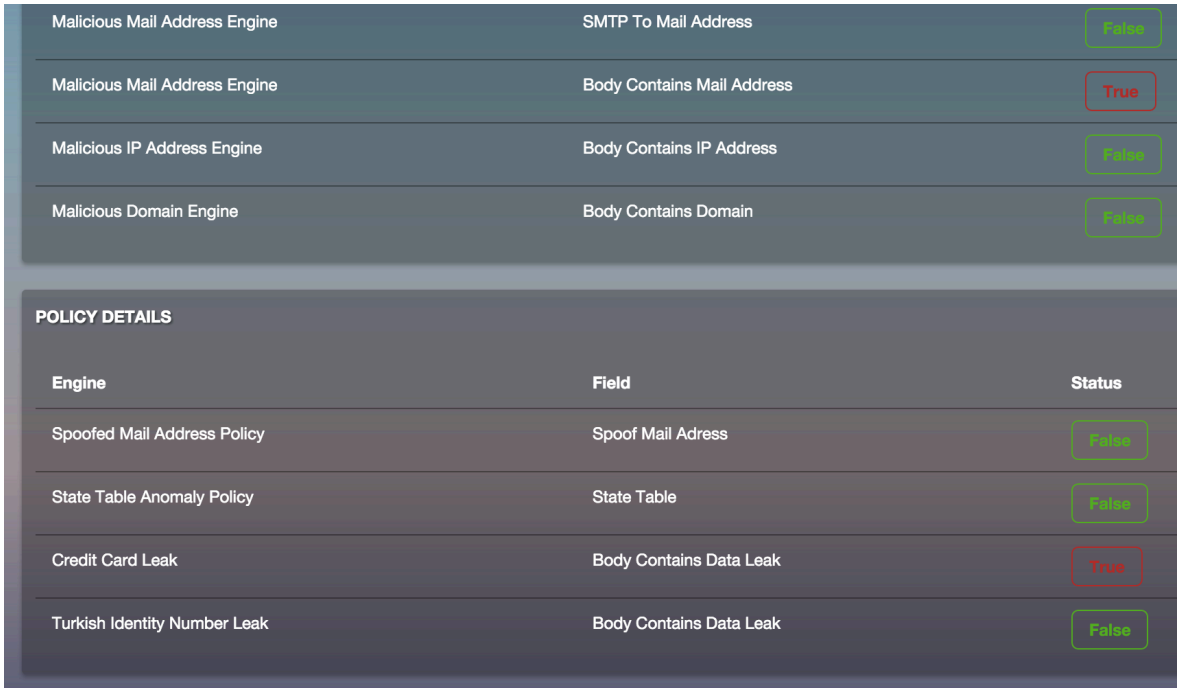
2.10.2 Örnek Senaryo Uygulamaları

Projenin çalışmasının izlenmesi için çok sayıda örnek senaryo denemesi yapılmıştır. Ancak kirlilik yaratmamak adına tüm hepsi burada gösterilmemektedir. Bunlar içinden en kritik olabileceklerden birisi veri kaçaklarıdır. Bunun için proje içerisinde kurallar bulunmaktadır. En önemli ve kritik veri kaçaklarının başında kredi kartı gelmektedir. Önlenmesi için tüm 16 haneli sayıları değerlendirmek yanlıştır. Bu yüzden algoritmasına göre kontrol edilmesi gerekir. Örneğin iç ağdan Şekil 2.54'te gösterildiği gibi bir veri kaçağı e-postası gönderilsin.



Şekil 2.54. Örnek veri kaçağı e-postası.

Bu tarz bir e-posta gönderimi gerçekleştirilen proje tarafından veri kaçağı olarak tespit edilmektedir. Aynı zamanda ilgili senaryo içerisinde gönderilen e-posta hesabı da saldırgan listesinde yer almaktadır. Bu yüzden iç ağdan ilgili e-posta adresine gönderilecek herhangi normal bir e-posta içeriği de zararlı olarak yakalanacaktır. Ancak normal bir kişiye gönderilen içerisinde veri kaçağı bulunan e-posta da sadece veri kaçağı statüsünde yakalanacaktır. Haliyle gönderilen e-postaların hem içerikleri, hem gönderen kişinin e-posta hesabı hem de IP adresleri büyük önem taşımaktadır. Sistem tüm önemli gördüğü alanlarda arama işlemi yapmaktadır. İlgili veri kaçağı e-postasının yakalandıktan sonra detay sayfasında nasıl gözüktüğü Şekil 2.55 ile görülebilir.



Şekil 2.55. Yakalanan e-postanın detay sayfası.

Panel arayüzü gerçekleştirilen çalışmalardan sonra hiçbir yenileme ihtiyacı duymadan verileri sunmaktadır. Örneğin gelen bir e-posta doğrudan analiz kuyruğu kısmına düşmektedir ve analiz işlemi bitince analiz edilenler kısmına geçmektedir. Benzer şekilde ağ arayüzlerinin durumu da gerçek zamanlı olarak izlenebilmektedir. Bu şekilde incelenen verilerin arasında işlemci sıcaklığı, disk durumu, bellek durumu gibi değerler de vardır. Arayüzün son hali Şekil 2.56 ile görülebilir.

Şekil 2.56. Panel arayüzünün çalışır durumdaki hali.

3. SONUÇLAR

Proje kapsamında tasarımı yapılan tüm aşamalar başarıyla tamamlanmıştır. Kurulan test ağ ortamında yer alan iki sunucunun birinden diğerine gönderilen e-postalar analiz işlemine tabi tutulabilmektedir. Analiz işlemleri sırasında elde edilen sonuçlar başarıyla panel içerisinde gösterilebilmektedir. Analiz motorlarının daha performanslı çalışması için gerekli tüm çalışmalar yapılmıştır. Bu amaçla kendisini ispat etmiş kütüphanelerden faydalanılmıştır. Haliyle ortaya çıkan sonu başarılı olmuştur. Geliştirme sürecinin kolay geçmesi açısından yazılan harici betikler de başarıya ulaşmada büyük etken olmuştur.

4. ÖNERİLER

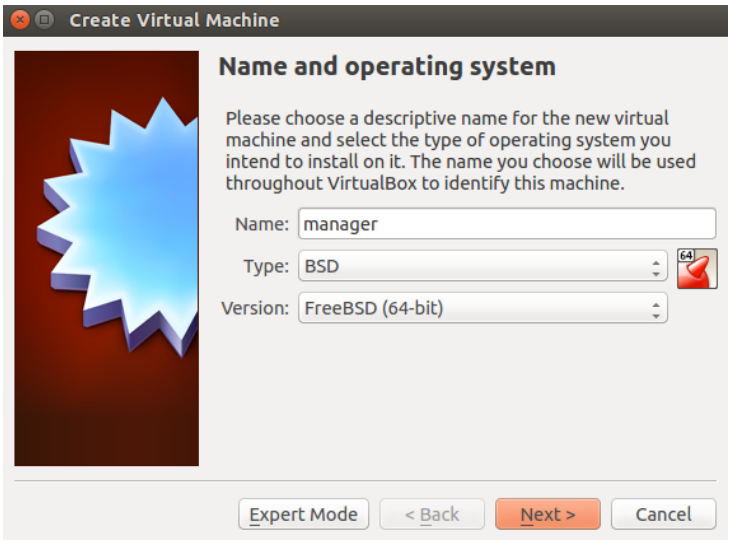
Korgavus projesi özgür ve açık kaynak kodlu güvenli ağ geçidi uygulamalarının ilkel bir prototipidir. Üzerinde gerçekleştirilen analiz işlemleri sadece çok temel düzeydedir. Gerçek anlamda yapılması gereken çok şey vardır. Örneğin ek dosyalarının içeriklerinin taranması, dosya türlerine göre farklı analiz işlemlerinin yapılması ve e-posta sunucularına yönelik yapılabilecek saldırıları engelleme gibi eklenmesi gereken çok özellik vardır. Zaten tüm bu özellikler proje kapsamında düşünülmüştür ancak minimum gereksinimlerin sağlanması için atlanmıştır.

5. KAYNAKLAR

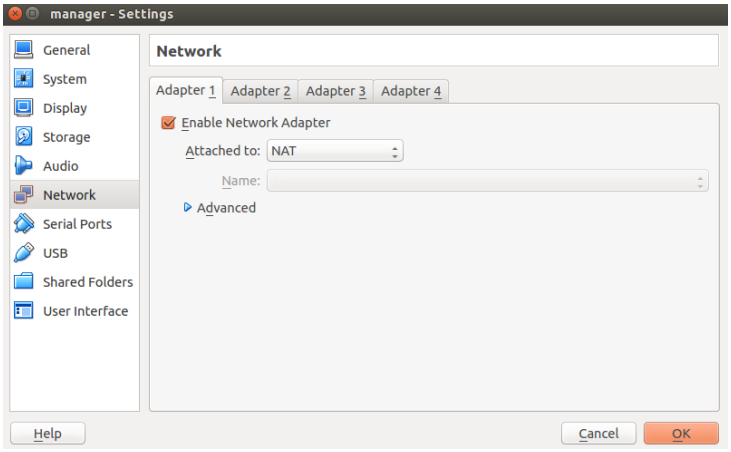
[1] https://en.wikipedia.org/wiki/Simple_Mail_Transfer_Protocol
[2] https://en.wikipedia.org/wiki/Post_Office_Protocol
[3] https://tr.wikipedia.org/wiki/IMAP
[4] http://www.postfix.org/documentation.html
[5] http://www.dovecot.org/
[6] https://en.wikipedia.org/wiki/Microsoft_Exchange_Server
[7] https://www.zimbra.com/
[8] https://roundcube.net/
[9] https://www.mozilla.org/en-US/thunderbird/
[10] http://www.netsq.com/documents/DIDS_DOE_91.pdf
[11] https://www.snort.org/
[12] https://en.wikipedia.org/wiki/Phishing
[13] http://semver.org/
[14] https://www.pfsense.org/
[15] https://www.virtualbox.org/
[16] https://en.wikipedia.org/wiki/Dynamic_Host_Configuration_Protocol
[17] https://en.wikipedia.org/wiki/Domain_Name_System
[18] http://www.ubuntu.com/server
[19] http://netcat.sourceforge.net/
[20] http://archlinuxarm.org/
[21] http://www.linuxfromscratch.org/blfs/view/svn/basicnet/bridge-utils.html
[22] http://libnids.sourceforge.net/
[23] http://www.digip.org/jansson/
[24] http://linux.die.net/man/3/libuuid
[25] http://www.json.org/
[26] http://yara.readthedocs.org/en/latest/gettingstarted.html
[27] http://www.celeryproject.org/
[28] https://www.python.org/
[29] https://pypi.python.org/pypi/pyinotify
[30] https://www.djangoproject.com/
[31] http://getbootstrap.com/2.3.2/
[32] https://jquery.com/
[33] https://www.mongodb.org/
[34] https://www.raspberrypi.org
[35] http://www.postgresql.org/
[36] https://www.rabbitmq.com/
[37] http://gunicorn.org/
[38] http://nginx.org/
[39] http://supervisord.org/
[40] http://docs.fabfile.org/en/1.10/
[41] http://www.fsarchiver.org/Main_Page
[42] https://en.wikipedia.org/wiki/Rsync
[43] http://www.clamav.net/
[44] https://en.wikipedia.org/wiki/Denial-of-service_attack
[45] https://en.wikipedia.org/wiki/Iptables
[46] https://en.wikipedia.org/wiki/SYN_flood

EK 1

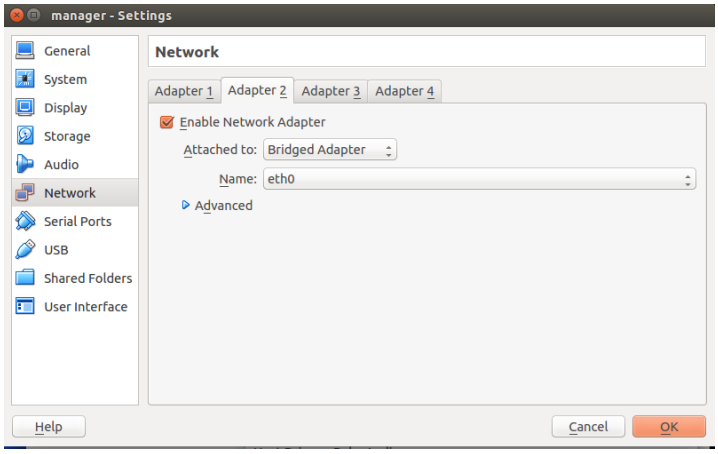
Pfsense, *FreeBSD* tabanlı, açık kaynak kodlu yazılımsal bir güvenlik duvarıdır. Web yönetim arayüzü sayesinde yönetimi oldukça basittir. Sistem kurulduktan sonra yapılandırma işlemleri web arayüzü sayesinde gerçekleştirilir. Proje kapsamında ağ yönetiminde *pfsense* aracı kullanılmıştır. Tüm sanal makineler Virtual Box aracı üzerinde oluşturulmuştur. Öncelikle sanal makine üzerinde yeni bir makine oluşturulmuştur. Bu makineye *manager* ismi konulmuştur. *Pfsense FreeBSD* tabanlı olduğundan Şekil 1'deki gibi seçilimler gerçekleştirilmiştir. Daha sonra network ayarlarında Adapter-1 ve Adapter-2 seçimleri Şekil 2 ve Şekil 3'te görüldü gibi yapılmıştır. Adapter-1, *NAT* ağına bağlanmıştır. Adapter-2 ise *Bridged*'e bağlanmıştır.



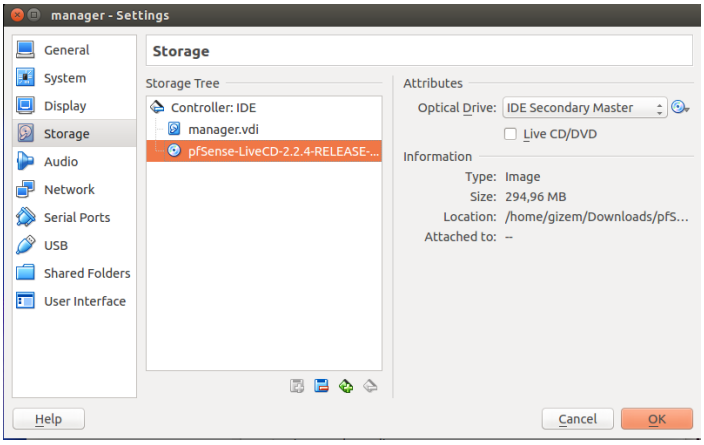
Şekil 1.



Şekil 2.

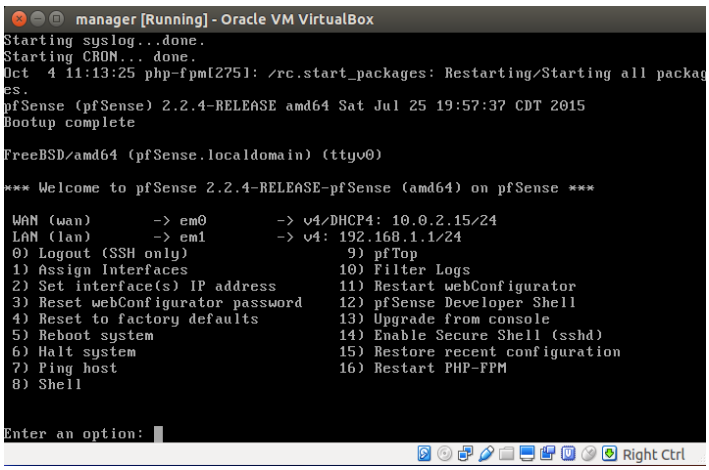


Şekil 3.

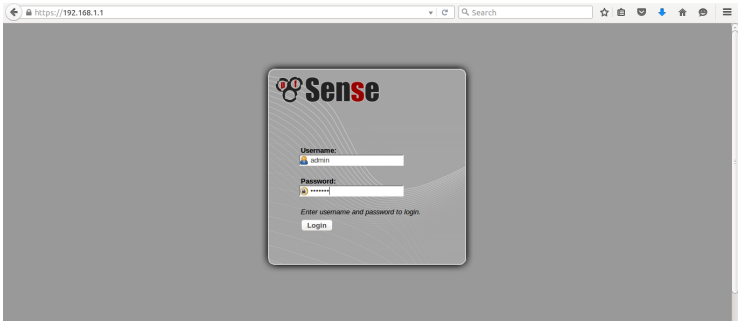


Şekil 4.

Gerekli ayarlamalar yapılarak kurulum tamamlanmıştır. Kurulum tamamlandıktan sonra ekrana Şekil 5'teki gibi bir pencere gelmektedir. Uygulamanın web arayüzüne 192.168.1.1 ip adresinden ulaşılmaktadır (Şekil 6). Sisteme giriş yapıldıktan sonra *hostname*, *domain name* ve *DNS* serverlar belirlenmiştir (Şekil 7).

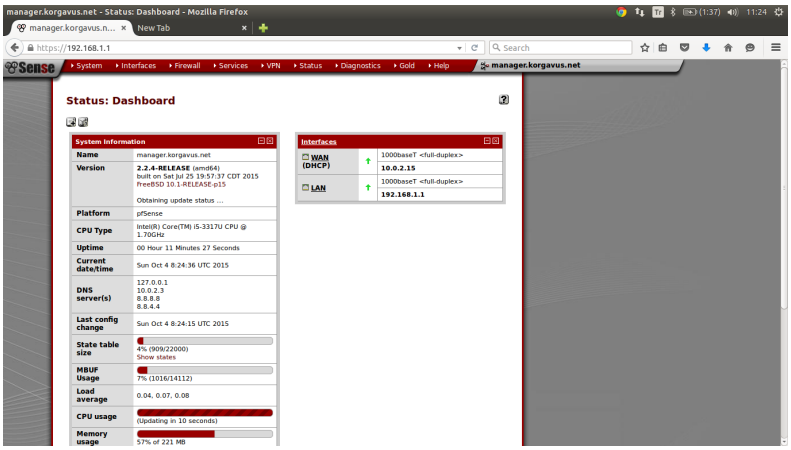


Şekil 5.



Şekil 6.

Şekil 7.



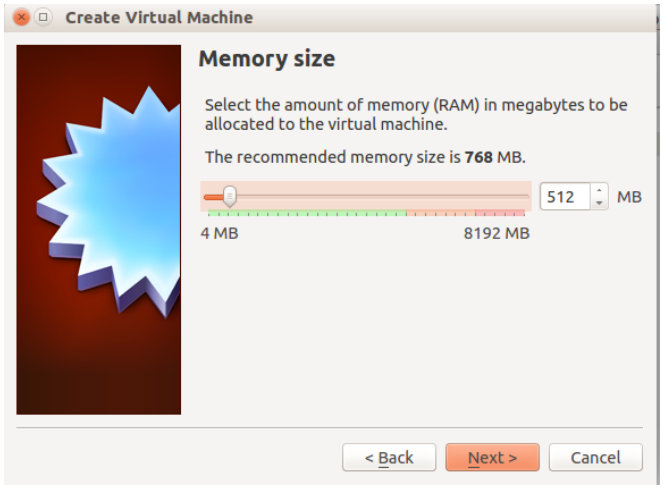
Şekil 8.

EK 2

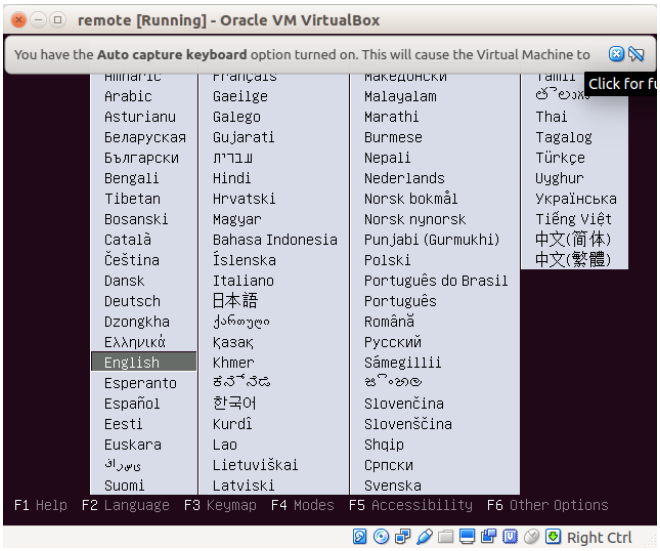
Proje kapsamında *victim*, *attacker* ve *remote* makineleri Ubuntu Server 14.04.3 olarak kurulmuştur. Bu kısımda sadace *Ubuntu Server 14.04.3* kurulumu anlatılacaktır. Bu üç makinenin kurulumu aynıdır. Ubuntu server kurulumuna başlanırken *virtual box* üzerinde yeni bir makine oluşturulur. Makinenin ismi, işletim sisteminin tipi ve işletim sisteminin versiyonu belirlenir (Şekil 1).

Şekil 1.

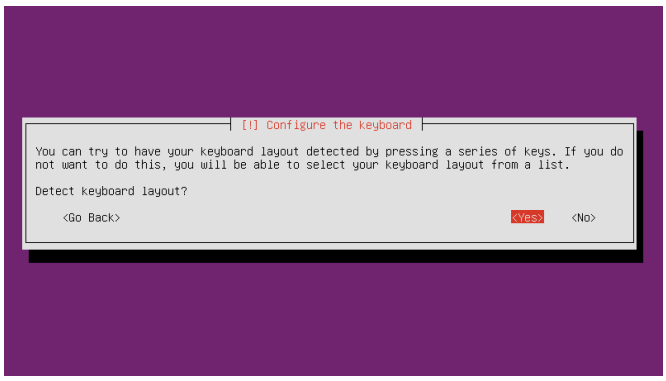
Bellek miktarı olarak tüm sunucular için 512MB seçilmiştir. Aynı zamanda disk miktarı olarak da 8GB seçilmiştir. Bu seçimler tamamlandıktan sonra kurulum aşamalarına geçilmiştir. Dil olarak İngilizce seçilmesine rağmen klavye Türkçe olarak yapılandırılmıştır. Kurulum aşamasında sadece *OpenSSH* harici olarak kurulmuştur. E-posta sunucusu olarak kullanılanlar da ise ek olarak *postfix* ve *dovecot* kurulumları da yapılmıştır.



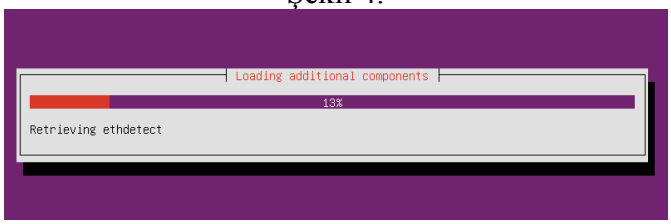
Şekil 2.



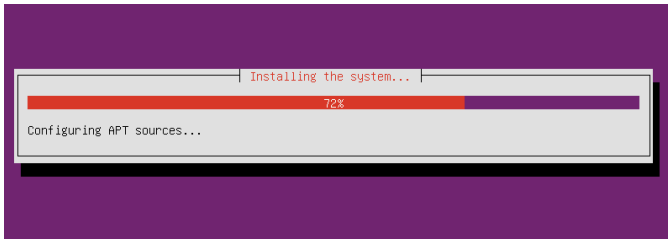
Şekil 3.



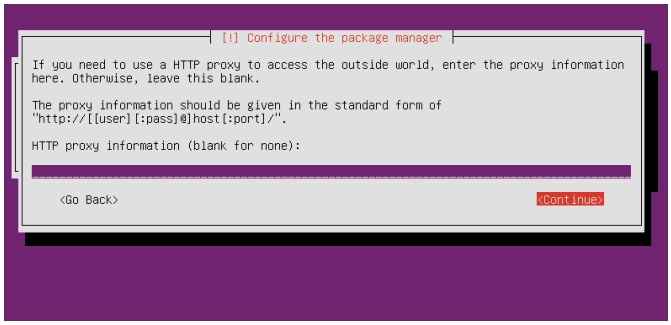
Şekil 4.



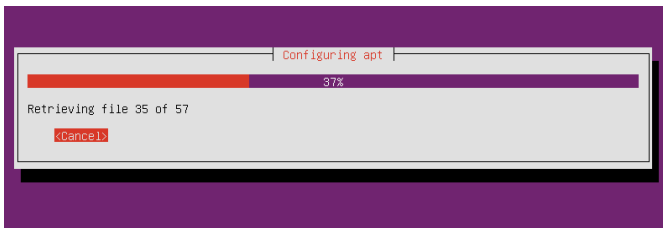
Şekil 5.



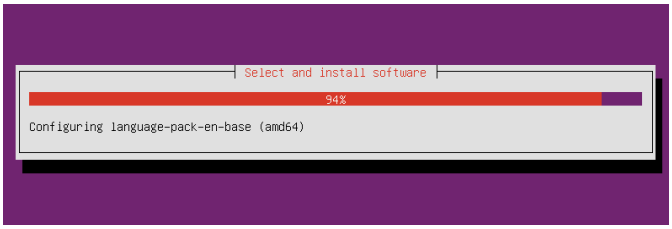
Şekil 6.



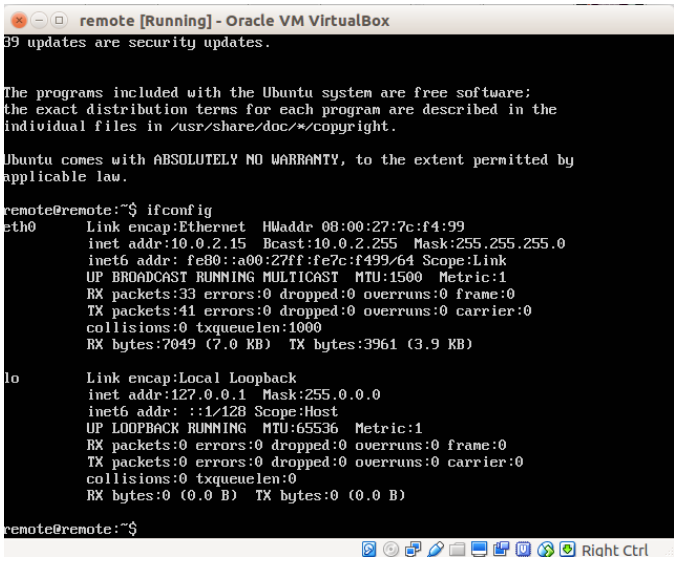
Şekil 7.



Şekil 8.



Şekil 9.



Şekil 10.

EK 3

Raspberry PI 2 cihazına *Arch Linux ARM* işletim sisteminin kurulumu gayet kolay bir şekilde tamamlanabilir. Öncelikle cihazımıza bağlayacağımız SD kart içeriğini, işletim sisteminin rahatlıkla çalışabileceği şekilde ayarlamamız gerekiyor. Bunun için Linux dağıtımlarında genellikle bulunan *fdisk*, *cfdisk*, *cgdisk* gibi araçlar kullanılabilir. *Korgavus* platformunun çalışması için kullanacağımız test cihazımızda, disk üzerinde iki adet bölüm oluşturulmuştur. Bu disk bölümlerinden ilki 100M boyutunda *W95 FAT32* tipinde parçalanmıştır. Diskin geri kalan kısımı normal olarak parçalanmıştır.

Bölüm tablosu oluşturup, disk içerisine yazıldıktan sonra bölümlerin formatlanması gerekmektedir. İlk disk bölümü *FAT*, ikincisi *EXT4* tipinde formatlanmıştır. İlk bölümün formatlanması ve sistem içerisine bağlanması için aşağıdaki komutlar kullanılmıştır. Parçalanan diskin */dev/sdc* yolunda olduğuna dikkat edilmelidir.

```
mkfs.vfat /dev/sdc1
mkdir boot
mount /dev/sdc1 boot
```

İkinci disk bölümü, *Arch Linux* işletim sisteminin barınacağı yer olduğundan *EXT4* disk formatı kullanılmıştır. Bölümün oluşturulup, sisteme dahil edilmesi için aşağıdaki satırlar kullanılmıştır.

```
mkfs.ext4 /dev/sdc2
mkdir root
mount /dev/sdc2 root
```

STANDARTLAR ve KISITLAR FORMU

Projenin hazırlanmasında uyulan standart ve kısıtlarla ilgili olarak, aşağıdaki soruları cevaplayınız.

1. Projenizin tasarım boyutu nedir? (Yeni bir proje midir? Var olan bir projenin tekrarı mıdır? Bir projenin parçası mıdır? Sizin tasarımınız proje toplamının yüzde olarak ne kadarını oluşturmaktadır?)

Projemiz tamamen yeni bir projedir. Sıfırdan tasarlanıp oluşturulmuştur.

2. Projenizde bir mühendislik problemini kendiniz formüle edip, çözdünüz mü? Açıklayınız.

Proje kapsamında paketlerin ham olarak toplanıp e-posta trafiğine döndürülmesi işlemi tamamen bizim tarafımızdan yapılmıştır.

3. Önceki derslerde edindiğiniz hangi bilgi ve becerileri kullandınız?

İşletim sistemleri dersi ile sunucu kurulumu, bilgisayar ağları dersi ile ağ kurulumu, veritabanı dersi ile veritabanı tasarımı, programlama dilleri ve programlamaya giriş kısımları ile uygulamaların geliştirilmesi kısımları.

4. Kullandığınız veya dikkate aldığınız mühendislik standartları nelerdir? (Proje konunuzla ilgili olarak kullandığınız ve kullanılması gereken standartları burada kod ve isimleri ile sıralayınız).

RFC 821 standartına bağlı kalarak e-posta trafiği ayrıştırılmıştır.

5. Kullandığınız veya dikkate aldığınız gerçekçi kısıtlar nelerdir? Lütfen boşlukları uygun yanıtlarla doldurunuz.

a) Ekonomi

Maaliyeti minimum yapabilmek için işletim sistemi tarafında bellek tüketimi minimuma indirilmeye çalışılmıştır. Bellek tüketimi içinde uygulama seçimi özenle gerçekleştirilmiştir. aynı zamanda işlemci saat frekansı arttırılarak aynı cihaz ile daha yüksek işlem kapasitesine ulaşılmıştır.

b) Çevre sorunları:

Projenin yapısı gereği herhangi bir çevre sorunu bulunmamaktadır.

c) Sürdürülebilirlik:

Sürdürebilirliği sağlamak için irili ufaklı çok sayıda kod parçası yazılmıştır. Aynı zamanda yazılımları ve servisleri sürekli ayakta tutacak harici araçlardan faydalanılmıştır. Ek olarak da disk ve bellek sağlığının korunması için ufak kod parçaları kullanılmıştır.

d) Üretilebilirlik:

Tasarlanan tüm sistem toplu üretime hazır olacak şekilde tasarlanıp gerçekleştirilmiştir. Yardımcı araçlar sayesinde tamamen sıfırdan yeni bir cihaz oluşturma süresi olabilecek minimum zamana çekilmiştir.

e) Etik:

Projenin tüm kısımlarında özgür ve açık kaynak kodlu yazılımlar kullanıldığı için herhangi bir etik ihlali bulunmamaktadır.

f) Sağlık:

Proje yapısı gereği herhangi bir sağlık sorunu teşkil etmemektedir.

g) Güvenlik:

Proje güvenli bir e-posta trafiği oluşturma üzerine tasarlanıp gerçekleştirilmiştir.

h) Sosyal ve politik sorunlar:

E-posta güvenliği kaynaklı sosyal ve politik sorunların çözümünde proje yapıcı bir rol üstlenebilir.