

# qmail 1.0 'ın Üzerinden Geçen 10 Yılın Ardından Güvenlik Konusunda Bazı Düşünceler\*

**Metin KAYA**

EnderUNIX Üyesi

Endersys Yazılım Mühendisi

metin at enderunix.org

metin.kaya at endersys.com.tr

<http://www.enderunix.org>

<http://www.endersys.com.tr>

11 Kasım 2007 Pazar EEST 03:23:19

\* D. J. Bernstein 'ın <http://cr.yp.to/qmail/qmailsec-20071101.pdf> adresli makalesinden birebir çeviridir.

## ÖZET

qmail, 1997 'den beri güvenlik garantisi olan ve yaygın olarak kullanılan bir e-posta sunucu yazılımıdır. Bu makalede qmail 'in yazarı qmail mimarisinin güvenlik ile ilgili kısımlarını ve tarihini tekrar gözden geçirir, qmail 'in başaramadığı bölümleri standartları açıkça ifade eder, qmail 'in bu başarısızlığını mühendislik açısından inceler ve güvenli program yazmaya dair gelecekle ilgili düşüncelerini belirtir.

### Kategoriler ve Konu Tanımlayıcıları

D.2.11 [Yazılım Mühendisliği]: Yazılım Mimarileri – açıkları kapatma, gereksiz kodları silme; D.4.6 [İşletim Sistemleri]: Güvenlik ve Koruma; H.4.3 [Bilgi Sistemleri ve Uygulamaları]: İletişim Uygulamaları – e-posta

### Genel Açıklamalar

Güvenlik

### Anahtar Sözcükler

Açıkların kapatılması, kodların elenmesi, güvenilen (trusted) kodların azaltılması

## 1. GİRİŞ

### 1.1 Ayın Açığı (the bug-of-the month) Kulübü

Bütün İnternet Servis Sağlayıcıları (İSS) e-posta iletim yazılımı (MTA – Mail Transport Agent) kullanır. MTA, yerel kullanıcılardan aldığı e-postaları karşı tarafa SMTP (Simple Mail Transfer Protocol) üzerinden gönderir. Ayrıca, kendisi de SMTP üzerinden aldığı e-postaları yerel kullanıcılara iletir.

Eric Allman 'ın Sendmail yazılımındaki güvenlik açıklarından bıktığım için 1995 'te bir MTA, qmail, yazmaya başladım. Sendmail, o zamanlar internetin en popüler MTA 'yi konumundaydı; [4] 'e bakabilirsiniz. Aşağıda, Aralık 1995 'te yazdığım qmail dokümanından bir parça görülmektedir:

*Her birkaç ayda bir CERT tarafından Sendmail ile ilgili yerel veya uzak kullanıcıların makinenin kontrolünü ele geçirmesine izin veren güvenlik açıkları duyuruluyor. Sendmail 'de keşfedilmeyi bekleyen çok fazla sayıda güvenlik açığının bulunduğundan eminim. Çünkü Sendmail 'in tasarımı, 41000 satır koddaki en küçük bir hatayı çok büyük bir güvenlik tehlikesine dönüştürüyor. Smail gibi diğer e-posta yazılımları ve Majordomo benzeri e-posta liste yöneticisi yazılımlar da aynı şekilde kötü görünüyor.*

1996 ve 1997 'de 14 Sendmail güvenlik açığı duyuruldu. Bundan sonra Sendmail 'in güvenlik açıklarını saymayı ve Sendmail ile ilgilenmeyi bıraktım. Araştırmalara göre Sendmail 'in güvenlik açıklarına karşı çıkardığı son sürümü olan 8.13.6, Mart 2006 'da çıkarıldı; [10] 'a bakabilirsiniz ("yetkilendirilmemiş uzak saldırganlar Sendmail sürecinin haklarıyla sistemde istedikleri kodu çalıştırabiliyor").

Sendmail 'in 20 yıldır uzak saldırılara açık olduğu biliniyor; peki şimdi kimse Sendmail 'in son sürümünün uzak saldırılara karşı güvenli olduğunu iddia edebilir mi? Sendmail 'le ilgili güvenlik açıklarının duyurulması seyrekleşmiş olabilir; ancak bu gerçek, sistemleri Sendmail yüzünden saldırıya uğramakta olan yöneticilere yardımda bulunamaz.

## 1.2 qmail 'in Çıkarılması

Aralık 1995 'te yoğun olarak qmail için kod yazmaya başladım. Cebirsel sayı teorisiyle ilgili verdiğim ders henüz bitmişti ve kendime ayırabileceğim boş zamanım olmuştu. Beni qmail yazmaya iten son şey ise bir meslektaşımın onun için büyük bir e-posta liste programı yazacağına dair söz vermemdi. Sendmail, listeler için istediğim kolaylığı sunmuyordu. Sendmail 'in güvenilirlik ve güvenlik sorunları bir yana, büyük bir liste üyelerine e-posta göndermek sonsuza dek sürecekmış gibi görünüyordu (ciddiyim!).

qmail 'in 7 Aralık 1995 sürümündeki kodda aşağıdaki komuta göre ölçüldüğünde 14903 kelime bulunuyordu:

```
$ cat *.c *.h | cpp -fpreprocessed sed 's/[_a-zA-Z0-9][_a-zA-Z0-9]*/x/g' |  
tr -d '\012' | wc -c
```

(Diğer ölçümlerde de benzer sonuçlar alınır.) qmail 'in 21 Aralık 1995 sürümü ise 36062 kelimeydi. 21 Ocak 1996 sürümünde, qmail 0.70, 74745 kelime vardı. Bu sürümü kendi bilgisayarımda birkaç gün izledikten sonra qmail beta testlerini başlatmak için herkese sundum.

1 Ağustos 1996 'da 105044 kelimeye sahip qmail 0.90 sürümünü çıkararak beta testlerini bitirdim. 20 Şubat 1997 ise 117685 kelimelik qmail 1.00 'ı yayınladığım tarihtir. 15 Haziran 1998 'de 124520 kelimelik mevcut sürümü, qmail 1.03, çıkarttım. Bu sürümün türevi olan 124911 kelimelik netqmail 1.05 ise komite tarafından çıkarıldı. qmail 1.0 'daki 4 açığın farkındayım.

Karşılaştırma yapacak olursak: Sendmail 8.7.5 Mart 1996 'da çıkarıldı ve 178375 kelimeydi; 209955 kelimelik Sendmail 8.8.5 Ocak 1997 'de yayınladı; Mayıs 1998 'de çıkarılan Sendmail 8.9.0 sürümü ise 232188 sözcük içeriyordu. Tüm bu sürümlerde Sendmail 'in yüzlerce açığı not ediliyordu. Kullanıcı tarafından bakıldığında Sendmail ile qmail arasında farklılıklar var. Örneğin; qmail 'in POP desteğine karşılık Sendmail 'de UUCP desteği ve qmail 'de kullanıcılar tarafından kontrol edilen e-posta listesine karşılık Sendmail 'de uzaktan root olabilme açığı (sadece bir şaka!) mevcut. Ancak tüm bunlar karmaşıklık sorununu açıklamaz. Her paketdeki kodların çoğu, tipik internet sitelerinin ihtiyaç duyduğu MTA 'lerin çekirdek özelliklerine adanmıştır.

Parmak izleri (fingerprinting) gösteriyor ki internetteki mevcut SMTP sunucularının 1 milyondan fazlasında qmail 1.03 veya netqmail 1.05 kullanılıyor. 3. parti [qmail.org](http://qmail.org) ise şunu söylüyor:

*Çok fazla sayıda internet sitesi qmail kullanıyor: USA.net 'in giden e-posta sunucusu, Address.com, Rediffmail.com, Colonize.com, Yahoo! mail, Network Solutions, Verio, MessageLabs (kötü niyetli e-postaları tespit etme için araştırma yapan kurum haftada 100 Milyon e-posta alıyor), listserv.acsu.buffalo.edu (1996 'dan beri qmail kullanan büyük bir listserv hub'ı), Ohio State (ABD 'nin en büyük üniversitesi), Yahoo! Groups, Listbot, USWest.net (ABD 'nin batı İSS 'si), Telenordia, gmx.de (Alman İSS), NetZero (açık İSS), Critical Path, PayPal/Confinity, Hypermart.net, Casema, Pair Networks, Topica, MyNet.com.tr, FSmal.net, Mycom.com, and vuurwerk.nl.*

Birçok yazar qmail kitabı yazdı: [7], [20], [14], [22]. Çok kapsamlı istatistikleri toplamak zor; ancak örnekler gösteriyor ki internet üzerindeki yasal e-postaların büyük bir kısmını qmail alıp gönderiyor.

### 1.3 qmail Güvenlik Garantisi

Mart 1997 'de qmail 'in son sürümünde gerçekliği kanıtlanabilir bir güvenlik açığı bulan ilk kişiye 500 \$ ödül vereceğimi vaat etmiştim. Örneğin; qmail 'in başka bir kullanıcı haklarıyla çalışmasını sağlayacak bir açık. Teklifim hala geçerli. Kimse qmail 'de bir güvenlik açığı bulamadı. Bu vesileyle ödülü 1000 \$ 'a yükseltiyorum.

“qmail 'deki güvenlik açığı” ifadesi elbette qmail dışı problemleri kapsamıyor. Örneğin; NFS, TCP/IP veya DNS 'teki güvenlik sorunları, .forward dosyalarının çalıştırdığı betiklerdeki açıklar ve genel işletim sistemi sorunları. qmail kurulmadan önce zaten saldırıya açık olan bir sistemdeki qmail 'i herhangi bir problemden dolayı ayıplamak aptalca bir hareket olur.

qmail 'i e-postaları bir ağa gönderirken şifrelemede veya doğrulamada başarısız olması durumunda ayıplamak ahmakça olmaz; çünkü kriptografi TCP/IP 'den ziyade qmail gibi uygulamaların ele alması gereken bir konudur. Ancak kriptografi qmail güvenlik garantisinin kapsamı dışındadır.

DoS (Denial of Service) saldırıları özellikle kapsam dışıdır. Çünkü bunlar tüm MTA 'lerde var olan, kapsamlı belgelendirilmiş sorunlardır ve bazı önemli protokolleri ağır bir yapılandırmadan geçirmeden bu saldırıların aşılması çok zordur. Birileri buna karşı çıkıp e-postaların ümitsizce bu yapılandırmaya ihtiyacı olduğunu söyleyebilir; ben de ona katılıyorum ama bu konu başka bir makalenin konusudur.

qmail, kullanıcıların qmail 'i istismar edip diğer kullanıcıların bilgilerini çalamayacağını ve onlara zarar veremeyeceğinin garantisini verir. Eğer diğer programlar da aynı standardı yakalarsa ve ağ bağlantılarımız kriptografik olarak korumaya alınırsa internetteki tek güvenlik sorunu DoS saldırıları olur.

## 1.4 Bu Makalenin İçeriği

qmail emsalsiz güvenlik seviyesine ulaşmayı nasıl başardı? qmail 'i güvenlik açısından üstün kılan neydi ve neler onu daha ileri götürebilirdi? Diğer yazılım projelerinin bu şekilde güvenlik garantisine sahip olması için onları nasıl yapılandırabiliriz?

Güvenlik hakkındaki görüşlerim yıllar geçtikçe çok daha acımasız oldu. Güvenliğe çok büyük para ve emeğin yatırıldığını gördüm ve çok daha fazlasının harcanacağından eminim. Güvenlik konusundaki çabaların çoğu dünün saldırılarını önlemek üzere tasarlanır; fakat yarının saldırıları karşısında başarısız olur ve gayet sağlam olması beklenen yazılımlarda kullanılmaz. Uzun vadeli işlerde bu çabalar oyalayıcı olmaktan öteye geçmez.

Geçmişe bakıldığında qmail 'in güvenlik mekanizmalarından bazılarının yarı pişmiş olduğu, ihmal edildiğinde hiçbir güvenlik kaybının olmaması ve bazı şeyleri gerçekten başaramamış olmasından anlaşılır. Diğer mekanizmalar qmail 'in göstermiş olduğu başarılı güvenlik performansından sorumludur. Bu makaledeki asıl maksadım aradaki farkı ileri seviyede açıklamaktır – yazılım mühendisliği teknikleri uzun soluklu güvenlik açısından nasıl değerlendirilmelidir.

Bu makalenin 2. kısmında çok sağlam yazılım sistemlerine yönelik 3 özel talimat anlatılmaktadır. Geriye kalan bölümler, qmail 'in bu 3 talimatta göre başarılı ve başarısız olduğu yönlerden bahsetmektedir.

## 2. NASIL İLERLEYEBİLİRİZ?

Bir yazılım hatası, tanımda belirtildiği üzere kullanıcının gereksinimlerini bozmak demektir. Yazılımla ilgili güvenlik açığı da kullanıcının güvenlik gereksinimlerini bozan yazılım özelliğidir. Bu nedenle her güvenlik açığı bir hatadır.

(Resmi şartnamelerin, güvenlik politikalarının, vb şeylerin savunmalarında açıkça belirtilmiştir ki kullanıcıların tüm ihtiyaçlarını yapmak oldukça zordur. – Özellikle de kullacıların üzerinde çok düşünmediği gri alanlarda-. Aşağıda belirtildiği gibi, bu karmaşıklık güvenlik açıklarını gidermede büyük zorluklar çıkarmaktadır ama bu karmaşıklık olmasa da güvenlik açıklarını çözme işi yine zor olurdu. Çünkü bugünkü yazılımlar aklıma gelen en basit güvenlik gereksinimlerini bile karşılamaktan uzaklar.)

Her  $N$  adet kod sözcüğünde ortalama 1 açığın ve bilgisayarınızda  $10000N$  adet kod sözcüğü olduğunu varsayalım. Mutsuz sonuç: bilgisayarınızda 10000 açık var ve bunların büyük çoğunluğu tahminen güvenlikle ilgili.

Hiç güvenlik açığı bırakmayacak şekilde nasıl ilerleyebiliriz? Bu ilerlemeyi nasıl ölçebiliriz? Buna bölüm 3 yanıt vermektedir. Ayrıca komitenin dikkatini dağıtıp ilerlemesine engel olan 3 konuyu da tartışmaktadır.

## 2.1 Yanıt 1: Açıkları Gidermek

İlk ve en açık yanıt: açık oranını azaltmaktır.

Bir yazılım mühendisliği sürecinin açık oranını, o ana bitmiş kodları dikkatlice gözden geçirip bulunan açıkların sayısını gözden geçirilen kod miktarının fonksiyonu şeklinde ifade ederek bulabiliriz. Farklı yazılım mühendisliği süreçlerinin açık oranını karşılaştırabiliriz. Düşük açık oranlarıyla meta-engineering yapabiliriz.

(Bazen göze batmayan küçük hatalar, kodu gözden geçirirken araya kaynamaktadır. Ancak; deneyimlerim normal açık oranının, gözden hiç hata kaçırmadığınızda bulacağınız açık oranından çok az farklı olacağını söylüyor. Eric Raymond ‘un [17, Bölüm 4: “Release Early, Release Often”] ‘da belirttiği gibi “mevcut göz yuvarlaklarımızda bütün açıklar sığ kalır”.)

Açıkları giderme, yazılım mühendisliği araştırmalarının klasik bir konusudur. İyi bilinen bir örnek: tipik yazılım mühendisliği sürecindeki açık oranı kapsam testleri yapılarak ciddi oranda azaltılabilir. Fakat yazılım mühendisliği sürecinin önceki aşamalarında yaptığınız tercihlerin açık oranını azaltacağı çok bilinen bir kavram değildir. Detaylı bilgi için 3. kısma bakınız.

## 2.2. Yanıt 2: Kod Elemek

10000N adet kod sözcüğü olan bilgisayar sistemini tekrar göz önüne alın. Açık oranının kelime başına 1 ‘den biraz daha az olduğunu ama tüm açık miktarının yine 10000N ‘e yakın olduğunu varsayalım. Bu durumu nasıl düzeltebiliriz?

2. yanıt, bilgisayardaki kod miktarını azaltmaktır. Yazılım mühendisliği süreçleri sadece kod miktarındaki açık sayısına göre değil, kullanıcının istediği özellikleri sağlayan kod hacmine göre de değişim gösterir. Kod eleme, yazılım mühendisliği araştırmalarındaki diğer klasik konudur: farklı yazılım mühendisliği süreçlerinin kod hacimlerini karşılaştırabiliriz ve meta-engineering ile aynı işi daha az miktardaki kodla yapabiliriz. Ayrıntılar için makalenin 4. kısmına bakınız.

Bir ürünün açık (bugs)/kod oranını veya kod/iş (jobs) oranını ölçmektense açık/iş oranını gözlemek daha yerinde bir hareket olacaktır. Ancak farklı ölçümler farklı tekniklere dikkati çekecek gibi görünüyor. Ayrıca, kod/iş oranı yazılım mühendisliği zamanında önceden tahminlerde bulunacak kişiden bağımsız bir konudur ve aynı şekilde açık/kod oranı da hata ayıklama zamanında öngörülerde bulunacak kişiden bağımsızdır.

### 2.3 Güvenilen Kodların Azaltılması

Daha az miktarda kodu olan bir bilgisayar sistemi ve daha az açıklık olduğunu varsayalım. Bu durumda sistemimizde 1000 'den az açık olacaktır. Bunların çoğu, tahminen, hala güvenlik açıklarıdır. Nasıl ilerleyebiliriz?

3. yanıt, bilgisayar sistemindeki güvenilen kod miktarını azaltmaktır. Bilgisayar mimarisini kodların çoğunu *güvenilmezler* hapisanesine atarak yeniden tasarlayabiliriz. *Güvenilmez* koddan kastım ne yaptığıyla, ne kadar kötü çalıştığıyla, ne kadar açıklığının olduğuyla hiç ilgilenmediğiniz, kullanıcının güvenliğini istismar edemeyecek olan koddur. Bilgisayarımızdaki güvenilen kod miktarını ölçüp meta-engineering yaparak bu miktarı en aza indirebiliriz. (Mütercimin notu: D.J.B. 'nin güvenilen kod ile anlatmak istediği, program yazarken kullanılan kütüphane fonksiyonları gibi hazır kodlardır. D.J.B; *strlen*, *strcpy*, *strcmp*, v.s. başta olmak üzere özellikle katar (string) ve sistem fonksiyonlarını yeniden yazmıştır.)

Bir sistemdeki kod miktarını azaltmak için kullanılan teknikler güvenilen kod miktarını azaltmak için de geçerlidir ama ilave teknikler güvenilen kod miktarının toplam kod miktarından daha az olmasını sağlamaktadır. Güvenilen kod miktarını azaltmakla açık sayısını azaltmak arasında hoş bir sinerji var: güvenilen kod miktarı az olduğundan bunlardaki açıkları gidermek nispeten daha ucuzdur.

Örneğin; Sendmail 'in e-postanın başlığından e-posta adresini çıkarmaktan sorumlu olan kodunu göz önüne alalım. [12] 'ye göre, Mark Dowd 2003 'te bu kodda aşağıdaki güvenlik açıklığını buldu:

*Saldırganlar, root veya superuser kullanıcısının haklarını ele geçirip Sendmail sunucusunu kendi çıkarları için kullanabiliyor... Bu açık özellikle çok tehlikeli; çünkü saldırgan sadece bir e-posta göndererek hedef sunucu hakkında hiçbir bilgiye sahip olmaksızın sunucuyu taciz edebilir... X-Force bu açıklığın gerçek dünyadaki Sendmail kurulumlarında uygulanabileceğini gösterdi.*

Aynı adres çıkarma kodunun basit 2 veri akış kuralını kullanan bir yorumlayıcıyla çalıştığını varsayalım:

- Kodun sistemde görebildiği tek yer üzerinde çalıştığı e-postadır;
- Kodun sistemde sahip olabileceği tek etki, üzerinde çalıştığı e-postadan elde ettiği e-posta adresini yazmaktır.

Bu aşamadan sonra kodun, kullanıcının güvenliğini kötüye kullanma şansı yoktur. Sisteme e-posta gönderen saldırganın elinde tuttuğu tek şey kodun çıkarttığı e-posta adresidir ki saldırgan bunu koddaki herhangi bir açıklığı kullanmadan da elde edebilir. Saldırganın başka hiçbir şey yapabileme şansı yoktur. Aynı konuyla ilgili başka bir örneği kısım 5.2 'de bulabilirsiniz.

(Kullanıcının güvenlik gereksinimlerinin e-posta göndericisinin e-posta adresi çıkarma kontrolünü saldırganın elinden almadığını farz ediyorum ama gereksinimler daha karmaşık olabilir. Belki de kullanıcı birden fazla kaynaktan bir e-posta eklentisi oluşturuyordur. Belki de kullanıcı e-posta eklentilerinin birbirini ve e-posta başlığı gibi özellikleri etkilememesini istiyordur. Bu durumda yorumlayıcı, ilgili veri akış kısıtlarını göz önünde bulundurmak zorundadır.)

Güvenilen kod miktarını ve açık oranını yeterince azalttığımızda ve güvenilen kodun makul ölçülerde açık sahibi olduğunu kabul edersek bu durumda sistemde hiç güvenlik açığı olmayacaktır. Tahminen güvenilen kodda hala açıklar olacaktır; ancak bu açıklar kullanıcının güvenliğini kötüye kullanamaz. Güvenli bir yazılım sistemi tasarlamak açığı olmayan bir yazılım sistemi tasarlamaktan daha kolaydır.

## **2.4 Dikkati Dağıtmanın 1. Yöntemi: Saldırganın Peşine Düşmek**

Çoğu insan için *güvenlik*, mevcut saldırganları izleyip bazı şeyleri (hatta her şeyi!) değiştirmek ve onların başarısız olmasını sağlamaktan ibarettir. Bu yöntemin çekiciliğini anlamak kolaydır: daha önce başarılı olmuş bir saldırının bu girişiminde başarısız olduğunu izlemek sistem güvenliğisine bir anlık mutluluk verir.

Bazen değişiklikler saldırganlar tarafından istismar edilen özel açıkları kapatmaya yöneliktir. Ubuntu Linux dağıtımı bu yıl içerisinde çok sayıda program açıklarını gideren 100 ‘den fazla acil güvenlik yaması çıkardı.

Yapılan değişiklikler bazen hiçbir problemi çözmez. Güvenlik duvarları, anti virüs sistemleri ve saldırı tespit sistemleri, hedeflenen yazılımı yamalamadan önce saldırıları girişimlerini tespit etmeye çalışır.

Yazılım mühendisliği eksikliklerini çözmeyen değişiklikler, güvenlik açıklarına öncülük yapmıştır. Başarıyı, mümkün olan tüm saldırıları önlemekten çok, dünün saldırılarını durdurmak olarak tanımlarsak, sistemimizin yarının saldırılarına karşı savunmasız olmasına şaşırılmamalıdır.

## **2.5 Dikkati Dağıtmanın 2. Yöntemi: Ayrıcalıkları En Aza İndirme**

Birçok güvenlik çabası uygulamaların en az ayrıcalıkla çalışması prensibine yöneliktir. Bu prensip [18] ‘de “Tüm program ve sistem kullanıcıları işlerini yürütebilecek en az ayrıcalıkla çalışmalıdır” diyen Saltzer ve Schroeder tarafından ortaya atılmıştır.

Bu güvenlik çabaları aşağıda anlatıldığı gibi çalışmaktadır. Program P ‘nin R kaynağına meşru erişim hakkının olmadığını varsayalım. Daha sonra işletim sistemini, P ‘ye bu kontrol hakkını verecek şekilde kullanalım (hakları genişletme de olabilir). Resim görüntüleme yazılımının ağdan veri göndermesini önleyelim, bir DNS sorgulama yazılımının diskten dosya okumasını engelleyelim, ... Örnekler için [3], [21], [11], [2], [15], [1], [16] ve [23] ‘e bakınız. Kısım 5.1 konuyla ilgili qmail örneklerini içermektedir.

En az ayrıcalıkla çalışma prensibinin temelde yanlış olduğunu düşünüyorum. Ayrıcalıkları kısıtlamak zararları azaltabilir ama hiçbir zaman güvenlik açıklarını kapatmaz. Ayrıcalıkların azaltılması güvenilen kodların azaltılması gibi değildir, aynı faydayı sağlamaz ve bizi güvenli bir bilgisayar sistemine yaklaştırmaz.

Bir örnek olarak [11] 'in Netscape 'in "DNS Helper" programına olan bağlılığını düşünün. Bu programın yerel diske erişiminin engellenmiş olması bile [8] 'de anlatılan `libresolv` açığının, Netscape 'te bir güvenlik tehlikesi oluşturmasını önleyemedi. "DNS Helper" ı hedef alan bir saldırgan, programın kontrolünü ele geçirip Netscape 'in gönderdiği bütün DNS verilerini değiştirip kullanıcının bütün web bağlantılarını çalabilir. [11] 'den önce durum, "DNS Helper" programının kullanıcının güvenlik gereksinimlerini çiğneme potansiyeline sahip olduğu ve düzeltilmesi gerektiği yönündeydi. [11] 'den sonra ise "DNS Helper" programının kullanıcının güvenlik gereksinimlerini çiğneme potansiyeline sahip olduğu ve düzeltilmesi gerektiği yönündeydi.

Güvenilmeyen (untrusted) kodun tanımında kullanıcının güvenlik gereksinimlerini çiğneyemeyen kod yatar. "DNS Helper" programını tamamen güvenilmeyen kodlarla yazmak, bu programın eriştiği işletim sistemi kaynaklarıyla ilgili kısıtları sisteme zorla kabul ettirmiş olmaktan daha az istila edicidir. "DNS Helper" birçok kaynağın verisini ele alır; ancak her kaynak bir diğer kaynağın verisini değiştiremeyecek şekilde yapılandırılmalıdır.

## 2.6 Dikkati Dağıtmanın 3. Yöntemi: Hız, Hız, Hız

Programcılar, yazılımlarının kritik olmayan kısımlarının hızını artırmak için çok uzun süre düşünür ve kaygılanır. Ancak bu girişimleri, genelde programlarının bakımı ve hata ayıklaması göz önüne alınca negatif yönde çok büyük etkiye sahip oluyor. Programın çalışma zamanını %97 'sine çekmek gibi küçük performans hamlelerini unutmuyoruz. Olgunlaşmamış iyileştirme, bütün kötülüklerin anasıdır; Knuth – [13] 'ün 268. sayfasına bakabilirsiniz.

Programcıların, yazılımlarını hızlandırmak adına yaptıkları en büyük iş kodlarından küçük parçaları kesip atmaktır.

Programcılar bunu yaptıklarında yazılımlarının daha hızlı çalıştığını ve açık oranının arttığını görebilirler. Genelde tasarım süreçlerini bu gayreti es geçecek şekilde değiştirmekten hafif bir mutluluk duyarlar. Knuth 'ın yorumları, yazılım profili çıkaramayan acemi programcılar için geçerli sanırım.

Maalesef hızlanmak adına yapılan işlerin açıkça görülmemesi ve onarılması çok güç etkileri var.

Kısım 2.3 'teki e-posta adresi çıkarma örneğini göz önüne alın. Basit veri akışı işlemleri için yorumlayıcı kullanmak güvenlik dışı hatalara neden olabilir – önemli bir fayda. Ancak programcıların çoğu "Yorumlayıcı kullanılan kod çok yavaş çalışıyor!" der ve yorumlayıcı kullanmayı hiç denemez bile.

Yeni bir işletim sistemi süreci başlatmak benzer kısıtları, yorumlayıcı kullanılsa da, kabul etmek zorundadır. Ama programcılar “Her adres çıkarma için yeni bir süreç başlatamazsın!” der ve kullanmayı hiç denemez bile.

Bir programlama dilini, program veya sistem mimarisini geliştirme girişiminde bulunanlar benzer engelleri aşmak zorundadır. Bu programcılar bazı içeriklerin yavaşlamasıyla karşı karşıya gelecektir (veya karşılaştıklarını varsayacaklardır) ve bu yavaşlıktan dolayı gelişmeyi suçlayacaktır ki bu tam bir pazarlama felaketi.

Bilgisayarımı beklemeyi sevmiyorum. Aslında başkasının bilgisayarını yüzünden bekletilmeyi de sevmiyorum. Araştırmalarımın büyük çoğunluğu sistem performansını değişik seviyelerde artırmaya yöneliktir. (Örneğin; [6] ‘daki makalem: “25519 Eğrisi: Yeni Diffie Hellman Hız Kayıtları”) Fakat daha sonra güvenliğin, hızdan önemli olduğunu anladım. Bugün, mevcut sistemlerimizden 10 kat daha yavaş çalışsa bile güvenlik açığı olmayan yazılım sistemlerine ihtiyacımız var. Yarın, bu sistemleri hızlandırmaya çalışmak için çalışmaya başlayabiliriz.

Gayet sağlam sistemler tasarladığımızda güvenliğin aslında çok fazla CPU zamanına ihtiyacı olmadığına inanıyorum. CPU zamanının büyük kısmını program kodumuzun küçük bir kısmı kullanır ve güvenlik doğrulamasının harcadığı zaman, küçük kodlarımızın harcadığı zaman yanında ihmal edilebilecek kadar azdır. Sıradan bir sıcak nokta (hot spot), tekil bir kaynaktan milyonlarca CPU saat çevrimi harcar. Modern derleme teknikleri, bazı durumları delillerle kanıtlanmış, güvenlik doğrulamalarını iç döngülerden yukarıya çekebilecektir. Ara sıra, ağ paketlerinin şifrelenmesi gibi hileli güvenlik kısıtları olan sıcak noktalar (hot spot) açığı olmayan güvenilir kod olacaktır.

### 3. AÇIKLARI KAPATMAK

Uzun yıllardır sistematik olarak hataya eğilimli program yazma alışkanlıklarını, kendimin ve diğer insanların hatalarını inceleyerek ve literatürü gözden geçirerek teşhis etmeye çalışıyorum ve programlama ortamımı bu hatalardan soyutlandırmaya çalışıyorum.

qmail ‘i yazmaya başladığımda bu konuda zaten birikimim vardı ve qmail 1.0 ‘ı geliştirirken büyük bir ilerleme kaydettim. Buradan qmail ‘in açık oranından memnun olduğum anlaşılmasın. Hata yapma oranım son on yılda azalmaya devam etti ve qmail programlama ortamımın modası geçmiş birçok yönünü ümitsizce görüyorum.

Allah’tan 1990 ‘ların ortalarında açık oranım, qmail ‘in kaynak koduna oranla, yeterince düşüktü ki qmail 1.0 ‘da sadece birkaç açık vardı. Bu anlattıklarım, qmail ‘in sıra dışı güvenliğine dair bir açıklama niteliğindedir. qmail ‘in ayrıcalıkları en aza indirme mekanizmasının genele çok bir faydasının olduğu söylenemez; ayrıntılar için kısım 5.1 ‘e bakınız.

Bu kısım, programlama dillerini, program yapılarını, v.s. değiştirip meta-engineering yaparak açığı olmayan kod yazma örneklerini içerir.

### 3.1 Açık Veri Akışını Uygulamak

Global değişkenlerle ilgili standart bir tartışma vardır: “Global değişkenler genellikle programcılarını şaşırtan gizli veri akışı oluşturur.”

qmail 0.74 ‘te düzeltilen şu açığı göz önüne alın: “qmail-send, newfield\_datemake ‘i her yeni bounce için çağırır bile, eğer newfield\_datemake ‘e zaten bir ilk değer atanmışsa newfield\_datemake, newfield\_date ‘i yalnız bırakacaktır.”

Dışarıya sadece bir mesaj gönderen ve newfield\_date ‘te uygun olarak saklanan tam olarak 1 tane Date alanına ihtiyaç duyan newfield\_datemake fonksiyonunu, aslında qmail-inject programı için yazmıştım. Fakat daha sonra bu fonksiyonu pek çok mesaj gönderen (bounce eden e-postalar, gönderilemeyen mesaj raporları, v.s.) ve dışarıya giden her mesaj için bir Date alanına ihtiyaç duyan qmail-send için yeniden kullandım. newfield\_datemake ‘in mevcut newfield\_date ‘i yeniden düzenlemediğini unutmuşum (newfield\_date global değişkeni, özelliği gereği, eski Date alanıyla ilgili önceki mesajdan yeni mesaja veri transferini sonlandırdı).

Gizli veri akışı, tampon taşıma olaylarında da vardır. C ‘de  $x[i] = m$  ifadesi, ilk bakışta sadece  $x$  değişkenini değiştiriyormuş gibi görünüyor; ancak  $i$  belirlenen aralığın dışındaysa bu ifade, fonksiyon dönüş adresi, bellek ayırma yapıları da dâhil, programdaki herhangi bir değişkeni değiştirebilir. Aynı şekilde  $x[i]$  ‘yi okumak isterken programdaki herhangi bir değişken okunabilir.

Qmail ‘in tasarımı, qmail ‘in iç kısımlarındaki veri akışını izlemeyi kolaylaştırır. Örneğin; farklı UNIX süreçlerinde çok sayıda qmail çalışsın. Bu süreçler bir iş hattıyla (pipeline) birbirine bağlanmıştır. Bu iş hattı çoğunlukla dosya sistemini ve nadiren diğer iletişim tekniklerini kullanır ama asla birbirlerinin değişkenlerine doğrudan erişmez. Çünkü her süreç kendi çapında küçük bir alanda çalışır ve programcının aradaki veri akışını berbat etmesine çok az fırsat verir. Daha düşük düzeylerde ise indisler belli aralıklarda iken çalışan çeşitli dizi erişim fonksiyonları tanımladım ve kontrolü zor olan fonksiyonları kullanmaktan kaçındım.

Bugünlerde küçük ölçekli parçalara ayırmada, alt ve üst sınır kontrolünde, özet değişkenlerin güncellenmesinde (örneğin; bir dizinin değeri sıfır olan elemanlarının sayısı), v.s. daha çok ısrarlıyım. Alt ve üst sınır kontrolünden kastım, insanların normalde yaptıkları indeks numarası sınırların dışındaysa hata mesajı vermek değil; dizilere eleman eklendiğinde dizinin otomatik olarak genişlemesi ve eleman okunduğunda dizinin sıfırlanmasıdır. (Belleğin dışında mı? Kısım 4.2 ‘ye bakınız.) Benzer işleri elle yapmak aptallıktır.

### 3.2 Tam Sayıların Anlamını Sadeleştirme

Programcılarını şaşırtan bir başka durum ise  $y$  ‘nin  $y = x + 1$  işleminden sonra  $x$  ‘ten küçük olmasıdır. Bunun için  $x$  ‘in ifade edilebilen en büyük tam sayı olan  $2^{31} - 1$  olması gerekir.  $x$  ‘in bu değerinde  $y$ , ifade edilebilen en küçük tam sayı olan  $-2^{31}$  olur.

qmail 'in en yakınından geçtiği tehlike, kontrol etmeyi unuttuğum 32 bit sayaç taşıma potansiyeliydi (Georgi Guninski tarafından keşfedilmişti). Neyse ki sayacın taşıyacak kadar büyümesini bellek miktarı engelliyordu. Fakat aynı içerikteki başka bir sayaç artırma işlemi felaket gibi bir açığa neden olabilirdi.

Benzer tavsiyeler diğer tam sayı işlemleri için de geçerlidir. Programcıların kodlarla yapmak istediği işlemler matematiksel işlemlerle tamamen eş anlamlıdır ama uygulamalarında yaptıkları, matematikle pek örtüşmez. Bu durumları teşhis etmek istesem, tampon taşımaları için ek olarak çalışmam gerekir. Eğer matematiksel işlemlerimin akli başında olması için – mesela tam sayı taşımasının sadece bellek yetmezliği gibi bir durumda olması – ek bir çalışma ile genişletilmiş tam sayı kütüphanesini yazmam gerekir.

Programlama ortamlarının çoğu yazılım üretmeyi kolaylaştıracak yönde tasarlanmıştır. Aslında bunların yanlış kod yazmayı önlemeyi ön planda tutması gerekirdi. Gerçekten yapmak istemediklerimi yazmam, aslında niyetimde olanları yazmamdan çok daha fazla çalışmamı gerektirmelidir.  $2^{32}$  (veya  $2^{64}$ ) 'e göre mod alma şansım var ama bu işi gerçekleştirmek çok çalışmam gerektiğinden ötürü mutluyum.

Bazı dillerde  $a + b$  tam olarak  $a$  ile  $b$  'nin toplamı demektir. Bu diller çoğu zaman genel kullanımlarda “çok yavaş” oldukları için işten uzaklaştırılmışlardır: aşağıdaki gibi bir döngü

```
for (i = 0; i < n; ++i) c[i] = a[i] + b[i];
```

aniden  $n$  adet `gmp_add()` gibi çok pahalı yüksek doğruluklu tam sayı aritmetik fonksiyon çağrısı gerektirir. Fakat derleyici için büyük tam sayıların bulunduğu yerlerin izlerini takip eden kodları üretmek ve `gmp_add()` çağrılarını küçük tam sayıları kullanan makine seviyesine indirmek roket bilimi değildir. Muhtemelen adreslemeler yüzünden bazı yavaşlamalar olacaktır – kısım 2.6 'da belirtildiği gibi – ancak ilk olarak kodu doğru yazmalı, daha sonra programın hızı için kaygılanmalıyız.

### 3.3 Ayırıştırma (Parsing) Kaçınma

Bilgisayar dünyasında 2 çeşit komut satırı arayüzü gördüm: iyi ara yüzler ve kullanıcı ara yüzleri.

Kullanıcı ara yüzlerinin özü ayırıştırma: dağınık bir dizi komutu, katı mühendislikten çok psikolojik olarak düzgün bir biçimde bir araya getirme.

Bir başka programcı kullanıcı arayüzüyle konuşmak istediğinde arayüzün söylediklerini alıntı yapmalıdır: programcının düzenli verilerini al, yapılandırılmamış komutlara çevir ve (ümit edilir ki) ayırıştırıcı bu dağınık komut dizisini yapılandıracak.

Bu durum tam bir felaket formülüdür. Ayırıştırıcılar çoğunlukla açıkları olan yazılımlardır: belge arayüzü nedeniyle bazı giriş verilerini okumada başarısız olabilir. Programcı ile kullanıcı arayüzünün arasındaki aktarıcının da açıkları vardır: çoğu zaman sonuç olarak ürettiği verilerin anlamı doğru değildir. Sadece bazı neşeli durumlarda ayırıştırıcı ile aktarıcının ikisi de arayüzü yanlışlıkla aynı şekilde yorumlar.

Bu açıklamaları qmail 'in aşırı sade dosya yapılarında ve program seviyesindeki ara yüzlerinde ayırtmadan ve aktarmadan nasıl kaçınıldığını 2 örnekle, orijinal qmail belgelerine yazmıştım. Ancak dış kısıtların daha iyi ara yüz yazmaya izin vermediği durumlarda ayırtma ve alıntılama yaparken ortaya çıkan açıklar hakkında bir şey söylememiştim.

qmail 0.74 'te düzeltilen aşağıdaki açığı düşünelim: "qmail-inject USER 'in alıntı yapılıp yapılmayacağını kontrol etmiyordu." qmail-inject 'in aşağıda yaptığı kullanıcı adını gösteren bir From satırı oluşturmaktır:

```
From: "D. J. Bernstein" <djb@cr.yip.to>
```

Yukarıdaki örnekte djb olan kullanıcı hesap adı, normalde @ karakterinden önce harfi harfine yazılabilir ama isim parantez gibi sıra dışı karakterler içeriyorsa olduğu gibi alınmalıdır. Testlerim USER 'daki sıra dışı karakterleri özel olarak incelemiyordu. Bu nedenle harfi harfine yazmayı düzgün kodlamadan ayırt edemediler.

Başka bir örnek olarak UNIX 'in logger programındaki açığı göz önüne alın: muhtemelen sisteminiz syslog(pri, "%s", buf) yerine syslog(pri, buf) yapıyordur. Tahminim bu durum logger programında DoS 'tan öte bir güvenlik açığı oluşturmaz; çünkü saldırgan elde ettiği VM adresleri anlaşılır ASCII karakterlerine dönüştürürken korkunç derecede kötü anlar yaşayacaktır ama ayırtılmış(disassembled) nesne kodunu görmeden kesin bir şey söyleyemem. Baştan temkinli davranmak, sonradan acı çekmekten iyidir.(Better safe than sorry)

buf 'a % eklemeyen testler syslog(pri, "%s", buf) ile syslog(pri, buf) arasındaki farkı algılamayacaktır, [5] ve [19, introduction] incelenebilir.

Normal girdilerin harfi harfine kopyalanması ve anormal girdilerin alıntılama ile kopyalanması kullanıcı ara yüzlerinin evrensel bir özelliğidir. Ara yüzlerin geliştirilmediği durumlarda sonuçta oluşan açığı kapatmanın bir yolu sistematik olarak her alıntı kuralını başka bir teste çevirmektir.

### 3.4 Hatalardan Giriş Verilerine Genelleştirme

Aşağıdaki açık qmail 0.90 'da düzeltildi: "Failure to stat .qmail-owner was not an error."

Eğer Bob, ~bob/.qmail-buddies dosyasına birden fazla adres yazarsa, qmail bob-buddies e-postalarını bu adreslere gönderecektir. Ön tanımlı olarak gönderme işlemi sırasında yaşanan hatalar orijinal göndericiye iletilecektir; fakat Bob ~bob/.qmail-buddies-owner dosyasına farklı bir adres yazarsa bu durumda bilgilendirme mesajları bu adrese gönderilecektir.

qmail, ~bob/.qmail-buddies-owner dosyasının varlığını UNIX 'in `stat()` komutu ile kontrol eder. Eğer `stat()` fonksiyonu dosyanın var olduğunu söylerse qmail hataları bob-buddies-owner 'a yönlendirecektir. Eğer `stat()` fonksiyonu dosyaların var olmadığını söylerse qmail hata mesajlarını orijinal göndericiye bildirecektir.

Sorun, `stat()` fonksiyonunun geçici bir süre için başarısız olma ihtimalidir. Örneğin; `.qmail-owner` dosyası o an için erişilemeyen bir ağ üzerindeki dosya sisteminde olabilir. 0.90 sürümünden önce, qmail bu duruma dosyanın var olmadığı durumlarla aynı işlemi yapıyordu. Bu elbette yanlış bir davranış ama qmail 'in böyle bir durumda yapabileceği tek doğru hareket dosyayı bir süre sonra tekrar aramaktır.

Test araçlarım geçici olarak başarısız olma durumunu ele almıyordu. Genel hatalar için testler yaptım, hileli durumlar için testler yaptım ama ağ dosya sistemi hatalarıyla ilgili testler yapmamıştım doğrusu.

Hataları ele alan kodlar kısım 4.2 'de belirtildiği gibi ciddi şekilde azaltılabilir ancak bilgisayar sistemleri her zaman bu kodlara ihtiyaç duyar. Çok ender rastlanan hataları ele alan kodlardan nasıl kaçınabiliriz?

Sadece dosya sistemiyle değil her şeyle konuşabilen (1) bütünüyle fonksiyonel bir protokol yöneticisi yazabilirsem ve (2) `stat()` 'ı bu protokole bağlayan sade bir sarmalayıcı yazabilirsem test aşamalarım daha kolaylaşır. Protokol yöneticisine kapsamlı bir test kümesi verildiğinde işi daha da kolaylaşır ve bu açığı hemen bulur.

### **3.5 Açıklara Karşı İlerleme Sürecini Gerçekten Ölçebilir miyiz?**

Genelde kullanıcı arayüzü, özelde ise açık eleme araştırmalarında bariz bir zorluk vardır. Kullanıcıların, bu durum için programcıların, arzuladıkları etkiyi meydana getirmede başarılı olmak için mümkün olduğunca az hata yapmaları gerekir. Bu durumu nasıl modelleyebiliriz? İnsan psikolojisini biz (uzmanlar hariç) nasıl modelleyebiliriz ki? İnsanların yardımı olmadan hataları nasıl bulabiliriz?

Eğer birileri bir sınıf hataları bulan bir program yazabilirse bu çok müthiş olur – bu programı kullanıcı arayüzüne dâhil edeceğiz – ama geri kalan hataları yine bulamayacağız. Bir matematikçi olarak bu problemi biçimlendirememekten sıkıldım. Bir problemi çözebilsem de çözemesem de açıkça tanımlamayış isterim.

Neyse ki araştırmalar modeller olmadan ilerleyebiliyor. İnsanların algoritmalarını bilmeden de programlarındaki açık oranını ölçmek için onları gözlemleyebiliriz. Matematiksel olarak ispatlayamasak da yazılım mühendisliğindeki bazı araçların hataya eğilimli olduğunu bazılarının da olmadığını görebiliriz.

## 4. KODLARIN ELENMESİ

*Bugünlerde geri zekâlı yazılım yöneticileri programcının üretkenliğini, harcadığı kod satırıyla değil de kaç satır kod yazdığıyla değerlendiriyor.  
Dijkstra [9, sayfa EWD962-4]*

Bu bölüm, kod hacminin azaltılmasıyla ilgili meta-engineering örneklerinden bahseder: kod miktarını azaltmak için programlama dilini, program yapısını, v.s. değiştirme. 3. kısımda anlatıldığı üzere bu örneklerden bazıları qmail 'de kullanılmış ve açık oranının düşmesi sağlanmıştır. Diğer örneklerin de daha iyisini yapabileceği gösterilmiştir.

### 4.1 Genel Fonksiyonların Belirlenmesi

Aşağıda Sendmail 8.8.5 'in `util.c` dosyasının 1924. satırından başlayan bir kısmı görülüyor:

```
if (dup2(fdv[1], 1) < 0)
{
    syserr("%s: cannot dup2 for stdout", argv[0]);
    _exit(EX_OSERR);
}
close(fdv[1]);
```

`dup2` fonksiyonu dosya tanımlayıcısını (file descriptor) bir yerden başka bir yere kopyalar. `dup2()`/`close()` çiftinin yaptığı işi Sendmail 'de yapan birkaç fonksiyon daha var.

Yukarıdaki işi qmail 'de `fd_move()` fonksiyonu yapar ve sadece 1 yerde tanımlanıp onlarca yerde kullanılmıştır:

```
int fd_move(int to,int from)
{
    if (to == from) return 0;
    if (fd_copy(to,from) == -1) return -1;
    close(from);
    return 0;
}
```

Programcıların çoğu bu kadar küçük fonksiyonlar oluştururken pek zahmet çekmeyecektir ama `dup2()`/`close()` çiftinin yerine `fd_move()` fonksiyonunu 1 kez kullanmak programcıya birkaç satır kazandıracaktır. `dup2()`/`close()` ikilisinin onlarca yerde kullanıldığını düşünürsek `fd_move()` 'u tercih ettiğimizde onlarca satır kodu yazmaktan kurtulup zaman tasarrufunda bulunduğumuzu görebilirsiniz. (Bu fonksiyon aynı zamanda testler için doğal bir hedeftir.)

`fd_move()` sadece bir örnek; aynı fayda büyük sistemlere ve çok sayıda fonksiyona ölçeklenebilir. Çoğu durumda kodları genel kullanımları sezmek için taramak yardımcı fonksiyonlar bulmanızı sağlayabilir. Ancak; zaten var olan koddan "Bunu daha önce görmemiş miydim?" deyip yeni bir fonksiyon çıkaracak kadar ileri gitmemelisiniz.

## 4.2 Geçici Hataları Otomatik Olarak Ele Alma

qmail-local ‘den seçilmiş aşağıdaki parçaya bakın:

```
if (!stralloc_cats(&dtline, "\n")) temp_nomem();
```

stralloc\_cats fonksiyonu dinamik olarak yeniden boyutlandırılmış dtline katar (string) değişkeninin önceki içeriğine yeni katarı ve bir satır sonu karakteri ekler. Maalesef bu bitişirme işlemi belleğin dışına taşabilir. Bu durumda stralloc\_cats fonksiyonu 0 döner ve qmail-local de temp\_nomem() ile qmail sisteminin bu işlemi daha sonra tekrar denemesini sağlayacak şekilde sonlanır.

qmail ‘de binlerce koşullu dallanma var. Bunların hemen hemen yarısı – tam olarak saymayı denemedim – geçici hataları kontrol etmekten başka bir şey yapmıyor.

Çoğu durumda aşağıdaki gibi, bir işlemi gerçekleştirmeyi deneyen ve geçici hataya bağlı olarak programı sonlandıran, fonksiyonlar yazdım:

```
void outs(s)
char *s;
{
    if (substdio_puts(&ssl, s) == -1) _exit(111);
}
```

Ancak her işlem için aynı işi yapmayı tekrar etmeyi sevmedim ve sevmiyorum.

Bu testlerin hepsini daha alt seviyeli bellek ayırma, disk okuma, v.s. ve programdan geçici bir hataya bağlı olarak çıkma gibi daha az sayıda fonksiyonlarla gerçekleyebilirdim. Ama bu strateji qmail-send gibi uzun süre çalışan programlar için uygun değildir. Bu programlar sistem yöneticisi izin vermediği müddetçe kendilerini sonlandıramazlar!

Şunu fark ettim ki çoğu kütüphane ve dil aynı stratejiyi kullanıyor ve bu da onları uzun süreli programlarda kullanılamaz kılıyor. Belki de özel amaçlı programlar özel yazılım mühendisliği araçlarını kullanmalı ama uzun süre çalışan programları özel amaçlı olarak görmüyorum ve yazılım mühendisliği mantığını uzun süreli programlar için uygun olmadıklarından ötürü sorguluyorum.

Neyse ki programlama dilleri hataları ele alırken güçlü istisna durum değerlendirmesi, otomatik hata raporları ve alt programlardan net bir şekilde çıkış gibi özelliklere sahip. Programlama dillerinde aşağıdaki ifadeyi hata kontrolü için çok fazla çaba sarf etmeden yazabilmeliyim:

```
stralloc_cats(&dtline, "\n")
```

Veya sadece aşağıdaki ifadeyi:

```
dtline += "\n"
```

Kod miktarındaki azalma açık oranını azaltacaktır. Örneğin; qmail 0.92 ‘de düzeltilen şu hatanın meydana gelme şansı olmazdı: “ipme\_init() ‘in -1 döndürmesine rağmen qmail-remote çalışmaya devam ediyor”.

qmail 'i yazarken birçok dili son kullanıcı için derleme ve kullanmada C 'den çok daha zor olduğu için reddettim. Kodu daha iyi bir dilde yazdıktan sonra bir çevirici program kullanarak dağıtım dili olarak seçtiğim C 'ye dönüştürmemem anlaşılmaz bir körlüktü. Stroustrup 'un cfront 'unun, C++ 'dan C 'ye derleyici, her ne kadar şu ana dek istisnaları ele alma desteği olduğunu duymasam da ilham verici bir örnek olduğunu söyleyebilirim.

### 4.3 Ağ Araçlarını Tekrar Kullanmak

UNIX 'in ağ bağlantılarını dinleyen genel amaçlı `inetd` aracı mevcuttur. Bir bağlantı kurulduğunda `inetd`, bu bağlantıyı idare edecek başka bir program çalıştırır. Örneğin; `inetd` gelen SMTP bağlantılarını yönetmesi için `qmail-smtpd` 'yi çalıştırabilir. `qmail-smtpd` programının ağ, çoklu görev bölüşümü gibi konularla ilgilenmesine gerek yok. Sadece standart girişindeki bir istemciden SMTP komutlarını alır ve yanıtları standart çıktısına gönderir.

Sendmail ağ bağlantılarını dinlemek için kendi kodunu kullanır. Bu kod `inetd` 'den daha karmaşıktır; çünkü sistemin yük ortalamasını takip eder ve CPU 'yu almak için yarış olduğu sıralarda servisleri azaltır.

Sendmail neden CPU yoğun iken e-posta almayı istemiyor? Temel problem şu: Sendmail e-posta alır almaz e-postanın nereye gitmesi gerektiğini çözmek için çok fazla gayret sarf ediyor ve e-postayı göndermeyi deniyor. Eğer Sendmail 'e göndermesi için aynı anda çok sayıda e-posta verirseniz Sendmail hemen e-postaların tamamını göndermeye çalışacaktır ve neticede bellek taşma hatası alıp çoğunu göndermede başarısız olacaktır.

İşte bu nedenden ötürü Sendmail CPU 'nun yük durumunu kontrol eder. Eğer CPU yoğun ise e-postaları henüz gönderilmeyenler kuyruğuna atar ve periyodik olarak kuyrukte bekleyen sıradaki e-postayı göndermeyi dener. Bu mekanizma bilgisayara kesinlikle çok fazla yük bindirmez; çünkü aynı anda ne kadar mesaj gelirse gelsin, Sendmail bu e-postalardan kuyruğu tekrar taramak istediğinde haberdar olacaktır ve bu sayede yüksek paralellikte çalışmaya devam edecektir.

Sendmail, normalde kuyruğu 15 dakika ile 1 saat arasında değişen sürelerde çalıştırır. RFC 1123 'ün 5.3.3.1 kısmı ise bu sürenin en az 30 dakika olmasını tavsiye eder. Çok kısa zaman aralığı, örneğin 30 saniye, atayan sistem yöneticileri Sendmail 'in kuyruktaki her e-postayı günde yüzlerce kez göndermeye çalıştığını görür.

Sendmail neden bir arka plan mekanizmasına ihtiyaç duyar? Neden gelen tüm e-postaları bir kuyruğa atmıyor? Yanıtı şu: kuyruğa atılan her e-posta – CPU meşgul değilse bile – hemen gönderilmez. E-postaları kuyruğa atmak mesajların gönderilmesi için kuyruğun yeniden çalıştırılması beklemek demektir. Bu nedenle kullanıcılar e-postaların gecikmesinden şikâyetçi olacaktır.

qmail 'de kuyruğa bir e-posta atıldığında çok az miktarda ilave kod, `qmail-send`, ile arka plandaki e-posta gönderme mekanizması fark edilir. `qmail-send` programı – meşgul değilse – anlık olarak gelen e-postayı göndermeyi dener ve gönderemezse mantıklı bir çizelgeye göre belli aralıklarla e-postayı tekrar göndermeyi dener. Sonuç olarak arka plan e-posta gönderme mekanizması yok olur; yani qmail 'de arka planda e-posta göndermek şeklinde bir mekanizma yoktur. Ayrıca sistem yükünü kontrol etmek de yoktur; qmail, `inetd` tarafından çalıştırılmaktan dolayı mutludur.

Eğer sistemin yük dengesini kontrol etmek istersem bunu `inetd` gibi genel amaçlı bir araçla yaparım; aynı kodu her uygulama için tekrar yazmam.

#### 4.4 Erişim Kontrollerini Tekrar Kullanma

UNIX kullanmaya 20 yıl önce Ultrix ile başladım. `/tmp` dizinindeki bir dosyayı çalıştırması için `.forward` dosyasını ayarladığımı hatırlıyorum. Çıktı olarak üretilen binlerce dosyayı denetlediğimi ve Sendmail 'in ara sıra benimkinden farklı bir kullanıcı kimlik numarasıyla program çalıştırmasını hayretle izlediğimi hatırlıyorum.

Sendmail, kullanıcıların `.forward` dosyasını şimdi anlatacağım şekilde yönetir. Öncelikle kullanıcının `.forward` dosyasını okuma izninin olup olmadığı kontrol edilir – `.forward` belki de kullanıcının okuma hakkının olmadığı gizli bir dosyaya sembolik bağlantıdır. Daha sonra bu dosyada belirtildiği şekilde gönderme komutlarını ayarlar ve programı çalıştıran kullanıcıyı, komutlardan kimin sorumlu olduğunu not eder (muhtemelen daha sonra işlenmek üzere oluşturulmuş bir kuyruk dosyası). Tüm bunları yapmak için ciddi miktarda kod kullanılıyor (örneğin; `safefile.c` ve çeşitli dağınık kod parçaları) ve bu kodlarda tam olarak az miktarda açık var.

Elbette işletim sistemi kullanıcının bir dosyaya erişim izni olup olmadığını kontrol eden ve hareketlerini takip eden kodları var. Aynı kodu tekrar yazmak, neden?

qmail bir kullanıcıya e-posta göndermek istediğinde sadece `qmail-local` programını doğru kullanıcı kimlik numarasıyla çalıştırır. `qmail-local` e-postayı göndermek için gerekli komutları çalıştırmak istediğinde işletim sistemi zaten o kullanıcının erişim haklarını kontrol eder. `qmail-local` kullanıcı tarafından belirtilen bir programı çalıştırmak istediğinde, işletim sistemi otomatik olarak bu programa doğru kullanıcı kimlik numarasını atayacaktır.

Bu kodu tekrar kullanmak için CPU zamanında küçük bir ücret ödedim: qmail her gönderme işlemi için yeni bir süreç başlatır. Fakat Sendmail 'in erişim izinlerini kontrol etmek için kullandığı, fazlalık olan, hiçbir sistem çağrısını kullanmadım. Zaten birileri gerçek dünyada qmail 'in e-posta gönderirken yaptığı `fork()` çağrılarında dar boğaza girmiş bir bilgisayar gösterene kadar, bu süreyi azaltmak için fazladan kod yazmayı kesinlikle düşünmüyorum.

#### 4.5 Dosya Sistemini Yeniden Kullanmak

National Security Agency 'nin SMTP sunucusunun `efd-friends@nsa.gov` e-posta listesi için e-posta kabul ettiğini varsayalım. MTA bu e-postanın `nsa.gov` tarafından kabul edileceğini nasıl anlar? MTA `efd-friends` için e-posta gönderim komutlarını nasıl öğrenir?

Açıkçası MTA `nsa.gov` ve `efd-friends` isimlerini bir veritabanında arar. Veritabanı demek doğru olmayabilir; bu işlem için belki de bir çağrışimli dizi veya başka bir şey kullanılıyordur. İsmi her ne olursa olsun bu veritabanı, sistem yöneticisi ve e-posta liste yöneticisi tarafından `efd-friends` ismi sorgulandığında geriye bu isme ait bilgileri dönebilmeye yetisine sahiptir.

Sendmail 'de ise `nsa.gov` ve çeşitli diğer yapılandırmalar daha az karışık olan bir “yapılandırma dosyasında” tutulur. `efd-friends` gibi e-posta liste isimleri “takma isimler dosyasında” tutulur. `nsa.gov` ve `efd-friends` gibi isimleri bu dosyalarda aramak ciddi miktarda ayırıştırma yapmayı gerektiriyor.

Elbette ki işletim sistemi bir isim altına yığılmış bilgileri saklayan ve daha sonra bu bilgileri geri döndüren koda sahip. Bu yığınlar “dosya”dır, isimleri “dosya ismi”dir ve koduna da “dosya sistemi kodu” denir. Aynı kodu tekrar yazmak, neden?

qmail ‘de `efd-friends` için e-posta gönderme komutları `.qmail-efd-friends` dosyasının içeriğidir. Bu komutları bulmak veya değiştirmek bir dosyayı açmak kadar basit bir iştir. Kullanıcılar zaten dosya oluşturma ve düzenleme araçlarına sahipken qmail ‘in bu yazılımları tekrar keşfetmesine gerek yok.

`nsa.gov` ile ilgili yapılandırmaları `/var/qmail/control/domains/nsa.gov` içerisine aynı sadelikteki kodla yazsam iyi olurdu ama biraz karışık işler yaptım: `nsa.gov` bir dizindeki dosyadan ziyade bir dosya satırıdır. Verimli ve hızlı çalışmak adına endişelenmişim: çoğu UNIX dosya sistemi lineer zamanlı tecrübesiz algoritmalar kullanıyordu ve ben qmail ‘in binlerce alan adı tutan bilgisayarlarda yavaş çalışmasını istemiyordum. UNIX dosya sistemi, aynı zamanda, küçük bir dosyayı kaydetmek için kilobyte ile ölçekli bir şeyler tüketiyordu.

Geçmişe baktığımda ayrıştırma için yazdığım kodlar – sadece dosya ayrıştırmak için olanlar değil, e-postaları dizinlere dağıtmak için yazdıklarım da dâhil – benim aptallığımdır. Çünkü dar boğaz olarak ölçmediğim, varsayımdan ibaret performans problemleriyle ilgilenmem hataydı. Ayrıca, ölçümler gösterdiğince, dosya sistemini kullanan her programdan ziyade dosya sistemini kilitleyip problemi kaynağında adreslemeliydim (er ya da geç yoğun sitelerde yaptıkları gibi).

## 5. GÜVENİLEN KODLARIN AZALTILMASI

### 5.1 TCB ‘nin Doğru Ölçülmesi

qmail belgelerinde “Tüm programlar birbiriyle uzlaşmış olduğundan davetsiz bir misafir `qmaild`, `qmails` veya `qmailr` hesaplarının ve e-posta kuyruğunun kontrolünü eline geçirirse bile sisteminizin kontrolünü eline alamaz.” demişim. “Diğer hiçbir program bu beşinin döndürdüğü sonuca güvenmemektedir” tarzında konuşmaya devam edip ayrıcalıkların kısıtlanması ve güvenlik hakkında faydalı konulardan bahsetmişim.

Bu konu hakkında bir süre düşünelim. `qmail-remote` ‘da bir saldırganın `qmailr` hesabının kontrolünü eline geçirmesine izin veren bir açık olduğunu kabul edelim. Bu durumda saldırgan, ağ bağlantıları güçlü kriptografi ile korunmuş olsa bile sistemin giden e-postalarını çalabilir veya değiştirebilir. Bu bir güvenlik felaketi ve onarılması gerekiyor. qmail ‘in bu açıktan korunmak için yaptığı tek şey açıklardan sakınmaktır.

Benzer şekilde; qmail ‘in, sahibi `root` olan dosyaları etkileme yeteneği olan kodlarına dikkat çekmemeliydim. qmail ‘in neredeyse tüm kodu normal kullanıcıların dosyalarını – diskte olsun e-posta sisteminde olsun – etkileme yeteneğine sahip. Bu nedenle qmail ‘in tüm kodu kullanıcıların güvenlik gereksinimlerini istismar edecek konumdadır. Bu güvenlik tehlikesinden kaçınmanın tek yolu açık vermemektir.

Kelime işlemci veya müzik oynatıcı yazılımların programcıları genellikle çok fazla güvenlik endişesi taşımaz. Fakat kullanıcılar bu programların e-posta ile gelen veya internetten indirdikleri dosyaları da işlemlerini bekler. Bu dosyaların bazıları kötü niyetli insanlar tarafından hazırlanmıştır ve bu dosyaları yöneten programların çoğu da saldırganların kötüye kullanabileceği açıklara sahiptir.

2004 'te "UNIX Güvenlik Açıkları" dersini verirken öğrencilerden ödev olarak yeni güvenlik açıkları bulmalarını istemiştim. Öğrencilerim 44 güvenlik açığı buldu ve bunların çoğu da – yanlışlıkla – sistemin güvenilen kod tabanının dışında görülen programlardaydı.

Örneğin; Ariel Berkman `xine-lib` 'de, film oynatma kütüphanesi, bir tampon taşırmasıyla ilgili bir açık buldu. Kullanıcılar internetten indirdikleri filmleri oynattıklarında risk altına giriyor: saldırgan, film oynarken kullanıcının dosyalarını okuyabilir ve çalıştırdığı programları takip edebilir, v.s.

Güvenli işletim sistemleri ve sanal makineler güvenliklerini küçük tabanlı güvenilen kodların sağladığını söylüyor. Maalesef, olaya yakından baktığımızda bir programın birden fazla kaynak dosyasından gelen verileri idare etmeye başladığında, bahsedilen güvenliğin anlamsız hale geldiğini görüyoruz. Örneğin; işletim sistemi, e-posta okuyan bir program diğer e-postaları değiştirmeye veya çalmaya kalktığında veya bir internet tarayıcısı web sayfası çalmaya veya gelen web sayfaları değiştirmeye kalktığında bu programı durdurmak için hiçbir şey yapmıyor. E-posta okuyan ve web tarayan kodlar gerçekte aynı güvenilen kodları kullanır: bu koddaki açıklar kullanıcının güvenliğini bozar.

## 5.2 Tekil Kaynak Değişimlerini Yalıtma

`jpegtopnm` programı bir JPEG dosyasını, sıkıştırılmış resim, giriş verisi olarak okur. Sıkıştırılmış resmi açar, çıktı olarak bir bitmap üretir ve sonlanır. Şu ana kadar bu program güvenilir: açıkları güvenlikle uzlaşabilir. Şimdi bunu nasıl düzeltebileceğimizi görelim.

`jpegtopnm` programının son derece sıkı denetlenen bir alanda sadece standart girişten JPEG dosyası okumasına, sonucu standart çıkışa yazmasına ve kısıtlı bir belleği kullanmasına izin verildiğini varsayalım. Mevcut UNIX araçları `root` 'un bu alanda şunları yapmasına izin verir:

- `RLIMIT_NOFILE` sınırını 0 yaparak yeni dosya, yeni soket, v.s. oluşturulmasını yasaklar.
- Dosya sistemi erişimini yasaklar: boş bir dizine `chdir` ve `chroot`.
- Bir sürecin kimlik numarasına bir kullanıcı kimlik numarası sabitleyebilir. Bunu yapmak, diğer sistem yönetim araçları aynı kullanıcı kimlik numarası aralığından uzak durduğu müddetçe, taban kimlik numarasına sürecin kimlik numarasını eklemek kadar rahat yapılabilir.
- Bu kullanıcı adıyla hiçbir şeyin çalışmamasını sağlayın: `setuid(targetuid)`, `kill(-1, SIGKILL)` ve `_exit(0)` komutlarını çalıştıracak bir çocuk süreç oluşturduktan sonra çocuk sürecin normal olarak sonlanıp sonlanmadığını kontrol edin.
- Hedef kullanıcı kimlik numarasının kullanıcı ve grup kimlik numarasını düzenleyerek `kill()`, `ptrace()`, v.s. komutların çalıştırılmasını yasaklayın.
- Mevcut ve maksimum `RLIMIT_NPROC` değerini 0 yaparak `fork()` çağrısını yasaklayın.
- Bellek ve diğer kaynaklara istediğiniz sınırları koyun.
- Programın geri kalanını çalıştırın.

Bu aşamada işletim sisteminde çok büyük açıklar yoksa `jpegtopnm` programının ilk dosya tanımlayıcısından başka bir iletişim kanalı yoktur.

Saldırganın sahte bir JPEG dosyası ile `jpegtopnm` programını istismar edip programın tüm kontrolünü eline geçirdiğini varsayalım. Saldırgan şu an istediği bitmap çıktıyı üretebilir – bu zaten hiçbir açık olmadığı da yapabildiği eylemdir – ama bundan başka hiçbir şey yapamaz. Bu noktada `jpegtopnm` programı, güvenilen kod değildir ve artık bu programdaki açıklar kullanıcının güvenliğini bozma yeteneğinden çok uzaktır.

Kısım 2.3 ‘te olduğu gibi, kullanıcının güvenlik gereksinimlerinin bir JPEG dosyayı etkileyen kişinin, çıktı olarak üretilen bitmap ‘i kontrol etmesine izin verdiğini ima ediyorum. Bu güvenlik gereksinimleri üzerine getirilen bir kısıtlama; ancak mantıklı olduğunu düşünüyorum: şimdiye kadar hiç kimsenin bazı JPEG parçalarının birbirlerinden korunmasını istediğini duymadım. Daha önemlisi, güvenlik gereksinimlerindeki farkların yalıtılmış dizindeki kod parçaları tarafından barındırılmayacağına inanmak için bir neden göremiyorum.

Ayrıca, CPU ‘nun gizli bilgileri diğer süreçlere sızdırmadığını varsayıyorum. Bu varsayım tartışılabilir. Örneğin; kısım 2.3 ‘teki yorumlayıcının CPU komutlarına erişimi sınırlı olmalıdır. Performans için kısım 2.6 ‘ya bakınız.

Önerilerimin ardından Ariel Berkman, UNIX ‘in standart `xloadimage` resim görüntüleme aracının tamamını `jpegtopnm` gibi bir dizi modüler filtreler ayırma işini yeniden tasarladı. Her filtre, yukarıda anlatılan tekniklerle hapsedilip güvenilen kod miktarı çok az indirildi.

### 5.3 Çok Kaynaklı Birleşmeleri Geciktirmek

Kullanıcının e-posta kutusundaki mesajları görüntüleyen bir e-posta okuma programı düşünün. Tek kaynaklı dönüşümler – örneğin; eklenti olarak gelen JPEG dosyasını açma – kısım 5.2 ‘de anlatılan tekniklerle hapsedilebilir. Fakat birden fazla kaynağı kullanan dönüşümlerde durum nedir?

E-postaların konularının liste halinde görüntülenmesi çok sayıda kaynaktan gelen verilerin birleştirilmesiyle elde edilir. Her e-posta, listede kendi kaydını kontrol etme hakkına sahiptir ama diğer e-postaları etkileme hakkı yoktur. Her ne kadar liste ilk yerde doğru oluşturulmuş olsa bile, sonraki alt dönüşümler listedeki e-postalar arasındaki ayrımı çığneyebilir.

Her e-postanın konusu farklı bir alanda saklanıp, birbirlerinden bağımsız olarak dönüştürülüp daha sonra birleştirildiğinde durum çok farklı olur. Bu durumda dönüşümler tek kaynaklıdır. Birleşmeyi geciktirme, güvenilen kod miktarını azaltır.

Örneğin; çapraz kodlamayı durdurmak, şu an çok sayıda kaynaktan gelen verileri bir dosyada birleştiren ve web sayfasına dönüşüme yardım eden kodun her parçasına özellikle dikkat etmek anlamındadır. Eğer farklı kaynaklardan gelen veriler ayrı ayrı yerlerde tutulur ve son anda web tarayıcısı tarafından birleştirilirse sadece son birleştirme adımı dikkat gerektirir.

### 5.4 Küçük Bir TCB ‘ye Gerçekten İhtiyacımız Var mı?

qmail 'in kodunun tamamını güvenilmeyen hapsilere koymayı başaramadım. Kodun herhangi bir yerindeki açık güvenlik tehlikesi arz ediyor. Bu başarısızlığa rağmen qmail 'in hayatta kalmasının sebebi, kısım 3 ve 4 'te anlatıldığı gibi, açık sayısının çok az olmasıdır.

Kod hacminin küçültülmesi ve açık oranının azaltılması çok büyük sistemlerin de aynı sebeple ayakta kalmasını muhtemelen sağlayacaktır. Yarım milyon satır koddan oluşmasına rağmen hiç açığının bulunmadığına inanılan sistemler duydum. Dizüstü bilgisayarında daha çok miktarda kod var – güvenliğini riske atan çok fazla miktarda kod – ama dünyanın bu kodlardaki tüm açıkları kapatabileceği akla yatabilen bir gerçektir.

Ancak kısım 5.2, koddaki açıkları elemeye oranla, büyük miktardaki kodun TCB 'den daha düşük maliyet – kesinlikle çok düşük – ve daha yüksek bir güvenilirlikle elenebileceğini gösterir. Bu örneğin ölçeklenebileceğini konusunda iyimserim. TCB 'nin en az ne kadar küçük olabileceğini bilmiyorum ama öğrenmeyi dört gözle bekliyorum. Elbette geriye kalan koddaki açıkları kapatmaya hala ihtiyacımız var!

## 6. KAYNAKLAR

[1] Anurag Acharya, Mandar Raje, MAPbox: using parameterized behavior classes to confine untrusted applications, 9th USENIX Security Symposium (2000). URL: <http://www.usenix.org/publications/library/proceedings/sec2000/acharya.html>. Alıntılar: §2.5.

[2] Vinod Anupam, Alain Mayer, Security of web browser scripting languages: vulnerabilities, attacks, and remedies, 7th USENIX Security Symposium (1998). URL: <http://www.usenix.org/publications/library/proceedings/sec98/anupam.html>. Alıntılar: §2.5.

[3] M. Lee Badger, Daniel F. Sterne, David L. Sherman, Kenneth M. Walker, Sheila A. Haghighat, A domain and type enforcement UNIX prototype, 5th USENIX Security Symposium (1995). URL: <http://www.usenix.org/events/security95/badger.html>. Alıntılar: §2.5.

[4] Daniel J. Bernstein, Internet host SMTP server survey, posting to comp.security.unix, comp.mail.misc, comp.mail.sendmail (1996). URL: <http://cr.yp.to/surveys/smtpsoftware.txt>. Alıntılar: §1.1.

[5] Daniel J. Bernstein, Re: Logging question (1996). URL: <http://www.ornl.gov/lists/mailling-lists/qmail/1996/12/msg00314.html>. Alıntılar: §3.3.

[6] Daniel J. Bernstein, Curve25519: new Diffie-Hellman speed records, in [24] (2006), 207-228. URL: [cr.yp.to/papers.html#curve25519](http://cr.yp.to/papers.html#curve25519). Alıntılar: §2.6.

[7] Richard Blum, Running qmail, Sams Publishing, 2000. ISBN 978-0672319457. Alıntılar: §1.2.

[8] Computer Emergency Response Team, CERT advisory CA-2002-19: buffer overflows in multiple DNS resolver libraries (2002). URL: <http://www.cert.org/advisories/CA-2002-19.html>. Alıntılar: §2.5.

[9] Edsger W. Dijkstra, Introducing a course on mathematical methodology, EWD962 (1986). URL: <http://www.cs.utexas.edu/users/EWD/ewd09xx/EWD962.PDF>. Alıntılar: §4.

[10] Jeff Gennari, Vulnerability Note VU#834865: Sendmail contains a race condition (2006). URL: <http://www.kb.cert.org/vuls/id/834865>. Alıntılar: §1.1.

[11] Ian Goldberg, David Wagner, Randi Thomas, Eric Brewer, A secure environment for untrusted helper applications (confining the wily hacker), 6th USENIX Security Symposium (1996). URL: <http://www.usenix.org/publications/library/proceedings/sec96/goldberg.html>. Alıntılar: §2.5, §2.5, §2.5, §2.5.

[12] Internet Security Systems, Remote Sendmail header processing vulnerability (2003). URL: <http://www.iss.net/issEn/delivery/xforce/alertdetail.jsp?oid=21950>. Alıntılar: §2.3.

- [13] Donald Knuth, Structured programming with go to statements, *Computing Surveys* 6 (1974), 261–301. Alıntılar: §2.6.
- [14] John R. Levine, *qmail*, O’Reilly, 2004. ISBN 978–1565926288. Alıntılar: §1.2.
- [15] Nimisha V. Mehta, Karen R. Sollins, Expanding and extending the security features of Java, 7th USENIX Security Symposium (1998). URL: <http://www.usenix.org/publications/library/proceedings/sec98/mehta.html>. Alıntılar: §2.5.
- [16] David S. Peterson, Matt Bishop, Raju Pandey, A flexible containment mechanism for executing untrusted code, 11th USENIX Security Symposium (2002). URL: <http://www.usenix.org/events/sec02/peterson.html>. Alıntılar: §2.5.
- [17] Eric Raymond, *The cathedral and the bazaar* (1997). URL: <http://www.catb.org/~esr/writings/cathedral-bazaar/cathedral-bazaar/>. Alıntılar: §2.1.
- [18] Jerry H. Saltzer, Mike D. Schroeder, The protection of information in computer systems, *Proceedings of the IEEE* 63 (1975), 1278–1308. Alıntılar: §2.5.
- [19] Umesh Shankar, Kunal Talwar, Jeffrey S. Foster, David Wagner, Detecting format string vulnerabilities with type qualifiers, 10th USENIX Security Symposium (2001). URL: <http://www.usenix.org/publications/library/proceedings/sec01/shankar.html>. Alıntılar: §3.3.
- [20] Dave Sill, *The qmail handbook*, Apress, 2002. ISBN 978–1893115408. Alıntılar: §1.2.
- [21] Kenneth M. Walker, Daniel F. Sterne, M. Lee Badger, Michael J. Petkac, David L. Sherman, Karen A. Oostendorp, Confining root programs with domain and type enforcement, 6th USENIX Security Symposium (1996). URL: <http://www.usenix.org/events/sec96/walker.html>. Alıntılar: §2.5.
- [22] Kyle Wheeler, *Qmail quickstarter: install, set up and run your own email server*, Packt Publishing, 2007. ISBN 978–1847191151. Alıntılar: §1.2.
- [23] Chris Wright, Crispin Cowan, Stephen Smalley, James Morris, Greg Kroah-Hartman, Linux security modules: general security support for the Linux kernel, 11th USENIX Security Symposium (2002). URL: <http://www.usenix.org/events/sec02/wright.html>. Alıntılar: §2.5.
- [24] Moti Yung, Yevgeniy Dodis, Aggelos Kiayias, Tal Malkin (editors), 9th international conference on theory and practice in public-key cryptography, New York, NY, USA, April 24–26, 2006, proceedings, *Lecture Notes in Computer Science*, 3958, Springer, Berlin, 2006. ISBN 978–3–540–33851–2. [6] ‘ya bakınız.