

Moving Target Defense

En kartläggning av forskningsbidrag

HANNES HOLM, JOHAN BENGTTSSON,
JACOB LÖFVENBERG, MATS PERSSON, TEODOR SOMMESTAD



Hannes Holm, Johan Bengtsson, Jacob
Löfvenberg, Mats Persson, Teodor Sommestad

Moving Target Defence

En kartläggning av forskningsbidrag

Bild/Cover: Jesper Holm

Titel	Moving Target Defense – En kartläggning av forskningsbidrag
Title	Moving Target Defense – A survey of research contributions
Rapportnr/Report no	FOI-R--3942--SE
Månad/Month	September
Utgivningsår/Year	2014
Antal sidor/Pages	48
ISSN	1650-1942
Kund/Customer	Internt
Forskningsområde	4. Informationssäkerhet och kommunikation
FoT-område	
Projektnr/Project no	I354152
Godkänd av/Approved by	Christian Jönsson
Ansvarig avdelning	Informations- och aerosystem

Detta verk är skyddat enligt lagen (1960:729) om upphovsrätt till litterära och konstnärliga verk. All form av kopiering, översättning eller bearbetning utan medgivande är förbjuden

This work is protected under the Act on Copyright in Literary and Artistic Works (SFS 1960:729). Any form of reproduction, translation or modification without permission is prohibited.

Sammanfattning

Historien antyder att det i de allra flesta system kan finnas IT-säkerhetsbrister, eller sårbarheter, som exponeras mot antagonister. Moving Target Defense (MTD) är ett samlingsnamn för tekniker som försöker uppnå säkra system genom att ständigt förändra var sårbarheter befinner sig istället för att motarbeta deras existens. Säkerheten för det skyddade systemet ges i och med att attackeraren inte hinner identifiera var sårbarheter för en konfiguration finns innan denna konfiguration byts ut.

Begreppet MTD är relativt nytt men den bakomliggande idén har använts inom IT-området i åtminstone ett decennium. Exempelvis har en variant av MTD funnits för Linux sedan 2001 och en annan variant studerades samma år av DARPA. Den förhållandevis intensiva forskning som idag bedrivs inom området startade dock för ett par år sedan när bland andra Department of Homeland Security riktade dedikerade forskningsmedel till det.

Denna studie kartlägger artiklar som presenterar lösningar baserade på MTD och syftar till att identifiera vilka varianter av MTD som finns, hur användbara dessa varianter är, vilka prestandaegenskaper de har och om de ger mer säkerhet. En litteratursökning identifierade 129 artiklar; av dessa artiklar var 32 relevanta för studiens syfte.

Sex områden för MTD identifierades: 1) rörlig kodtransformering, 2) rörlig minnesallokering, 3) rörliga applikationer, 4) rörliga maskiner, 5) rörliga nätadresser och 6) kombinationer av dessa fem områden. Fokus för de identifierade artiklarna är på introduktion och presentation av skydd snarare än utvärdering av deras prestanda eller hur de erbjuder ökad säkerhet. Sådana egenskaper diskuteras ofta med teori och/eller simuleringsresultat som stöd, men i de allra flesta fall utan empiri och under orealistiska förutsättningar. Detta gör att det är svårt att veta hur effektiva skydden faktiskt är och hur mycket de påverkar användarens upplevelse. Framtida arbete inom området bör fokusera mer på att utvärdera kvaliteten för förespråkade skydd, i synnerhet angående hur användaren påverkas och vilken hotmodell som är realistisk. Empiriska tester under realistiska förutsättningar, exempelvis med hjälp av så kallade cyber ranges, är ett sätt att uppnå mer valida resultat.

Nyckelord: IT-säkerhet, Moving Target Defense, Dynamiska försvar

Summary

Time has shown that most systems can have cyber security vulnerabilities that are exposed to attackers. Moving Target Defense (MTD) is a type of technique that attempts to obtain secure systems given this premise by continuously changing the attackable surface; thus changing between different (presumed vulnerable) configurations rather than attempting to mitigate existing vulnerabilities. System security is assured as the attacker does not have the time to identify where vulnerabilities are located before the current configuration is replaced by a new one.

MTD is a relatively new concept, but its core ideas have been available in the IT-community for more than a decade. For example, a variant of MTD has been available for Linux since 2001 and another variant was tested by DARPA during the same year. The relatively intensive research that currently is being carried out regarding MTD was however initiated a few years ago when large investments were made by e.g. the Department of Homeland Security for dedicated research within the area.

This study surveys articles that concern MTD with the purpose to identify what variants of MTDs that exist, how useful these are, what performance they possess, and what kind of security they bring. A literature study identified 129 articles; of these, 32 were relevant to the purpose of the study.

Six kinds of MTDs were identified: 1) moving code transformation, 2) moving memory allocation, 3) moving applications, 4) moving machines, 5) moving network addresses and 6) combinations of these five kinds. Focus for the identified articles lie in introduction of novel protection mechanisms rather than examining the quality of these mechanisms. Quality is often discussed using theory and/or simulation results, but in general without actual empirical data and given unrealistic premises. This makes it difficult to know how (from a security standpoint) effective the defense mechanisms are, and how much they affect the experience of the end-user. Future work within the area should thus focus more on evaluating the quality of defense mechanisms, in particular in regard to how the end-user is affected and given a realistic threat model. Empirical tests given realistic configurations, for instance using cyber ranges, is one mean of obtaining more valid results.

Keywords: IT security, Moving Target Defense, Dynamic Defenses

Innehållsförteckning

1	Introduktion	7
1.1	Syfte och avgränsningar	8
1.2	Litteraturstudie.....	8
1.3	Disposition.....	10
2	Resultat	11
2.1	Rörlig kodtransformering	14
2.2	Rörlig minnesallokering	16
2.3	Rörliga applikationer	21
2.4	Rörliga maskiner.....	22
2.5	Rörliga nätadresser	24
2.6	Kombinerad rörelse	28
3	Diskussion	30
3.1	Sammanfattning av skyddens värde	30
3.2	Framtida arbete	31
4	Referenser	34
	 Bilaga A. Buffer overrun	 39
	 Bilaga B. Kartläggningsramverk	 41
	 Bilaga C. Statistik från kartläggning	 42

1 Introduktion

Moving Target Defense (MTD) är av diverse aktörer, såsom US Federal Networking and Information Technology Research and Development Program (NITRD), en teknik som kan komma att totalt förändra spelplanen för hur tekniska system attackeras och försvaras [2].

MTD handlar om att förändra tekniska system över tid för att därmed försvåra möjligheten för lyckade attacker. Exempel på MTD innefattar dynamisk förändring av IP-adresser i nätverk för att försvåra spaningsarbetet för en attackerare och rörliga minnesadresser för mjukvaror för att försvåra injektion av skadlig kod i dem.

MTD kan liknas vid arkadspelet wack-a-mole, som pryder rapportens framsida. I detta spel ges poäng varje gång spelaren får in en ren träff på det objekt som dyker upp ur hålen. Svårigheten i detta spel ligger inte i att slå decimeterstora objekt med en klubba. Svårigheten är att hinna se, reagera och träffa objekten med rena träffar innan de försvinner ner i hålen igen. På samma sätt som den ständiga förändringen på wack-a-mole-bordet kan försvåra något så enkelt som att slå med en klubba så försvårar rörelsen skapad av ett MTD kartläggningsarbetet och/eller angrepp mot de tekniska system som det förändrar.

Tankegångarna kring MTD inom IT-området har funnits i över ett decennium, t.ex. så testade DARPA effektiviteten för en viss typ av rörliga IP-adresser redan 2001 [41]. Samma år introducerades även en implementation av rörliga minnesadresser i och med den så kallade Adress Space Layout Randomization (ASLR)-tekniken [4], vilken idag stöds av i princip varje operativsystem och de allra flesta mjukvaror. Så varför kallas MTD för en game changer när konceptet har funnits och testats i åtminstone 13 år? Vi ser två möjliga orsaker till detta:

1. **De stora svårigheterna att utveckla buggfri mjukvara, samt att installera och nyttja mjukvara utan att konfigurationsfel uppstår.** Idag finns det en betydligt större kunskap om förekomsten av tekniska sårbarheter än vad som fanns 2001. Enligt US National Vulnerability Database (NVD) [7] fanns det 2598 sårbarheter identifierade i januari 2001; i augusti 2014 var antalet sårbarheter uppe i 63 899. En framtid som involverar komplett frånvaro av sårbarheter är högst osannolik och med ett fungerande MTD innebär närvaro av sårbarheter inte att dessa enkelt kan utnyttjas. Detta är en attraktiv lösning på ett svårt problem.
2. **Det finns idag kraftfulla teknologier för förändring av viktiga attacktyper.** Virtualisering är ett bra exempel på en teknologi som idag kan nyttjas för att på ett enkelt sätt byta ut komplexa tekniska konfigurationer. Exempelvis kan hela den virtuella maskinen, eller individuella mjukvaror som finns på den, dynamiskt bytas ut mot andra som uppfyller samma typ av funktion, såsom en Apache-webserver mot

Nginx-webserver. Ett annat exempel är IPv6 som möjliggör en mycket större adressrymd för randomisering av IP-adresser än IPv4. Det finns helt enkelt bättre möjligheter att implementera MTD idag.

I och med de stora investeringar som genomförts i forskningsprojekt kring MTD under de senaste åren (då främst av organisationer i USA, såsom Department of Homeland Security [DHS]) så är det hög tid att utvärdera områdets status. Denna rapport presenterar resultatet av en sådan utvärdering samt försöker svara på forskningsfrågorna i avsnitt 1.1.

1.1 Syfte och avgränsningar

För att erhålla en grundläggande förståelse för området utförde projektgruppen en förstudie där artiklar från olika konferenser inriktade på MTD (t.ex. [33] och [34]) detaljstuderades. Förstudien visade att det finns en mycket bred syn på vad MTD faktiskt innebär. Publikationer som utger sig för att handla om MTD kan handla om allt från spelteori (t.ex. den dynamiska kampen mellan cyberbrottslingar och försvarare [19] och resursallokeringsalgoritmer som används av den amerikanska kustbevakningen [32]) till tekniska lösningar för rörliga IP-adresser [8] och rörliga minnesadresser [30].

Ett resultat från förstudien var att teoretisk forskning som inte har en tydlig anknytning till faktiska lösningsförslag, så som spelteoretiska bidrag, avgränsades från studiens omfång. Denna avgränsning gjordes dels för att fokusera på lösningar som kan bidra med faktisk nytta i relativ närtid, och dels på grund av de resursmässiga begränsningarna inom projektet. Försvar som vilseleder attackeraren (antagonisten) utan att förändra var målet befinner sig avgränsades också från studiens omfång av den enkla anledningen att inget rör sig.

Det andra resultatet från förstudien var ett antal forskningsfrågor som kunde ligga som grund för den systematiska kartläggningen av området:

- Vilka varianter av moving target defenses föreslås?
- Hur användbara är de moving target defenses som föreslås?
- Vilken prestanda har de moving target defenses som föreslås?
- Hur effektiva är de moving target defenses som föreslås utifrån ett säkerhetsperspektiv?

1.2 Litteraturstudie

Detta projekt startade under februari 2014 med en förstudie. Denna ledde fram till de forskningsfrågor och avgränsningar som beskrevs i avsnitt 1.1. Förstudien

identifierade också ett ramverk som kunde användas för att kartlägga MTD-tekniker [22]. Detta ramverk skapades under en vetenskaplig konferens organiserad av Sandia National Laboratories.

Under mars 2014 utfördes en systematisk sökning¹ av vetenskapliga databaser (Scopus, IEEE, ACM och Springer), samt en manuell komplettering av denna i form av egna sökningar på internet. Detta arbete resulterade i 129 vetenskapliga artiklar att studera. Tio slumpmässigt utvalda artiklar av dessa 129 artiklar lästes av alla projektgruppens fem medlemmar för att identifiera eventuella brister i Sandias ramverk inför kartläggningsarbetet. Ramverket konkretiserades därefter av projektmedlemmarna och kompletterades med ytterligare ett antal delar. Det slutliga ramverket återges i bilaga B.

Därefter studerades titel, nyckelord och sammanfattning för alla de 129 artiklarna för att avgöra om de passade in i studiens ramar. Varje artikel lästes av två gruppmedlemmar för att uppnå högre datakvalitet samt möjliggöra analys av projektgruppens konsensus. Ibland fanns det inte konsensus rörande om en artikel passade studiens ramar. Sådana fall löstes genom diskussion inom gruppen som helhet. Denna process ledde fram till 56 artiklar som projektetgruppen ansåg ligga inom ramen för studien.

De 56 artiklarna studerades först som helhet för att säkerställa att de innefattade adekvat kvalitet och faktiskt passade studien. Om de ansågs lämpliga kartlades de enligt kartläggningsramverket. Liksom tidigare steg lästes varje artikel av två gruppmedlemmar, varvid bristande konsensus hanterades genom diskussioner inom gruppen som helhet. Denna gång kunde bristande konsensus exempelvis röra att en projektmedlem tyckte att ett försvar testades empiriskt medan en annan tyckte att utvärderingen snarare var en simuleringsmodell, något som är en svår gränsdragningsfråga i allmänhet.

Av de 56 artiklarna exkluderas 24 artiklar, i 19 fall för att de inte passade studiens ramar och i 5 fall på att det fanns nyare varianter av samma bidrag publicerade. Totalt 32 artiklar ligger därmed till grund för analysen av området som presenteras i kapitel 2. Översiktlig statistik baserad på kartläggningsramverket beskrivs i bilaga C. Det kompletta datamaterialet är möjligt att ladda ned².

Det finns två begränsningar i denna litteraturstudie som läsaren bör känna till. För det första är denna studie avgränsad mot bidrag som har en tydlig anknytning till faktiska lösningsförslag. Denna avgränsning innebär att diverse mer teoretiska bidrag som publicerats inom MTD-forum, såsom rent spelteoretiska bidrag, inte studerats. Detta bör beaktas vid analys av denna studies resultat. För det andra är

¹ Med nyckelorden ("dynamic defense" OR "dynamic defence" OR "moving target defense" OR "moving target defence") AND security)

² www.foi.se/mtd

studien sannolikt inte heltäckande. Flera av de studerade bidragen som bedömdes röra MTD nämnde exempelvis inte orden ”moving target defense” över huvud taget. Det är därför rimligt att anta att flera sådana bidrag förbisågs under de manuella sökningarna som utfördes av projektgruppen (den systematiska sökningen missade troligen sådana verk helt på grund av de använda sökorden, se fotnot 1).

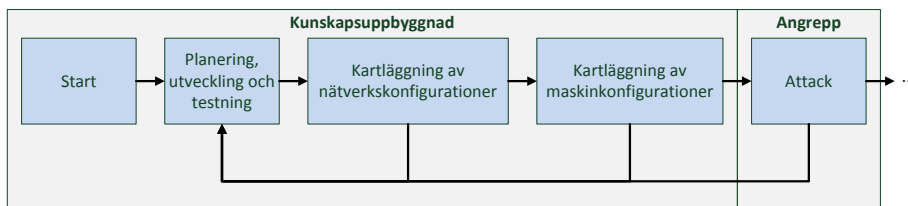
1.3 Disposition

Rapporten är strukturerad på följande sätt. Kapitel 2 presenterar och analyserar resultatet från studien. Kapitel 3 presenterar slutsatser och förslag på framtida arbete. Bilaga A beskriver buffer overruns, en vanlig typ av attack som många försvar explicit skyddar mot. Bilaga B beskriver det ramverk som nyttjades för att klassificera olika MTDs. Bilaga C beskriver en statistisk översikt av de kartlagda artiklarna.

2 Resultat

Hur attackerare komprometterar datorsystem har tidigare studerats av exempelvis Jonsson och Olovsson [36], McQueen et al. [42] och Schudel och Wood [52]. Jonsson och Olovsson menar att angreppsprocessen består av tre faser: (1) lärofasen (att skapa kunskap om målet), (2) den normala angreppsfasen (att testa välkända angreppsmetoder) och (3) den innovativa angreppsfasen (att ta fram nya angreppsmetoder). McQueen et al. delar också in angreppsprocessen i tre delprocesser: (1) när det finns färdig attackkod för en känd sårbarhet hos målet, (2) när det finns en känd sårbarhet hos målet men ingen färdig attackkod, och (3) när det inte finns någon känd sårbarhet hos målet. Schudel och Wood konkretiserar processbilden med exempelvis nätverkskartläggning och maskinkartläggning.

Alla dessa tre modeller beskriver två huvudsakliga aktiviteter som alla angreppsprocesser innefattar: kunskapsinsamling och angrepp. Kunskapsinsamling innefattar alla de aktioner som en attackerare måste genomföra för att identifiera sårbarheter, samt sätt att utnyttja dessa sårbarheter. Exempel på sådana aktiviteter kan vara att kartlägga nätverk, mjukvara och användare kring målet. Angreppet innefattar skickandet av skadlig kod i skarpt läge, d.v.s. den faktiska attacken mot målet. För angreppet är det av yttersta vikt att attackkoden är tillförlitlig, d.v.s. att den får det utfall som antagonisten förväntat sig.

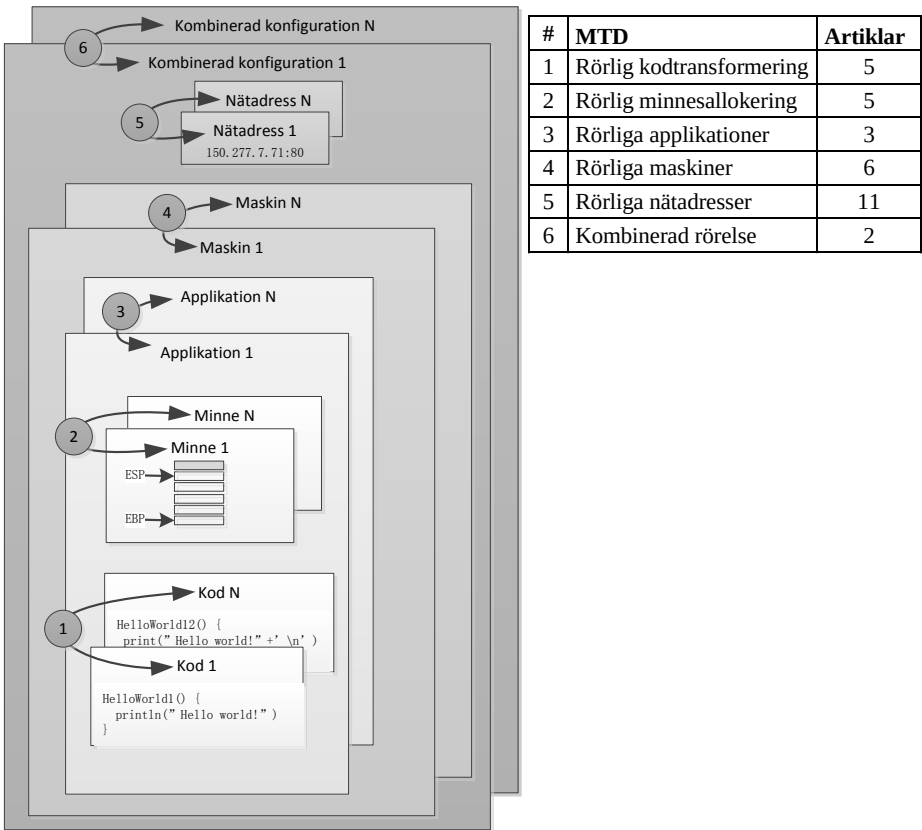


Figur 1. Översikt av attackprocess med fokus på delar av relevans för MTD.

En förenklad processmodell baserad på [36], [42] och [52] men med fokus på de delar som är av relevans för MTD återges i Figur 1. I denna processmodell består en cyberattack av kunskapsuppbyggnad och angrepp. Kunskapsuppbyggnad består vidare av planering, utveckling och testning, samt kartläggning av nätverk och maskiner. En vanlig typ av attack som ofta nämns av de studerade vetenskapliga artiklarna är kodinjektion genom sårbarheter av typen buffer overrun. Denna typ av sårbarhet beskrivs i bilaga A. Utöver kunskapsuppbyggnad och angrepp finns det även aktiviteter som involverar vad antagonister gör när attacker väl lyckats (exempelvis extrahering av information, privilegieeskalering eller försök att uppnå högre grad av persistens). Omfånget

för MTD rör dock typiskt kunskapsupbyggnad och angrepp – när ett angrepp väl lyckas är säkerhetseffektiviteten för ett MTD oftast antingen ingen eller mycket begränsad.

Syftet med ett MTD är att försvåra kunskapsupbyggnad och angrepp och därmed öka antagonists osäkerhet. Litteraturstudien identifierade MTD som förändrar en dators egenskaper på sex olika nivåer (se Figur 2). MTD på dessa nivåer beskrivs övergripande nedan och mer detaljerat i avsnitt 2.1 – avsnitt 2.6. En översikt av antalet kartlagda artiklar rörande de olika varianterna ges till höger i Figur 2.



Figur 2. De nivåer som MTD finns på i litteraturen.

1. *Rörlig kodtransformering* (#1 i Figur 2, innefattande [15], [40], [46], [50] och [56]) förändrar själva kodbasen för mjukvara. Detta kan exempelvis innebära att olika installationer av en applikation kompileras med olika (men snarlika) instruktioner eller att det injiceras extra kod i

en applikation vid varje exekvering. Rörlig kodtransformering syftar främst till att förhindra attacker, och inte så mycket till att förhindra kartläggningsarbetet. Denna variant beskrivs i avsnitt 2.1.

2. *Rörlig minnesallokering* (#2 i Figur 2, innefattande [17], [18], [24], [39] och [57]) förändrar var i minnet som en mjukvaras instruktioner laddas in. En dator har flera olika typer av minnen som mjukvaror nyttjar och som därmed kan förändras, så som *stack* (en begränsad men snabb minnesplats som hanterar temporär data såsom lokala variabler och returadresser) och *heap* (en större men långsammare minnesplats som hanterar mer persistent data såsom variabler deklarerade direkt i en klass). Likt *rörlig kodtransformering* förhindrar rörlig minnesallokering attacker, och har inget att göra med kartläggningsarbetet. Denna variant beskrivs i avsnitt 2.2.
3. *Rörliga applikationer* (#3 i Figur 2, innefattande [13], [14] och [47]) förändrar vilken applikation som för tillfället nyttjas för att realisera någon typ av tjänst. Exempelvis kan det handla om samma applikation som körs i flera olika instanser, där instansen som en användare exponeras för väljs ut beroende på när applikationen efterfrågas. Det kan också handla om olika applikationer som realiserar samma typ av tjänst, exempelvis en Apache-webserver som byts mot en Microsoft IIS-webserver. Rörliga applikationer påverkar främst själva attackerna men kan också påverka kartläggningsarbetet. Denna variant beskrivs i avsnitt 2.3.
4. *Rörliga maskiner* (#4 i Figur 2, innefattande [16], [20], [23], [29], [51] och [61]) förändrar vilken maskinkonfiguration som körs. Detta är relativt likt rörliga applikationer, men med skillnaden att hela maskinen (och inte bara en applikation) byts ut. Rörliga maskiner påverkar likt rörliga applikationer både attacker och kartläggningsarbetet. Denna variant beskrivs i avsnitt 2.4.
5. *Rörliga nätadresser* (#5 i Figur 2, innefattande [8], [9], [10], [11], [12], [26], [31], [35], [38], [49] och [58]) förändrar de adresser som används för kommunikation mellan datorer. Rörliga nätadresser påverkar främst kartläggningsarbetet. Denna variant beskrivs i avsnitt 2.5.
6. *Kombinerad rörelse* (#6 i Figur 2, innefattande [59] och [62]) involverar en förändring av alla nivåer i kombination (#1 - #5 Figur 2). Nätadresser, maskiner och applikationer förändras i kombination med syfte att uppnå särskilda konfigurationer. Minnesallokering och kod förändras antingen med tydlig avsikt att uppnå särskilda varianter eller indirekt då en förändring av operativsystem och applikationer innebär att även minnesallokeringen och koden förändras. Kombinerad rörelse

försvårar både kartläggningsarbetet och attacker. Denna variant beskrivs i avsnitt 2.6.

2.1 Rörlig kodtransformering

Rörlig kodtransformering handlar om att modifiera eller flytta exekverbar programkod. Många angrepp mot system utnyttjar brister i hur kod är skriven. En vanlig sådan brist är att applikationen inte validerar storleken på indata till dess funktionalitet, vilket öppnar upp för bufferöverskridningar³ (på engelska buffer overflows eller buffer overruns). I korta drag leder denna typ av brist (och många andra liknande brister) till att angripare kan ta över kontrollen av applikationens exekvering och därmed leda applikationsflödet till skadlig kod. Att ta kontroll över en process kräver dock att angriparen känner till hur kod placerats i minnet på maskinen och hur kod placeras i minnet bestäms av hur applikationen är skriven och hur den kompilerats. Genom att variera hur koden som realiserar funktioner är konstruerad och kompilerad kan problem därmed skapas för angripare. Två typer av förändringar diskuteras i litteraturen:

1. **Förändring av programkod på ett sätt som kräver en särskild tolkare av programkoden.** Exempelvis kan detta innebära att namnen på HTML-element byts för att försvåra automatisk inläsning av webbsidors formulär [56], eller att själva programmeringsspråket i sig förändras [15][40][50]. Den senare varianten bygger i de tre studerade bidragen på tekniken Instruction Set Randomization (ISR) som kan skydda mot en stor mängd attacker mot applikationer skrivna i vitt skilda språk. Exempelvis finns ISR implementerat både för kompillerade applikationer (skrivna i t.ex. C eller C++) och interpreterade applikationer (skrivna i t.ex. Perl eller Javascript).
2. **Förändring av programkod genom tillägg eller utbyte av operationer som påverkar kodbasen utan att påverka applikationens funktion.** Detta kan exempelvis ske med hjälp NOOP instruktioner⁴ eller ekvivalenta gadgets⁵ [46]. En av författarna till [46] var för övrigt en av de ursprungliga skaparna av ISR [37].

³ För den som är obekant med bufferöverskridningar rekommenderas beskrivningen i bilaga A.

⁴ *No Operation* (NOOP) är en assemblerinstruktion som enbart syftar till att öka programräknaren. Standardvarianten är NOOP (hexkod 90), men vilken instruktion som helst som inte påverkar applikationens tillstånd kan ses som en icke-operation, såsom *MOV ESP, ESP* (hexkod 89 E4).

⁵ En *gadget* är en serie instruktioner i den angripna applikationen som utför något värdefullt och sedan returnerar resultatet. T.ex. ökar värdet på ett processorregister och sedan returnerar detta resultat. Eftersom detta är kod som är en del av applikationen så är den oftast godkänd att köras av säkerhetsmekanismer som skyddar applikationen. Gadgets kan kombineras med varandra för att utgöra nya funktioner som inte finns i den ursprungliga applikationen, t.ex. kod för att stänga av just säkerhetsmekanismer.

2.1.1 Användbarhet och prestanda

Då det inte finns någon hårdvara som skapats med ISR i åtanke och en mjukvaruimplementation kräver att applikationer är statiskt länkade implementeras ISR för binärer i regel med hjälp av en emulator. Mätningar av prestandan för ISR har genomförts både med standardiserade tester (SPEC2000) och specialskräddade metoder av olika slag. Dessa tester visar på mellan någon procent overhead till 288% overhead, beroende på vilken applikation och typ av ISR som testats. De flesta tester visar på åtminstone dubbel eller trippel exekveringstid jämfört med utan skydd [15][40][50]. Skyddet mot automatisk inläsning av webbsidors formulär [56] testades på tre webbapplikationer där det uppmättes svarstider mellan 1% och 28% längre.

Prestandan för förändring av programkod genom utbyte eller tillägg av operationer utan att påverka applikationens funktion har också testats empiriskt. Tillägg av NOOPs påverkar CPU-overhead med mellan någon procent och 140%, beroende på typ av applikation och hur många NOOPs som läggs till [30]. Filstorleken på applikationen ökar på ett liknande sätt. Att byta ut instruktioner mot andra funktionellt sett ekvivalenta instruktioner har visats medföra en negligerbar CPU-overhead [46].

Ingen av artiklarna erbjuder dock mer än en översiktlig diskussion om hur den faktiska slutanvändaren påverkas av skyddet. De upplevda problemen med högre CPU-overhead, svarstid och större filstorlek beror exempelvis mycket på vilken applikation det handlar om. Detta gör det svårt att säga hur den upplevda användbarheten för den skyddade applikationen faktiskt påverkas. Ett annat potentiellt problem med rörlig kodtransformering är att det kan försvåra forensiskt arbete (t.ex. vid säkerhetsincidenter) och buggletande. Exempelvis innefattar båda dessa metoder emellanåt studier av det allokerade minnet för applikationer och maskiner, vilket kan vara problematiskt om minnet för samma applikation (givet samma tillstånd) ser fundamentalt annorlunda ut för varje exekvering⁶. Det förs ingen diskussion kring detta problem i de studerade bidragen.

2.1.2 Effektivitet

ISR [37] hävdas kunna skydda mot alla typer av attacker som kräver en dialog med en applikations inre eftersom kännedom om kryptonyckeln som gör koden begriplig behövs för att lyckas. Studier har dock visat att denna nyckel ibland kan läckas och ISR därmed kringgås [54]. Hur ofta detta skulle kunna vara fallet i praktiken är oklart. Det är också oklart hur ofta nyckeln bör bytas eller om den bör bytas alls utöver vid start av operativsystemet och/eller applikationen.

⁶ Se exempelvis Windows-verktyget Dr. Watson som tittar på minnet:
<http://support.microsoft.com/kb/308538>

Exempelvis föreslår Le Goues et al. [40] att nyckeln bör bytas varje 100ms, men motiveringen till just detta intervall är inte uppenbar. Ett annat problem är att äkta maskinkod förr eller senare måste ligga någonstans i minnet för att processorn ska kunna exekvera den. I denna korta tidslucka kan programkoden bli angripen. Eller så kan själva emulatoren som används av ISR-applikationen bli angripen. Ytterligare en begränsning är så kallade ROP-attacker⁷ där antagonisten nyttjar serier av gadgets (se fotnot 5) för att skapa skadlig kod. Då gadgets är en del av den skyddade applikationen kommer den skadliga koden redan ha korrekt kryptonyckel applicerad på sig, vilket gör att ISR kan kringgå. Då ROP-attacker kräver att antagonisten har vetskap om var särskilt utvalda gadgets befinner sig i en applikation kan de motverkas genom att gadgets byts ut mot andra för applikationen funktionellt sett lika gadgets.

Pappas et al. [46] har testat hur ofta funktionellt sett ekvivalenta gadgets kan identifieras. För de allra flesta filer (85% av alla testade) kunde ekvivalenta gadgets identifieras för 70% eller mer av alla existerande gadgets. Huruvida utförda byten kan härledas av en antagonist utforskades dock ej av författarna.

Byten av namn på HTML-element skulle sannolikt inte vara särskilt effektivt mot en motiverad angripare eftersom det är relativt enkelt att utvärdera vilka element som förändrats mellan laddningar av en webbsida. Exempelvis finns det bibliotek som kan användas för att läsa in hela DOM-strukturen för webbsidor i ett format som kan utfrågas med hjälp av egenskapade skript (exempelvis Nokogiri [3]). Ett sådant bibliotek skulle kunna nyttjas för att på ett enkelt sätt identifiera element som bytt namn eller struktur som förändrats.

2.2 Rörlig minnesallokering

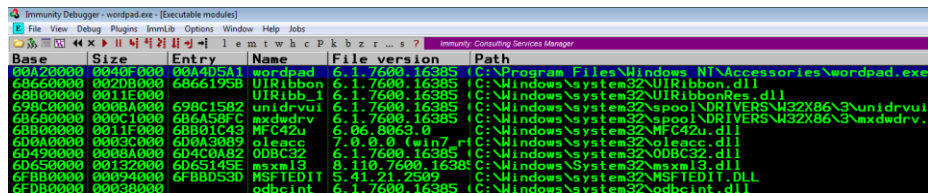
Rörlig minnesallokering handlar om att förändra minnesareor där applikationer sparar data. En applikation har flera minnesareor som kan förändras:

- **Stack:** en begränsad men snabb minnesplats som hanterar temporär data såsom lokala variabler och returadresser. Den maximala storleken på stacken tilldelas vid en applikations uppstart.
- **Heap:** en större men långsammare minnesplats som hanterar mer persistent data såsom variabler deklarerade direkt i ett programs klasser.
- **Minnesmappingssegment:** mappningar mellan applikationen och andra moduler som den nyttjar.

⁷ Return Oriented Programming (ROP) - attacker är på grund av dataexekveringsskyddet *No-eXecute* (NX) mycket vanligt förekommande i praktiken. NX finns implementerat för nästan alla nya applikationer, operativsystem och hårdvaror, och ROP är en teknik som kan nyttjas för att kringgå det.

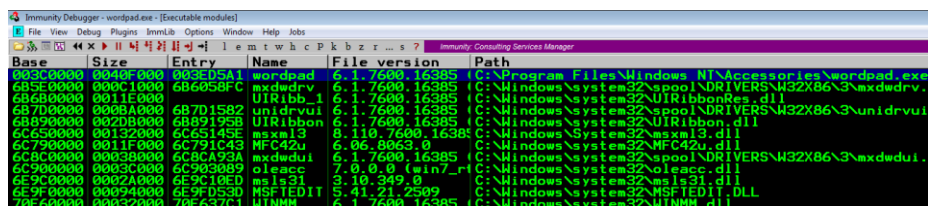
- **BSS-segment:** (Block Started by Symbol) ej initialiserade statiska variabler, såsom variabeln *A* direkt efter exekvering av raden *static char *A;*.
- **Datasegment:** statiska variabler som har tilldelats värde, såsom variabeln *A* direkt efter exekvering av raden *static char A = "MTD";*.
- **Textsegment:** applikationens binära image (kallas ibland för kodsegment).

Anledningen till varför det kan vara gynnsamt att förändra dessa olika minnesareor är att det förekommer attacker mot information sparad inom dem. Liksom *rörlig kodtransformering* (se avsnitt 2.1) rör de flesta forskningsprojekt prevention av bufferöverskridningar (se bilaga A). Denna attacktyp riktar mot specifika minnesadresser och kräver därför att en applikations adressrymd är densamma över tid (t.ex. mellan olika exekveringar) eller åtminstone känd innan attackkoden skickas iväg.



Base	Size	Entry	Name	File version	Path
00A20000	0040F000	00A405A1	wordpad	6.1.7600.16385	C:\Program Files\Windows NT\Accessories\wordpad.exe
68660000	0020B000	6866195B	UIRibbon	6.1.7600.16385	C:\Windows\system32\UIRibbon.dll
68B00000	0011E000		UIRibbon_1	6.1.7600.16385	C:\Windows\system32\UIRibbonRes.dll
698C0000	000BA000	698C1582	unidrdrv	6.1.7600.16385	C:\Windows\system32\spool\DRIVERS\W32X86\3\unidrdrv
6B600000	000C1000	6B6A58FC	mxdrv	6.1.7600.16385	C:\Windows\system32\spool\DRIVERS\W32X86\3\mxdrv
6B800000	0011F000	6B801C43	MFC42u	6.06.8063.0	C:\Windows\system32\MFC42u.dll
6D0A0000	0003C000	6D0A3089	oleacc	7.0.0.0 (win7_r)	C:\Windows\system32\oleacc.dll
6D490000	000BA000	6D4C0A82	ODBC32	6.1.7600.16385	C:\Windows\system32\ODBC32.dll
6D650000	00132000	6D65145E	msxml3	8.110.7600.16385	C:\Windows\System32\msxml3.dll
6FBB0000	0002A000	6FBB053D	MSFTEDIT	5.41.21.2509	C:\Windows\system32\MSFTEDIT.DLL
6FDB0000	00038000		odbcint	6.1.7600.16385	C:\Windows\system32\odbcint.dll

Figur 3. Windows 7 (32-bit) och Wordpad vid första uppstarten.



Base	Size	Entry	Name	File version	Path
00A20000	0040F000	00A405A1	wordpad	6.1.7600.16385	C:\Program Files\Windows NT\Accessories\wordpad.exe
6B5E0000	000C1000	6B6058FC	mxdrv	6.1.7600.16385	C:\Windows\system32\spool\DRIVERS\W32X86\3\mxdrv
68B00000	0011E000		UIRibbon_1	6.1.7600.16385	C:\Windows\system32\UIRibbonRes.dll
6B700000	000B0000	6B701582	unidrdrv	6.1.7600.16385	C:\Windows\system32\spool\DRIVERS\W32X86\3\unidrdrv
6B800000	002DB000	6B89195B	UIRibbon	6.1.7600.16385	C:\Windows\system32\UIRibbon.dll
6C650000	00132000	6C65145E	msxml3	8.110.7600.16385	C:\Windows\System32\msxml3.dll
6C790000	0011F000	6C791C43	MFC42u	6.06.8063.0	C:\Windows\system32\MFC42u.dll
6B800000	00030000	6B8A293A	mxdrv	6.1.7600.16385	C:\Windows\system32\spool\DRIVERS\W32X86\3\mxdrv
6C900000	0003C000	6C903089	oleacc	7.0.0.0 (win7_r)	C:\Windows\system32\oleacc.dll
6C9C0000	0002A000	6C9C10ED	msls31	3.10.349.0	C:\Windows\system32\msls31.dll
6E9F0000	00024000	6E9F053D	MSFTEDIT	5.41.21.2509	C:\Windows\system32\MSFTEDIT.DLL
70F60000	00032000	70F637C1	UIRibbon	6.1.7600.16385	C:\Windows\system32\UIRibbon.dll

Figur 4. Windows 7 (32-bit) och Wordpad vid andra uppstarten.

Den vanligaste typen rörlig minnesallokering är *Adress Space Layout Randomization* (ASLR, se [4]). ASLR randomiserar vilka minnesadresser som en applikations olika datatyper placeras på. Ett exempel kan ses nedan för två uppstarter av Windows 7 med ASLR påslaget och applikationen Wordpad i fokus. Figur 3 visar minnesadresserna för de inladdade modulerna vid första uppstarten; Figur 4 visar adresserna för de inladdade modulerna vid andra uppstarten. Tittar man på vart de olika modulerna laddas in (se kolumnen Entry i Figur 3 och Figur 4) kan man exempelvis se att *UIRibbon.dll* laddas in med start på adressutrymmet *0x6866195B* vid första uppstarten och *0x6B89195B* vid andra uppstarten. Då dessa två adresser skiljer sig åt kommer en attack som riktar mot

en specifik adress inom *UIRibbon.dll* troligtvis att misslyckas eftersom det är osannolikt att den eftersökta koden just då råkar befinna sig på den specificerade adressen.

Den vetenskapliga litteraturen har sedan introduktionen av ASLR presenterat ett antal förslag som innefattar förändring av minnesareorna där applikationer sparar data [17][18][24][39][57]. Alla dessa bidrag utom ett publicerades för cirka 10 år sedan, d.v.s. ungefär samtidigt som ASLR introduceras för kommersiella operativsystem. Lösningförslagen presenterade i artiklarna är alla relativt lika ASLR på ett övergripande plan men skiljer sig i detalj, vilket ger olika användbarhet och säkerhetseffektivitet.

2.2.1 Användbarhet och prestanda

Alla forskningsbidragen kring rörlig minnesallokering har samma kompatibilitetskrav. För att ASLR ska kunna tillämpas krävs det att mjukvaran har kompilerats med stöd för ASLR samt att operativsystemet och hårdvaran har stöd för ASLR. Det krävs dock inga speciella kodförändringar (utöver kompileringsflaggan) för att stödja ASLR.

För Windows påverkar inte ASLR körprestandan för en applikation nämnvärt. Det kan dock ta något längre tid att starta en applikation [1][25]. För Linux finns det möjliga prestandaproblem för applikationer som kompilerats med stöd för ASLR. Payer [48] visar på en CPU-overhead på upp till 26%. Detta beror på att Linux kräver att en applikation kompilerats som en Position Independent Executable (PIE) för att ASLR skall vara effektivt (annars randomiseras inte textsegmentet), något som inte krävs av Windows [48].

Idag har de allra flesta operativsystem och hårdvaror stöd för ASLR. Det är dock en bit kvar till komplett stöd för applikationer. Exempelvis visar en studie från 2013 [44] att endast 10% - 17% av operativsystembinärer i Linuxdistributioner (Debian, CentOS och Ubuntu) är kompilerade med stöd för PIE (och därmed effektivt ASLR). Orsaken bakom detta är sannolikt prestandaproblemen involverade med PIE. Exempelvis skriver utvecklarna av Linuxdistributionen Red Hat att *“The Fedora Engineering Steering Committee maintains a conservative list of packages that must be built using security features of GCC”* [43] och Ubuntu’s dokumentation att *“PIE has a large (5-10%) performance penalty on architectures with small numbers of general registers (e.g. x86), so it should only be used for a select number of security-critical packages”* [6].

Inget bidrag har undersökt hur tekniken faktiskt påverkar slutanvändaren (CPU- och minnesoverhead är inte samma sak som upplevd användbarhet). Däremot har de flesta forskningsbidragen kring rörlig minnesallokering genomfört laborativa tester för att mäta den CPU-overhead som introducerats av de föreslagna teknikerna. Detta görs typiskt med en standardiserad stressplattform kallad SPEC CPU [5]. CPU-overhead för de olika bidragen varierar stort. Exempelvis har

förslaget från Davi et al. [24] en CPU-overhead mellan 0.4% och 2.1% och en minnesoverhead på mellan 4% och 25%, medan förslaget från Bhatkar et al. [18] har på en CPU-overhead på mellan -1%⁸ och 21% och inte testat minnesoverhead. I praktiken når oftast endast teknologier med negligerbar overhead en implementation i vanliga kommersiella system [55]. Detta kan vara en anledning till varför ASLR, trots dess brister (se nästa avsnitt), fortfarande används i praktiken framför liknande skydd (som ISR beskrivet ovan).

2.2.2 Effektivitet

ASLR har haft en signifikant effekt på svårigheten att utveckla pålitlig attackkod – utan ASLR skulle tusentals idag relativt oproblematiska mjukvarubuggar vara kritiska säkerhetshål. Med det sagt så finns det brister med ASLR som gör att det ibland kan vara möjligt att gå runt skyddet trots att det är implementerat för både hårdvaran, operativsystemet och den attackerade mjukvaran. Detta avsnitt fokuserar på att beskriva olika brister med ASLR och hur de olika forskningsbidragen adresserar dessa.

Ett stort problem är att det relativt ofta finns moduler som inte stödjer ASLR. Detta är ett problem då det kan räcka med att en enda sådan modul är inladdad i minnet för att en antagonist skall kunna kringgå ASLR. Ett signifikant exempel är *msvcr71.dll*, en Windows-modul som under 2011 inte var kompillerad med stöd för ASLR och därför alltid laddades in på samma plats i minnet [21]. Detta medförde att alla attacker mot någon applikation körades i någon variant av Windows kunde gå runt ASLR, oavsett om applikationen i sig hade stöd för ASLR eller inte.

Ett annat problem kan illustreras genom en noggrannare jämförelse av startadresserna för *UIRibbon.dll* i Figur 3 (*0x6866195B*) och Figur 4 (*0x6B89195B*): endast byte ett och två (*6866* mot *6B89*) skiljer sig åt, byte tre och fyra förblir desamma (*195B*). Detta är en följd av att ASLR i 32-bitsarkitekturer enbart randomiserar två bytes. Shacham et al. [53] visade att attacker mot applikationer i en 32-bitsarkitektur, där varje attack försöker gissa rätt minnesadress, kan gå runt ASLR på 216 sekunder när misslyckade gissningar leder till automatisk omstart av processen (som det ofta gör i till exempel webbservar). För att få en effektiv ASLR-implementation krävs det därför en 64-bitsarkitektur (eller mer) som erbjuder större utrymme för randomisering, och därmed kräver betydligt fler gissningar för att kringgå. För en 64-bitsarkitektur visar mätningar utförda av Otterstad [45] att det under (för antagonisten) mycket goda förhållanden krävs mellan 1 och 34 timmar att kringgå ASLR. Ett annat problem som illustreras av exemplet i Figur 3 och Figur 4 är att endast modulers startadresser randomiseras – de relativa avstånden inom

⁸ Författarna ger ingen förklaring till varför den uppmätta overheaden blev negativ.

en modul förblir desamma. Om en attackerare kan lista ut en enda minnesadress i applikationen så kan resten av den (och möjligtvis även biblioteksfunktioner som kallas på inom modulen) ofta kartläggas med hjälp av den. I praktiken kan sådant minnesläckage exempelvis uppnås genom en sårbarhet av typen *format string* [27].

Ett tredje problem med ASLR är att dess effektivitet beror på vilken typ av operativsystem som används. Windows stödjer inte PIE, vilket Linux och Mac gör. Utan PIE har ASLR-kompilerade biblioteksfiler (.dll-filer) samma adressplats för varje ny exekvering av en applikation om inte maskinen startas om emellan. Exempelvis, om en attack mot Microsoft IIS riktas mot en adress inom *msvcr71.dll* och misslyckas så kommer den nyttjade IIS-processen krascha och starta om. Vid denna omstart kommer *msvcr71.dll* att befinna sig på samma plats som tidigare. Om PIE använts hade biblioteksfilen haft en ny plats vid varje omstart.

Ett fjärde problem med ASLR är att adressrymden inte förändras under exekveringstiden av en applikation utan enbart när applikationen startas upp. Om en attackerare därmed kan läcka information om applikationen utan att få den att krascha kan ASLR så småningom kringgås.

Ett femte problem med ASLR är att inte alla minnesareor randomiseras. BSS-segmentet randomiseras inte; textsegmentet⁹ och datasegmentet randomiseras, men de nuvarande använda adresserna för dessa segment ligger sparade i CPU-registerna CS (code segment) och DS (data segment). Om en sårbarhet involverar data i dessa minnesareor kan ASLR ibland kringgås.

Ett sjätte problem med ASLR är att det kan kringgås om attackeraren får möjlighet att styra minnesallokeringen för den attackerade processen. Detta är exempelvis möjligt om angriparen kan få målet att ansluta mot en webbserver. Då kan webbservern be webbrowsern att allokeras data med hjälp av Javascript eller ActiveX. Vid en sådan händelse kan attackeraren återkommande allokera samma skadliga kod tills hela applikationens heap är fylld med den. När detta är gjort kan attackeraren sikta på en arbiträr adress i heapen och med stor chans träffa någon av den skadliga kod som allokerats dit. Denna typ av attack kallas för *heap spray*. En lite mer intelligent variant av denna attack går ut på att allokera data med en särskild storlek som attackeraren vet kommer allokeras till en särskild minnesarea inom heapen. Då träffsäkerheten blir större behöver inte lika mycket data allokeras. Denna variant kallas *precision heap spray*.

Fokus för de olika forskningsbidragen ([17], [18], [24], [39] och [57]) ligger i att behandla någon eller flera av dessa sex begränsningar. En begränsning som behandlas i flera av forskningsbidragen är att ASLR enbart randomiserar startadresserna för moduler och inte de relativa avstånden mellan instruktioner

⁹ Givet Windows eller Linux med en PIE-stödjande modul.

inom modulen randomiseras ([17], [18], [24] och [57]). Som lösning på detta problem föreslås olika metoder för att generera nya relativa avstånd mellan instruktioner inom moduler över tid. Exempelvis presenterar Davi et al. [24] en metod för att flytta runt kodblock i minnet. En annan begränsning som behandlas av flera bidrag är att ASLR enbart randomiserar en delmängd av en applikations alla minnesareor ([17], [18], [24] och [39]). Forskarna föreslår randomisering av ytterligare minnesareor så som textsegmentet och minnesmappingssegmentet. Ingen av forskningsartiklarna tar dock hänsyn till att alla applikationer kanske inte väljer att stödja tekniken. De hanterar heller inte heap spray-attacker.

2.3 Rörliga applikationer

Rörliga applikationer handlar om att växla mellan olika varianter av applikationer som utför samma typ av tjänst. Till exempel kan man växla mellan olika programvaror i en webserver; en Linux-server kan växla mellan Apache, Nginx och lighttpd, där alla tre levererar samma webbsida. Det kan också handla om samma applikation, fast med olika versioner, konfigurationer, kompilersinställningar, eller programbibliotek.

Detta område har två inriktningar:

1. Applikationen placeras i en virtuell container. Denna container roterar mellan olika applikationskonfigurationer, både proaktivt (vid randomiserade tillfällen) och reaktivt (när en attack mot applikationen upptäckts) [47].
2. Applikationen bryts ned i olika celler, där en cell är en instans av applikationen som har en särskild konfiguration, t.ex. en särskild minnesrandomisering eller en särskild programversion. Systemet växlar sedan mellan dessa celler på ett slumpmässigt sätt [14][13].

Syftet med applikationsrotationen är att ständigt förändra attackytan då de olika applikationskonfigurationerna kan ha olika sårbarheter.

2.3.1 Användbarhet och prestanda

Teoretiskt sett bör en växling av applikationer inte vara särskilt tidskrävande, då alternativa konfigurationer kan finnas redo i datorns primärminne. En nackdel är att detta kan bli utrymmeskrävande, men det är ett mindre problem i moderna datorer. En större nackdel är att en applikations nuvarande tillståndsmodeLL kan vara svår att föra över mellan roterande applikationer. Exempelvis bygger vanliga webbapplikationer på TCP, vilket är sessionsbaserat. Det är inte självklart hur denna typ av information skall synkas mellan sessioner utan att negativt påverka upplevelsen för användaren.

Inget av de tre studerade bidragen har testats i en verklig miljö. Paulos et al. [47] nämner att det är viktigt att välja en applikationsrotationsfrekvens som passar situationen eftersom en högre frekvens leder till lägre tillgänglighet. Storleken på denna overhead är dock okänd. Prestandan för applikationsrotation baserad på celler [13][14] har testats med en simuleringsmodell. Denna simulering utfördes dock under relativt oklara omständigheter och förutsatte att den skyddade applikationen var attackerad. Validiten hos dessa resultat är därför oklar.

2.3.2 Effektivitet

Inget av bidragen har testats empiriskt. Azab och Eltoweissy [13][14] för en ytlig diskussion kring vilka typer av attacker som teoretiskt sett skulle kunna stoppas, såsom hur många gissningar som krävs för att identifiera rätt applikation, och genomför sedan simuleringen nämnd i föregående sektion. Paulos et al. [47] påstår att det utförts experiment, men visar inga faktiska resultat och tillhandahåller inte heller någon referens till var sådana beskrivs.

Medan rörliga applikationer teoretiskt sett har möjligheten att försvåra otaliga typer av attacker då olika konfigurationer har olika sårbarheter är frågan om detta är fallet i praktiken. Exempelvis genomförs många attacker av botten som automatiskt skickar attackkod mot internet-exponerade portar och applikationer utan att ta hänsyn till vad det faktiskt är för mjukvara. Exempelvis görs det frekventa inloggningsförsök mot publikt tillgängliga remote desktop protocol (RDP)-applikationer utan att ta hänsyn till vad för maskin som det handlar om [60]. Eftersom automatiska attacker är frekventa kan en mycket kort exponering av en sårbar applikation räcka för att maskinen komprometteras. För riktade attacker kommer det sannolikt att vara ett effektivt skydd på kort sikt. På lång sikt kommer dock antagonister med största sannolikhet att lära sig att kringgå skyddet. Exempelvis kan detta ske genom att först övervaka applikationens konfiguration för att skapa profiler för de konfigurationer som detekteras och sedan koppla särskild attackkod mot de olika identifierade konfigurationerna. Denna information kan därefter nyttjas för att automatiskt skicka relevant attackkod baserat på vilken konfiguration som för tillfället exponeras.

2.4 Rörliga maskiner

En maskin (d.v.s. en instans av ett operativsystem) innehåller applikationer, kod och minne. En maskin kan vara både virtuell (där hårdvara exponeras mot maskinen via en virtualiseringsmodul såsom Virtualbox) eller nativ (där hårdvara exponeras direkt mot maskinen). Som framgår i tidigare avsnitt är flera delar av en maskin viktiga för en angripare och genom att byta ut maskinen mot en annan kan därmed alla delar förändras, vilket försvårar för angriparen.

I litteraturen finns ett antal förslag på hur maskinvarianter kan göras rörliga. Huang och Ghosh [29] tar fasta på att webbservrar är ett vanligt mål för angrepp och föreslår en lösning som byter ut mjukvaran på maskinerna enligt ett roterande schema. I och med detta kan ett stort antal parametrar ändras, exempelvis vilken typ av webserver som körs (t.ex. Apache eller Tomcat), vilka andra applikationer som körs (t.ex. olika versioner av Java), vilket operativsystem som nyttjas (t.ex. olika varianter av Linux) och vilken typ av virtualiseringsmiljö som används (t.ex. VMware eller Virtualbox). Roeder och Schneider [51], Battista et al. [16] och Casola et al. [20], förordar liknande lösningar. Roeder och Schneider [51] testar att vid regelbundna intervall starta om distribuerade brandväggar och lagringstjänster och vid omstart föra in en annan version med förändrad kod. En liknande ansats föreslås av Battista et al. [16], men med fokus på inbäddade trådlösa sensorer och begränsat till att byta kryptosystem och protokoll. Casola et al. [20] föreslår en liknande lösning för att byta ut firmware på inbyggda system (också främst rörande nyttjat kryptosystem). Ett mer generisk och abstrakt tänk baserat på genetisk programmering föreslås av Crouse et al. [23]. I genetisk programmering introduceras återkommande mutationer i kod och resultatet av dessa testas för att identifiera de bästa varianterna. Crouse et al. [23] föreslår att detta bör användas för att skapa olika konfigurationer på system och förändra dessa över tid utan att utvecklingen (kromosompoolen) stagnerar.

Utöver förändringar av maskinens kod, minne och konfiguration kan maskinens fysiska plats vara av vikt för angripare, och att flytta runt maskiner fysiskt kan därför vara meningsfullt. Zhang et al. [61] föreslår en lösning för hur virtuella maskiner kan flyttas runt i molnlösningar. Att flytta runt virtuella maskiner är tänkt att göra det svårt för användare av molntjänsten att använda sidokanaler (t.ex. CPU-last) för att spionera på andra molnanvändare.

2.4.1 Användbarhet och prestanda

Huang och Ghosh [29] noterar att det finns två problem med deras lösning: (1) kostnaden för att skapa systemet med alla varianter av mjukvara och (2) svårigheten att hantera det när det är i drift. Uppenbara problem med att byta ut en hel maskin är dels det momentana avbrott som då sker och dels de tillstånd som behöver föras över till den nya maskinen på ett eller annat sätt (t.ex. existerande http-sessioner). Detta är inte bara ett problem för Huangs och Ghoshs förslag, som kräver att webbapplikationen är tillståndslös, utan för alla förslag där kod skall bytas ut under körning. I Roeders och Schneiders [51] tester syns tydlig påverkan på prestanda (latens och genomströmning) från såväl maskinbyte och återställning.

Det finns inga oberoende tester av de föreslagna lösningarna. Inte heller presenteras det tester utförda i skarpa miljöer, utan dessa utförs i specialbyggda testmiljöer. Andra problem som kan följa av att återkommande byta mellan olika

mjukvaruversioner är därför okända. Men ett rimligt antagande är att det inte bara försvårar buggletande för angripare, utan också för de som driftar och underhåller systemet.

2.4.2 Effektivitet

De artiklar som studerats är otydliga med vilka typer av angrepp som maskinbyten ska skydda mot. Men det skydd som ges av att byta ut maskiner är naturligtvis beroende av vad som byts ut. I alla ovanstående lösningar är en idé att skifta mjukvara som körs på maskinen. Avsnitt 2.3.2 tar upp effektiviteten av att göra dessa rörliga. En annan möjlighet som följer av att byta maskin är möjligheten att skifta konfiguration. Det finns helt klart möjligheter att skapa diversitet över tid. Huang och Ghosh [29] identifierar till exempel 452 möjliga sätt att köra webbapplikationer baserade på programmeringsspråket Ruby on Rails. Samtidigt är denna variation liten i jämförelse med andra alternativ (t.ex. ASLR som på 32-bit maskiner ger 2^{16} möjliga kombinationer).

En fördel av att byta maskiner jämfört med enbart mjukvaran på dem är att infekterade maskiner då kan ”tvättas” från existerande skadlig kod. Frågan är dock om detta är en verksam lösning i praktiken då en automatisk attackplattform som kartlagt existerande konfigurationer nästan ögonblickligen skulle kunna kompromettera även denna nya konfiguration.

2.5 Rörliga nätadresser

Det finns en hel del skrivet om möjligheten att variera ett nätverks utseende för att därigenom försvåra för en angripare. De mål man vill uppnå är att försvåra kartläggning av nätet, skydda mot riktade attacker och att anonymisera kommunikationen gentemot någon som lyssnar längs vägen. Den helt dominerande idén för att uppnå MTD för nätverk är att kontinuerligt byta de adresser som används vid kommunikationen så att de som inte känner till hur bytena sker ser en ström av meddelanden mellan ständigt varierande sändare och mottagare. Detta kallas *rörliga nätadresser* och tas upp i [8], [9], [10], [11], [12], [26], [31], [35], [38], [49] och [58].

Nätverkskommunikation med rörliga nätadresser fungerar precis som vanlig nätverkskommunikation. Skillnaden är att adresstilldelningen endast är dynamisk och endast är giltig en kort stund i förhållande till den tid en angripare behöver för att angripa de maskiner som är kopplade till nätverket. Den dynamik som t.ex. uppnås med en vanlig DHCP-lösning är i allmänhet långt från tillräckligt för att realisera någon form av MTD. Med DHCP är adresstilldelningar ofta timslånga och vid förnyad tilldelning delas ofta samma adress ut igen till samma maskin. För effektivt MTD behövs en variation på minutnivå snarare än timnivå. Dessutom behöver alla noder i MTD-nätverket måste vara överens om hur och

när adresser byts för att undvika att funktionen i nätverket påverkas när olika noder har olika syn på vilka adresser som ska användas.

De rörliga nätadresserna kan vara på nätverksnivå, dvs. att det är IP-adresserna som byts (t.ex. [26], [58] och [9]). Denna typ av adresshopp hjälper till att skydda hela maskiner eftersom det blir svårt att veta på vilken IP-adress en viss, för angriparen åtråvärd, maskin finns. Det är ingen principiell skillnad på rörliga nätadresser i IPv4 och IPv6. Men eftersom adresserna kan varieras över den adressrymd som nätet i fråga har tilldelats blir skillnaden stor i praktiken. IPv4 lider av adressbrist, vilket gör att de flesta nätverk har en adressrymd som är ungefär lika stor som mängden av maskiner som ingår. IPv6 däremot har en mycket stor adressrymd att tillgå vilket har betydelse ner till enskilda lokala nät.

De rörliga nätadresserna kan också vara på transportlagernivå, vilket i praktiken innebär att det är portnummer som varieras. Ett portnummer, tillsammans med en IP-adress pekar ut ett visst program på en viss dator, dvs. en servertjänst eller ett klientprogram. När dessa adresser hoppar hjälper det till att skydda enskilda program eller tjänster eftersom det blir svårt för en angripare att veta vilka program eller tjänster det är som kommunicerar över nätverket och hur dem ska nås. Det går förstås också att kombinera hoppande IP-adresser med hoppande portnummer, vilket exempelvis [12] och [38] gör.

Det finns två huvudspår i hur kommunikationsfunktionen upprätthålls i nätverk med rörliga nätadresser. Det ena är att alla i nätet var för sig kan avgöra när och hur en adress ändras, och kan anpassa sin kommunikation efter detta (t.ex. [58] och [38]). Detta kräver synkronisering av tid och hoppalgoritm så att adressbytet sker vid rätt tillfälle och adressändringen blir densamma hos alla inblandade parter. Problem kan uppstå på grund av små fel i tidssynkroniseringen och på grund av att meddelanden inte överförs momentant vilket gör att adressbytet sker under transporten av meddelandet. Detta går att hantera genom att t.ex. låta både nuvarande och närmast föregående adresstilldelning vara giltiga i hela systemet. På det sättet blir små fel i tidssynkroniseringen oproblematiske och meddelanden under transport hinner nå sitt mål innan adressen de använder blir ogiltig. Denna typ av lösning kräver dock att adressrymden är stor nog att rymma två giltiga adresser per adresserbar maskin.

Det andra huvudspåret för att upprätthålla kommunikationsfunktionen är att ha ett mindre antal maskiner med publikt kända adresser. När man vill komma i kontakt med någon annan maskin eller tjänst frågar man någon av de kända maskinerna om vilken adress som just nu ska användas för att nå den önskade destinationen. I vissa föreslagna lösningar ([49]) är det DNS-systemet som används. I andra fall ([35]) är det dedikerade proxy-servrar som bara har denna uppgift, i just de nätverk som ska skyddas.

Oavsett vilken lösning som används är det alltid tanken att de hoppande adresserna ska vara kända och tillgängliga för vanliga program och vanliga

användare. Var man lagt funktionaliteten som omvandlar från kända adresser och tjänster till de för tillfället gällande dynamiska adresserna varierar. Vanliga lösningar är att lägga funktionaliteten i en hypervisor (t.ex. [58]), i operativsystemet (t.ex. [26]) eller i nätverksstacken (t.ex. [9]).

2.5.1 Användbarhet och prestanda

Det kan finnas praktiska problem med att få rörliga nätadresser att fungera med uppkopplingsbaserade kommunikationsprotokoll som TCP. En uppkoppling görs inledningsvis mot en viss adress, men kan över sin livslängd råka ut för att motpartens adress byts flera gånger. Beroende på hur hoppandet implementerats kan det vara besvärligt att hantera dessa byten av adresser i ett protokoll där motpartens adress förväntas vara fast.

Detta problem är dock allmänt känt och har därför adresserats väl. De flesta presenterade lösningar klarar av att hantera detta problem, och andra liknande svårigheter. I vissa lösningar, exempelvis [10], är det inget problem alls eftersom översättningen till den dynamiska adressen görs i ett lager utanför operativsystemet och därför inte syns alls, varken för programmet som kommunicerar, nätverksstacken eller operativsystemet i övrigt.

Vid utvecklandet av olika varianter av hoppande adresser blir det lätt fokus på hur svårt det blir för en attackerare att kartlägga eller attackera någon nod i nätverket. Något som kanske inte diskuteras lika mycket är hur legitima användare påverkas, exempelvis den eventuella påverkan på nätverksprestandan. MT6D [26] är en av de föreslagna implementationerna av hoppande adresser där prestandatest har genomförts. Testet genomfördes med två datorer som kopplades in på samma subnät. Datorernas utrustades med den nödvändiga mjukvaran och konfigurerades att byta IP-adresser var tionde sekund. Resultatet blev en genomsnittlig överföringshastighet som var under 2 MB/s, vilket kan jämföras med en genomsnittlig överföringshastighet på 54-108 MB/s utan MT6D. Kompletterande tester tyder på att den låga prestandan kan bero på mjukvaran snarare än själva hoppandet av adresser.

2.5.2 Effektivitet

Rörliga nätadresser kan, enligt de studerade bidragen, försvåra flera av stegen en angripare vill utföra (se Figur 1). Ett av stegen i attackprocessen är att kartlägga det nät som ska angripas. Denna kartläggning kan göras genom att lyssna på kommunikation som flyter i nätverket eller genom att göra kontaktförsök med olika adresser i nätet för att se om och vad som svarar. Genom att använda rörliga nätadresser blir båda dessa kartläggningsmetoder besvärligare. Vid passiv lyssning kommer angriparen vid varje adressbytestidpunkt att få börja om kartläggningen eftersom alla adresser plötsligt byts ut till nya. Sker bytena tillräckligt ofta hinner angriparen bara se en liten del av alla för tillfället använda

adresser innan alla adresser byts ut. Då kan det bli svårt för en angripare att ens ta reda på hur många maskiner som finns inkopplade i nätet eftersom denne inte vet om ett svar kommer från en maskin som redan upptäckts, fast nu med en annan adress.

En vanlig typ av attack är överbelastningsattacker. Idén med en sådan attack är att skicka så många förfrågningar till en viss adress att den maskin som använder adressen inte hinner hantera dem. Den får då svårt att hantera legitim kommunikation. Genom att använda hoppande adresser blir det, enligt de studerade bidragen, svårt för angriparen att skicka någon kommunikation alls till målmaskinen, och därmed blir det också svårt att överbelasta den.

En ytterligare effekt av rörliga nätadresser som beskrivs är anonymisering, dvs. att någon som sitter längs vägen och lyssnar på kommunikationen har svårt att analysera vem som pratar med vem eftersom adresserna ändras hela tiden. Detta döljer både exakt vilka som kommunicerar med varandra och delvis också trafikmönstret.

Det finns dock frågor att ställa sig angående värdet av rörliga nätadresser i skyddssyfte. Genom att låta adresserna "hoppa" blir det svårt för de som inte är en del i hoppsystemet att kommunicera med de som ingår i hoppsystemet. Det är denna svårighet som anses hindra både kartläggning och överbelastningsattacker. Det är dock inte helt klart varför rörliga nätadresser är bättre än en välkonfigurerad brandvägg, eller en VPN-lösning. De flesta av de egenskaper som uppnås med rörliga nätadresser kan uppnås också med dessa mer standardiserade lösningar. Anonymisering kan också lösas med ett VPN, varför inte heller detta problem verkar kräva rörliga nätadresser för att lösas.

En annan sak att notera är att mot en insider ger de rörliga nätadresser i allmänhet inte något skydd alls. En insider ska kunna kommunicera i nätverket och den tekniska lösningen kommer alltså inte att hindra dennes kommunikation mer än annan, legitim kommunikation. Detta gäller strikt talat inte bara insiders, utan alla angrepp som tar sin början i en maskin som är en del av hoppsystemet. Det innebär att vid ett inbrott där angriparen bereder sig tillträde till en maskin på insidan så har rörliga nätadresser ett begränsat skyddsvärde. Även skadlig kod som kommer in i en maskin genom att användaren initierar kommunikation med angriparen, t.ex. via en websida som råkar vara fientlig, kommer att hindras minimalt av de rörliga nätadresserna.

Som synes av den korta diskussionen är det inte uppenbart att skyddsvärdet av rörliga nätadresser är större än vad man kan uppnå med mer klassiska metoder. En relevant fråga är också hur svårt det kan bli att felsöka i ett nätverk med rörliga nätadresser. Diagnosticering av nätverksproblem påminner i många avseenden om att kartlägga nätverket, varför sådan felsökning kommer att försvåras på ungefär samma sätt som en attackerares kartläggning. Standardiserade verktyg kommer att vara svåra eller omöjliga att använda och

logganalys kommer att kräva adressöversättning för att ge någon information. Rimligen blir även intrångsdetekteringssystem förvirrade av rörliga nätadresser.

Det har genomförts en större utvärdering av hur effektivt rörliga nätadresser är i avseende att göra det svårare för en attackerare att kartlägga ett nät [38]. Denna utvärdering gjordes på uppdrag av DARPA och baserades på ett förslag på rörliga nätadresser kallat DYNAT. Experimentet som låg till grund för utvärderingen genomfördes med ett antal attackerare, vars uppgift var att hitta den kritiska servern som befanns sig någonstans i nätet, och ett antal försvarare som konfigurerat DYNAT. Experimentet bestod av åtta olika scenarion där antingen konfigurationen av DYNAT eller attackerarnas förhandsinformation om systemet varierades. Utvärderingen visar på att användningen av DYNAT gjorde det svårare för attackerarna att kartlägga nätet och därmed identifiera den kritiska servern. Det observerades att DYNAT åtminstone dubblade eller fyrdubblade arbetsinsatsen för attackerarna (beroende på scenario). Det konstaterades också att attackerarnas beteende förändrades eftersom det inte längre var troligt att hitta något genom att bara peta runt i systemet. För att attackerarna skulle lyckas identifiera den kritiska servern krävdes omfattande datainsamling och analys, vilket innebar att de behövde befinna sig i nätverket en längre period. Därmed ökade risken att de skulle hinna upptäckas. För försvararna hade användandet av DYNAT minimal påverkan på klienter, servrar och nätverksarkitektur.

2.6 Kombinerad rörelse

Kombinerad rörelse involverar att förändra nätadresser, maskiner och applikationer i kombination (och därmed även kod och minnesallokering). Två bidrag identifierades inom denna kategori, Zhuang et al. [62] och Yackoski et al. [59], vilka båda involverar designscheman för hur nätadresser, virtuella maskiner och applikationer kan förändras över tid. Båda lösningsförslagen bygger på att en särskild konfigurationshanteringsnod styr över design och implementation av olika konfigurationer. Förslaget presenterat av Yackoski et al. [59] har enligt författarna implementerats och testats för ett antal olika typer av hårdvara och mjukvara.

2.6.1 Användbarhet och prestanda

Zhuang et al. [62] medger att konfigurationsbyten kan medföra en negativ upplevelse för användaren av det skyddade systemet då det kan vara svårt att synkronisera applikationsspecifika tillstånd så som HTTP-sessioner. De utforskar dock inte hur detta kan lösas eller hur stora konsekvenser det skulle kunna få. Yackoski et al. [59] diskuterar varken användbarhet eller prestanda.

Det är svårt att se att kombinationen av rörliga nätadresser, rörliga maskiner och rörliga applikationer skulle motverka de enskilda bristerna som finns med dessa

varianter (se respektive diskussionskapitel i kapitel 2). Mer sannolikt är att varianter som bygger på multipel rörelse ärver alla brister och om något skapar än större problem vid synkning av applikationstillstånd då än fler sessionstillstånd måste synkroniseras vid konfigurationsbyte.

2.6.2 Effektivitet

Zhuang et al. [62] studerade effektiviteten för denna lösning givet olika förändringsfrekvenser med hjälp av en simuleringsmodell och fann att en högre förändringsfrekvens medförde lägre sannolikhet för lyckade attacker. Yackoski et al. [59] beskriver endast ett hypotetiskt exempel på en attack som behandlas med det föreslagna skyddet.

På samma sätt som beskrivs i avsnitt 2.6.1 finns det inga direkta skäl att tro att det finns starka samspelseffekter mellan rörliga nätadresser, rörliga maskiner och rörliga applikationer. Effektiviteten höjs dock sannolikt mer än om vardera typ av skydd körs i isolering.

3 Diskussion

Som nämndes i inledning är MTD egentligen inget nytt; forskning och implementation av rörliga försvar har funnits ett flertal år. Exempelvis implementerades rörlig minnesallokering i form av ASLR för Linux under 2001 och tester av effektiviteten för rörliga nätadresser studerades av DARPA under samma år [41]. Trots detta är de allra flesta av de 32 artiklar som kartlades i denna rapport publicerade efter 2010. Det fokus som idag finns på forskning och utveckling av rörliga försvar initierades huvudsakligen för ett par år sedan när konceptet MTD myntades. Bland andra så var DHS med och finansierade forskning på konceptet. Idag har det förslagits MTD på sex olika nivåer:

- 1) rörlig kodtransformering ([15], [40], [46], [50] och [56]),
- 2) rörlig minnesallokering ([17], [18], [24], [39] och [57]),
- 3) rörliga applikationer ([13], [14] och [47]),
- 4) rörliga maskiner ([16], [20], [23], [29], [51] och [61]),
- 5) rörliga nätadresser ([8], [9], [10], [11], [12], [26], [31], [35], [38], [49] och [58]),
- 6) kombinerad rörelse ([59] och [62]).

Fokus för artiklarna ligger på introduktion av skydd snarare än studier av kvaliteten och värdet av dem. Kvaliteten och värdet diskuteras med teori och/eller simuleringsresultat som stöd, men sällan med empirisk data som grund. Empirisk data har använts i nio av artiklarna för att testa (från ett säkerhetsperspektiv) effektivitet, i en av artiklarna för att utvärdera användbarhet och i 16 av artiklarna för att testa prestanda. Detta gör att det överlag är svårt att veta hur effektiva skydden faktiskt är, eller hur mycket de påverkar den faktiska användningen av det skyddade systemet. En kort sammanfattning av läget följer i avsnitt 3.1. Därefter ges förslag på framtida arbete inom området i avsnitt 3.2.

3.1 Sammanfattning av skyddens värde

Rörlig minnesallokering är genom ASLR den enda typen av MTD som nått en bred praktisk tillämpning. ASLR finns idag implementerat för i princip varje hårdvara och operativsystem och är ett effektivt skydd som har transformerat otaliga säkerhetsbrister till harmlösa buggar. Det finns ett fåtal brister som gör att det ibland kan kringgå (se avsnitt 2.2.2) men forskning angående rörlig minnesallokering verkar ha nått sin kulmen för cirka 10 år sedan.

Rörlig kodtransformering bygger i de flesta fall på tekniken ISR, som bland forskare anses ha potential [15][40][50]. ISR kan tyvärr inte hantera så kallade ROP-attacker och har prestandaproblem, något som forskarna idag jobbar på

genom utbyte av så kallade gadgets (se fotnot 5). Ingen variant har testats under realistiska förutsättningar (t.ex. angående hotmodell) och det är därför relativt oklart både hur effektiva de är samt hur de påverkar användbarheten för de skyddade systemen.

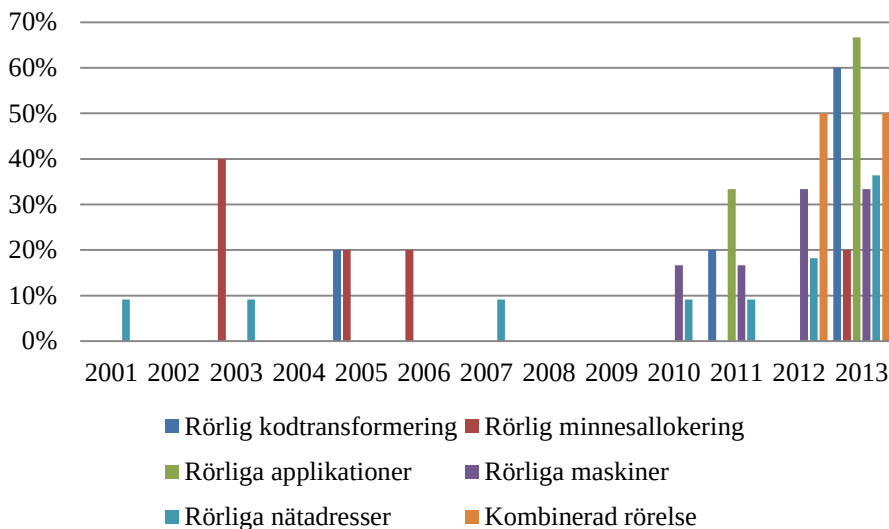
Rörliga applikationer och maskiner har teoretiskt sett möjligheten att försvåra otaliga typer av attacker då olika konfigurationer har olika sårbarheter. Då inga empiriska tester av skydden har genomförts i skarp miljö är det dock oklart om detta är fallet i praktiken. Sannolikt är skydden verksamma till en början innan tekniker för gå runt dem fått stor spridning. Exempelvis kan antagonisten skapa skript för att automatiskt kartlägga och exploatera maskiner. Rörliga applikationer och maskiner har också problemet att det finns mängder av viktiga applikationsegenskaper som måste överföras mellan rotationer för att användarnas upplevelse inte skall degraderas (exempelvis kakor under HTTP-sessioner). Hur detta skall genomföras i praktiken är ännu på ritbordet.

Rörliga nätadresser står under ett liknande dilemma som rörliga applikationer och maskiner. Skyddet kommer sannolikt att försvåra externa attacker som kräver direkt kunskap om IP-nummer. Tyvärr är denna effekt sannolikt bara temporär på samma sätt som för rörliga applikationer och maskiner. Ytterligare ett problem är att rörliga nätadresser sannolikt inte kan skydda mot attacker där antagonisten är en del av det skyddade nätverket. Och på samma sätt som för rörliga applikationer och maskiner kan skyddet även degradera användarens upplevelse på grund av problem med synkning av viktiga applikationsegenskaper mellan rotationer, samt försvåra nätverksadministratörers arbete med felsökning. Dessutom är det oklart vad rörliga nätadresser skyddar mot som inte redan idag går att skydda sig mot med etablerade tekniker som brandväggar och VPN-tunnlar.

Kombinerad rörelse ärver sannolikt alla fördelar och nackdelar som finns för rörliga applikationer, maskiner och nätadresser, men med ännu större problem vid synkronisering av sessionsinformation mellan konfigurationsbyten.

3.2 Framtida arbete

Många av forskningsprojekten som rör MTD befinner sig fortfarande i sin linda. Detta illustreras i Figur 5 som presenterar den procentuella fördelningen över publikationsåren för de 32 kartlagda bidragen kategoriserade efter de sex identifierade typerna av MTD. Undantaget rörlig minnesallokering är artiklarna i regel publicerade efter 2010. Det är alltså förhållandevis nya idéer och man kan förvänta sig en ökad mognad och mer forskning inom de närmsta åren. Omvärldsbevakning och fortsatta studier rörande MTDs vore därmed av värde. Förutom dessa aktiviteter anser författarna att det framförallt är tre huvudområden som bör adresseras i framtida studier: användarens upplevelse, realistiska hotmodeller och empiriska tester i skarpa miljöer.



Figur 5. Procentuell fördelning av publiceringsåren för bidrag av olika rörliga skyddsmekanismer.

När det kommer till hur MTD-skydden påverkar annat än säkerhet så har prestanda (t.ex. CPU-overhead) ibland studerats för vissa skydd. Men det finns få diskussioner om hur skydden påverkar användarens upplevelse. Som en följd av detta nämns sällan hur olika typer av användare (t.ex. utvecklare, nätverksadministratörer och slutanvändare) kan påverkas på olika sätt. Då minskad användbarhet är ett stort hinder för allmänt bruk av IT-försvar [55] bör det därmed i regel läggas större vikt på att studera hur användare påverkas av dem.

När det kommer till vilka hot som de förespråkade skydden är verksamma mot finns i bästa fall en mycket begränsad diskussion och ofta ingen diskussion alls. Framtida bidrag bör tydligt beskriva vilka hot som är relevanta och hur dessa hot behandlas av det föreslagna skyddet. Attacktyperna beskrivna av Evans et al. [28] skulle kunna utgöra en bas för en sådan analys.

Forskning kring MTD har hittills primärt fokuserat på att skapa olika varianter på skydd, men det har inte hunnit testats om dessa varianter är verksamma under mer realistiska förutsättningar. Är det exempelvis bättre att en förändring sker reaktivt eller bör den ske proaktivt? Bör förändringen vara randomiserad eller deterministisk? Hur ofta bör förändringar utföras? Frågor som dessa har ännu inga bra svar då näst intill inga valida tester har utförts. Framtida arbete bör därmed fokusera på att testa både effektiviteten och användbarheten för MTDs

under realistiska förutsättningar, såsom realistiska hotmodeller. Ett sätt att uppnå detta skulle vara att implementera MTDs i så kallade cyber ranges¹⁰, där riktiga penetrationstestare kan försöka att kringgå skydden under realistiska förutsättningar.

¹⁰ En cyber range är en labbresurs som används för att skapa datornätverk för användning vid experiment, tävlingar och övningar i IT-säkerhet.

4 Referenser

- [1] Address space layout randomization (aslr). <http://msdn.microsoft.com/en-us/library/bb430720.aspx#aslr>. Accessed: 2014-08-01.
- [2] Cybersecurity game-change research & development recommendations. http://www.nitrd.gov/pubs/-CSIA_IWG_%20Cybersecurity_%20GameChange_RD_%20Recommendations_20100513.pdf. Accessed: 2014-08-01.
- [3] Nokogiri. <http://nokogiri.org/>. Accessed: 2014-08-01.
- [4] Pax adress space layout randomization. <https://pax.grsecurity.net/docs/-aslr.txt>. Accessed: 2014-08-01.
- [5] Spec cpu 2006. <http://www.spec.org/cpu2006/>. Accessed: 2014-08-01.
- [6] Ubuntu security features. <https://wiki.ubuntu.com/Security/Features#pie>. Accessed: 2014-08-01.
- [7] Us national vulnerability database (nvd). <http://nvd.nist.gov/>. Accessed: 2014-08-01.
- [8] Ehab Al-Shaer. Toward network configuration randomization for moving target defense. In *Moving Target Defense*, pages 153–159. Springer, 2011.
- [9] Ehab Al-Shaer, Qi Duan, and Jafar Haadi Jafarian. Random host mutation for moving target defense. In *Security and Privacy in Communication Networks*, pages 310–327. Springer, 2013.
- [10] Massimiliano Albanese, Alessandra De Benedictis, Sushil Jajodia, and Kun Sun. A moving target defense mechanism for manets based on identity virtualization. In *Communications and Network Security (CNS), 2013 IEEE Conference on*, pages 278–286. IEEE, 2013.
- [11] Spyros Antonatos, Periklis Akritidis, Evangelos P Markatos, and Kostas G Anagnostakis. Defending against hitlist worms using network address space randomization. *Computer Networks*, 51(12):3471–3490, 2007.
- [12] Michael Atighetchi, Partha Pal, Franklin Webber, and Christopher Jones. Adaptive use of network-centric mechanisms in cyber-defense. In *Object-Oriented Real-Time Distributed Computing, 2003. Sixth IEEE International Symposium on*, pages 183–192. IEEE, 2003.
- [13] Mohamed Azab and Mohamed Eltoweissy. Defense as a service cloud for cyber-physical systems. In *Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom), 2011 7th International Conference on*, pages 392–401. IEEE, 2011.

- [14] Mohamed Azab and Mohamed Eltoweissy. Chameleonsoft: Software behavior encryption for moving target defense. *Mobile Networks and Applications*, 18(2):271–292, 2013.
- [15] Elena Gabriela Barrantes, David H Ackley, Stephanie Forrest, and Darko Stefanovic. Randomized instruction set emulation. *ACM Transactions on Information and System Security (TISSEC)*, 8(1):3–40, 2005.
- [16] Ermanno Battista, Valentina Casola, Antonino Mazzeo, and Nicola Mazzocca. Siren: a feasible moving target defence framework for securing resource-constrained embedded nodes. *International Journal of Critical Computer-Based Systems*, 4(4):374–392, 2013.
- [17] Sandeep Bhatkar, Daniel C DuVarney, and R Sekar. Efficient techniques for comprehensive protection from memory error exploits. In *Usenix Security*, 2005.
- [18] Sandeep Bhatkar, Daniel C DuVarney, and Ron Sekar. Address obfuscation: An efficient approach to combat a broad range of memory error exploits. In *USENIX Security*, volume 3, pages 105–120, 2003.
- [19] Daniel Bilar, George Cybenko, and John Murphy. Adversarial dynamics: The conficker case study. In *Moving Target Defense II*, pages 41–71. Springer, 2013.
- [20] Valentina Casola, Alessandra De Benedictis, and Massimiliano Albanese. A moving target defense approach for protecting resource-constrained distributed devices. In *Information Reuse and Integration (IRI), 2013 IEEE 14th International Conference on*, pages 22–29. IEEE, 2013.
- [21] corelanc0d3r. Universal dep/aslr bypass with msvc71.dll and mona.py. <https://www.corelan.be/index.php/2011/07/03/universal-depaslr-bypass-with-msvc71-dll-and-mona-py/>. Accessed: 2014-08-01.
- [22] Sean Crosby, Justin E Doak, Jason J Haas, Ryan Helinski, and Christopher C Lamb. Dynamic defense workshop: From research to practice. 2013.
- [23] Michael B Crouse, Errin W Fulp, and Daniel Cañas. Improving the diversity defense of genetic algorithm-based moving target approaches. In *Proc. Natl. Symp. Moving Target Research*, 2012.
- [24] Lucas Vincenzo Davi, Alexandra Dmitrienko, Stefan Nürnberger, and Ahmad-Reza Sadeghi. Gadge me if you can: secure and efficient ad-hoc instruction-level randomization for x86 and arm. In *Proceedings of the 8th ACM SIGSAC symposium on Information, computer and communications security*, pages 299–310. ACM, 2013.

- [25] Will Dormann. Differences between aslr on windows and linux. <http://www.cert.org/blogs/certcc/post.cfm?EntryID=191>. Accessed: 2014-08-01.
- [26] M. Dunlop, S. Groat, R. Marchany, and J. Tront. Implementing an ipv6 moving target defense on a live network. In *National Symposium on Moving Target Research*, 2012.
- [27] Tyler Durden. Bypassing pax aslr protection. *Phrack Magazine*, 59, 2002.
- [28] David Evans, Anh Nguyen-Tuong, and John Knight. Effectiveness of moving target defenses. In *Moving Target Defense*, pages 29–48. Springer, 2011.
- [29] Yih Huang and Anup K Ghosh. Introducing diversity and uncertainty to create moving attack surfaces for web services. In *Moving Target Defense*, pages 131–151. Springer, 2011.
- [30] Todd Jackson, Andrei Homescu, Stephen Crane, Per Larsen, Stefan Brunthaler, and Michael Franz. Diversifying the software stack using randomized nop insertion. In *Moving Target Defense II*, pages 151–173. Springer, 2013.
- [31] Jafar Haadi Jafarian, Ehab Al-Shaer, and Qi Duan. Openflow random host mutation: transparent moving target defense using software defined networking. In *Proceedings of the first workshop on Hot topics in software defined networks*, pages 127–132. ACM, 2012.
- [32] Manish Jain, Bo An, and Milind Tambe. Security games applied to real-world: Research contributions and challenges. In *Moving Target Defense II*, pages 15–39. Springer, 2013.
- [33] Sushil Jajodia, Anup K Ghosh, VS Subrahmanian, Vipin Swarup, Cliff Wang, and X Sean Wang. *Moving Target Defense II*. Springer, 2013.
- [34] Sushil Jajodia, Anup K Ghosh, Vipin Swarup, Cliff Wang, and X Sean Wang. *Moving Target Defense*. Springer, 2011.
- [35] Quan Jia, Kun Sun, and Angelos Stavrou. Motag: Moving target defense against internet denial of service attacks. In *Computer Communications and Networks (ICCCN), 2013 22nd International Conference on*, pages 1–9. IEEE, 2013.
- [36] Erland Jonsson and Tomas Olovsson. A quantitative model of the security intrusion process based on attacker behavior. *Software Engineering, IEEE Transactions on*, 23(4):235–245, 1997.
- [37] Gaurav S Kc, Angelos D Keromytis, and Vassilis Prevelakis. Countering code-injection attacks with instruction-set randomization. In *Proceedings of the 10th ACM conference on Computer and communications security*, pages 272–280. ACM, 2003.

- [38] Dorene Kewley, Russ Fink, John Lowry, and Mike Dean. Dynamic approaches to thwart adversary intelligence gathering. In *DARPA Information Survivability Conference & Exposition II, 2001. DISCEX'01. Proceedings*, volume 1, pages 176–185. IEEE, 2001.
- [39] Chongkyung Kil, Jinsuk Jim, Christopher Bookholt, Jun Xu, and Peng Ning. Address space layout permutation (aslp): Towards fine-grained randomization of commodity software. In *Computer Security Applications Conference, 2006. ACSAC'06. 22nd Annual*, pages 339–348. IEEE, 2006.
- [40] Claire Le Goues, Anh Nguyen-Tuong, Hao Chen, Jack W Davidson, Stephanie Forrest, Jason D Hiser, John C Knight, and Matthew Van Gundy. Moving target defenses in the helix self-regenerative architecture. In *Moving Target Defense II*, pages 117–149. Springer, 2013.
- [41] David Levin. Lessons learned in using live red teams in ia experiments. In *DARPA Information Survivability Conference and Exposition*, volume 1, page 110. IEEE Computer Society, 2003.
- [42] Miles A McQueen, Wayne F Boyer, Mark A Flynn, and George A Beitel. Time-to-compromise model for cyber risk reduction estimation. In *Quality of Protection*, pages 49–64. Springer, 2006.
- [43] Grant Murphy. Position independent executables (pie). <https://securityblog.redhat.com/2012/11/28/position-independent-executables-pie/>. Accessed: 2014-08-01.
- [44] Adrián Navarro. How effective is aslr on linux systems? <http://securityetali.es/2013/02/03/how-effective-is-aslr-on-linux-systems/>. Accessed: 2014-08-01.
- [45] Christian Otterstad. Brute force bypassing of aslr on linux. *Norsk informasjonssikkerhetskonferanse (NISK)*, 2012, 2012.
- [46] Vasilis Pappas, Michalis Polychronakis, and Angelos D Keromytis. Practical software diversification using in-place code randomization. In *Moving Target Defense II*, pages 175–202. Springer, 2013.
- [47] Aaron Paulos, P Pal, R Schantz, and Brett Benyo. Moving target defense (mtd) in an adaptive execution environment. In *Proceedings of the Eighth Annual Cyber Security and Information Intelligence Research Workshop*, page 62. ACM, 2013.
- [48] Mathias Payer. Too much pie is bad for performance. 2012.
- [49] Dhananjay Phatak, Alan T Sherman, Nikhil Joshi, Bhushan Sonawane, Vivek G Relan, and Amol Dawalbhakta. Spread identity: A new dynamic address remapping mechanism for anonymity and ddos defense. *Journal of Computer Security*, 21(2):233–281, 2013.

- [50] Georgios Portokalidis and Angelos D Keromytis. Global isr: Toward a comprehensive defense against unauthorized code execution. In *Moving Target Defense*, pages 49–76. Springer, 2011.
- [51] Tom Roeder and Fred B Schneider. Proactive obfuscation. *ACM Transactions on Computer Systems (TOCS)*, 28(2):4, 2010.
- [52] Gregg Schudel and Bradley Wood. Adversary work factor as a metric for information assurance. In *Proceedings of the 2000 workshop on New security paradigms*, pages 23–30. ACM, 2001.
- [53] Hovav Shacham, Matthew Page, Ben Pfaff, Eu-Jin Goh, Nagendra Modadugu, and Dan Boneh. On the effectiveness of address-space randomization. In *Proceedings of the 11th ACM conference on Computer and communications security*, pages 298–307. ACM, 2004.
- [54] Ana Nora Sovarel, David Evans, and Nathanael Paul. Where’s the feeb? the effectiveness of instruction set randomization. In *Usenix Security*, 2005.
- [55] László Szekeres, Mathias Payer, Tao Wei, and Dawn Song. Sok: Eternal war in memory. In *Security and Privacy (SP), 2013 IEEE Symposium on*, pages 48–62. IEEE, 2013.
- [56] Shardul Vikram, Chao Yang, and Guofei Gu. Nomad: Towards non-intrusive moving-target defense against web bots. In *Communications and Network Security (CNS), 2013 IEEE Conference on*, pages 55–63. IEEE, 2013.
- [57] Jun Xu, Zbigniew Kalbarczyk, and Ravishankar K Iyer. Transparent runtime randomization for security. In *Reliable Distributed Systems, 2003. Proceedings. 22nd International Symposium on*, pages 260–269. IEEE, 2003.
- [58] Justin Yackoski, Harry Bullen, Xiang Yu, and Jason Li. Applying self-shielding dynamics to the network architecture. In *Moving Target Defense II*, pages 97–115. Springer, 2013.
- [59] Justin Yackoski, Jason Li, Scott A DeLoach, and Xinming Ou. Mission-oriented moving target defense based on cryptographically strong network dynamics. In *Proceedings of the Eighth Annual Cyber Security and Information Intelligence Research Workshop*, page 57. ACM, 2013.
- [60] Lenny Zeltser. The dark side of remote desktop. <http://blog.zeltser.com/-post/9533396210/dark-side-of-remote-desktop-protocol>. Accessed: 2014-08-01.
- [61] Yulong Zhang, Min Li, Kun Bai, Meng Yu, and Wanyu Zang. Incentive compatible moving target defense against vm-colocation attacks in clouds. In *Information Security and Privacy Research*, pages 388–399. Springer, 2012.
- [62] Rui Zhuang, Su Zhang, Scott A DeLoach, Xinming Ou, and Anoop Singhal. Simulation-based approaches to studying effectiveness of moving-target network defense. In *National Symposium on Moving Target Research*, 2012.

Bilaga A. Buffer overrun

I augusti 2014 fanns det 4995 förekomster av buffer overrun sårbarheter i NVD (då cirka 10% av alla sårbarheter i den totala databasen). Merparten av dessa sårbarheter har potentialen att ge en attackerare samma privilegier som det sårbara programmet (vilket ofta innebär administratör).

Ett exempel på en buffer overrun sårbarhet kan se i programmet nedan. Programmet tar användarens första argument och kopierar detta till en lokal variabel *MyVar* (d.v.s., det sparas på stacken – se avsnitt 2.2) som har storleken 8 bytes.

```
#include <string.h>

void do_something(char *Buffer)
{
    char MyVar[8];
    strcpy(MyVar, Buffer);
}

int main (int argc, char **argv)
{
    do_something(argv[1]);
}
```

Exekveras programmet med färre än 8 bytes som argument (t.ex. 4 bytes, såsom "AAAA") kommer programmet bete sig som programmeraren tänkt sig. Om användaren däremot använder ett argument som är längre än 7 bytes (t.ex. "AAAABBBB") kommer funktionen *strcpy* skriva mer till *MyVar* än vad som allokerats för variabeln¹¹. Detta kommer medföra att användarens argument skriver över tidigare allokerad information på stacken. En viktig typ av information som befinner sig på stacken sedan tidigare är vart programmets flöde skall gå när funktionen *do_something* är färdig. Programmet är tänkt att gå tillbaka till *main*, men om argumentet som nyttjats är tillräckligt långt kan det komma att skriva över dirigeringen till *main*. Detta är vad som gör programmet sårbart: en attackerare kan instruera programmets flöde till någon annan del av minnet, såsom en plats där attackeraren har placerat skadlig kod.

För att lyckas med denna attack krävs det dock vanligen att attackkoden inkorporeras med en adress som antingen direkt eller indirekt¹² pekar på den

¹¹ 7 bytes räcker då en null-byte (0x00) behövs för att avsluta en sträng.

¹² Med indirekt referens menas att attackkoden pekar på en adress som i sin tur pekar på ett minnessegment som innehåller den skadliga koden. Exempelvis, om den skadliga koden ligger på toppen av stacken kan assemblerinstruktionen *JMP ESP* nyttjas för att nå den (då

skadliga koden. Ett exempel kan ges för attackkoden för en sårbarhet i applikationen FTPPad¹³. Denna applikation har en buffer som är sårbar för mer än 3342 bytes. Attackkoden för denna sårbarhet har tre hårdkodade adresser: `0x77FA6556`, `0x77f86556` och `0x77FA6526` (valet mellan dessa beror på vilket operativsystem som applikationen körs under). Alla dessa adresser pekar på instruktionen `CALL ESI`¹⁴, d.v.s. platsen där den skadliga koden har placerats.

Dessa hårdkodade adresser blir ett problem för attackerare om en applikations inre (t.ex. dess adressrymd) förändras över tid och rum.

processorregistret ESP pekar på toppen av stacken). I detta fall kan attackeraren kalla på en adress i applikationen som utför t.ex. `JMP ESP` eller `CALL ESP`.

¹³ <http://www.exploit-db.com/exploits/16726/>

¹⁴ ESI är ett processorregister

Bilaga B. Kartläggningssramverk

Nr.	Category	Subcategory	States	Description
1	Increases the uncertainty for the attacker		Yes/No	This category is fulfilled (=Yes) if the purpose of the defense is to increase the uncertainty for the attacker at successfully accomplishing malicious actions covered by defense.
2	When changes are made	Proactive	Yes/No	This category is fulfilled (=Yes) if the purpose of the defense is to provide proactive changes of the protected system.
		Reactive	Yes/No	This category is fulfilled (=Yes) if the purpose of the defense is to provide reactive changes of the protected system (e.g., changes after certain attacker behavior is observed).
		Unknown	Yes/No	This category is fulfilled (=Yes) if it is unknown under what conditions that changes are made to the protected system.
3	Application domain	Network	Yes/No	If the defense is employed to change network architecture. Includes all levels of the network stack.
		Host	Yes/No	If the defense is deployed to change the configuration of a node/host. Includes all layers of the software stack.
		Unknown	Yes/No	This category is fulfilled (=Yes) if the application domain is unknown.
		Free text		Here you can summarize the essence of what the application domain concerns
4	Type of change	Deterministic	Yes/No/Unknown	This category is fulfilled (=Yes) if the changes posed by a defense are made in a deterministic fashion (i.e., the defender has complete control of the change). Unknown is fulfilled (=Yes) if the changes posed by a defense are unknown.
		Random	Yes/No/Unknown	This category is fulfilled (=Yes) if the changes posed by a defense are made in a random fashion (i.e., the defender has no control of the change). Unknown is fulfilled (=Yes) if the changes posed by a defense are unknown.
5	Defensive Goals	Deception-centric	Yes/No/Unknown	If the purpose of the defense is to increase the time/effort required for compromise and at the same time hide from the attacker that its doing so. Unknown is the case if the defensive goals are unknown.
		Delay-focused	Yes/No/Unknown	If the purpose of the defense is to increase the time/effort required for compromise. Unknown is the case if the defensive goals are unknown.
6	Efficiency	Hypothetical	Yes/No	hypothetical examples, having unclear relation to actual phenomena and degree of realism.
		Simulation	Yes/No	simulation methods of some target. This simulation can be software-based (e.g., using Opnet) or software/hardware-based. However, it should produce data from a scenario that does not reflect the actual hardware/software/configuration that it is meant to be implemented in.
		Empirical	Yes/No	empirical methods, such as systematic experimental gathering of data from operational settings.
		Theoretical	Yes/No	formal or precise theoretical arguments to support results. This includes e.g. relating a metric or model to what attribute properties it is meant to represent and measure. This is often applicable for systems that involve guessing and entropy. However, when systems involve deterministic changes or estimations, theoretical evaluations are generally not applicable.
7	Usability	Hypothetical	Yes/No	hypothetical examples, having unclear relation to actual phenomena and degree of realism.
		Simulation	Yes/No	simulation methods of some target. This simulation can be software-based (e.g., using Opnet) or software/hardware-based. However, it should produce data from a scenario that does not reflect the actual hardware/software/configuration that it is meant to be implemented in.
		Empirical	Yes/No	empirical methods, such as systematic experimental gathering of data from operational settings.
		Theoretical	Yes/No	formal or precise theoretical arguments to support results. This includes e.g. relating a metric or model to what attribute properties it is meant to represent and measure. This is often applicable for systems that involve guessing and entropy. However, when systems involve deterministic changes or estimations, theoretical evaluations are generally not applicable.
8	Effectiveness	Hypothetical	Yes/No	hypothetical examples, having unclear relation to actual phenomena and degree of realism.
		Simulation	Yes/No	simulation methods of some target. This simulation can be software-based (e.g., using Opnet) or software/hardware-based. However, it should produce data from a scenario that does not reflect the actual hardware/software/configuration that it is meant to be implemented in.
		Empirical	Yes/No	empirical methods, such as systematic experimental gathering of data from operational settings.
		Theoretical	Yes/No	formal or precise theoretical arguments to support results. This includes e.g. relating a metric or model to what attribute properties it is meant to represent and measure. This is often applicable for systems that involve guessing and entropy. However, when systems involve deterministic changes or estimations, theoretical evaluations are generally not applicable.
9	Frequency of changes	Described	Yes/No	How often changes should occur is described by the article
		Free text		Here you can summarize the essence of the prescribed changes

Bilaga C. Statistik från kartläggning

Denna bilaga beskriver statistiska nyckelvärden från kartläggningen. För en detaljerad bild hänvisas läsaren till www.foi.se/mtd.

C.1 Rörlig kodtransformering

Tabell 1. När förändringar utförs

Proaktivt	Sant		Falskt	
Reaktivt	Sant	Falskt	Sant	Falskt
Antal	2	3	0	0

Tabell 2. Typ av förändring

Deterministisk	Sant		Falskt	
Randomiserad	Sant	Falskt	Sant	Falskt
Antal	1	0	4	0

Tabell 3. Defensiva mål

Villfarelse	Sant		Falskt	
Fördröjning	Sant	Falskt	Sant	Falskt
Antal	3	0	2	0

Tabell 4. Utvärdering av prestanda

Utvärdering	Hypotetisk	Simulering	Empirisk	Teoretisk
Sant	3	3	2	1
Falskt	2	2	3	4

Tabell 5. Utvärdering av användbarhet

Utvärdering	Hypotetisk	Simulering	Empirisk	Teoretisk
Sant	2	0	0	0
Falskt	3	0	0	0

Tabell 6. Utvärdering av effektivitet

Utvärdering	Hypotetisk	Simulering	Empirisk	Teoretisk
Sant	3	0	3	1
Falskt	2	0	2	4

Tabell 7. Frekvens av förändringar

Frekvens	Beskrivs i artikel
Sant	5
Falskt	0

C.2 Rörlig minnesallokering

Tabell 8. När förändringar utförs

Proaktivt	Sant		Falskt	
Reaktivt	Sant	Falskt	Sant	Falskt
Antal	3	2	0	0

Tabell 9. Typ av förändring

Deterministisk	Sant		Falskt	
Randomiserad	Sant	Falskt	Sant	Falskt
Antal	0	0	5	0

Tabell 10. Defensiva mål

Villfarelse	Sant		Falskt	
Fördröjning	Sant	Falskt	Sant	Falskt
Antal	0	0	5	0

Tabell 11. Utvärdering av prestanda

Utvärdering	Hypotetisk	Simulering	Empirisk	Teoretisk
Sant	5	1	5	0
Falskt	0	4	0	5

Tabell 12. Utvärdering av användbarhet

Utvärdering	Hypotetisk	Simulering	Empirisk	Teoretisk
Sant	1	0	0	0
Falskt	4	5	5	5

Tabell 13. Utvärdering av effektivitet

Utvärdering	Hypotetisk	Simulering	Empirisk	Teoretisk
Sant	5	0	3	4
Falskt	0	5	2	1

Tabell 14. Frekvens av förändringar

Frekvens	Beskrivs i artikel
Sant	5
Falskt	0

C.3 Rörliga applikationer

Tabell 15. När förändringar utförs

Proaktivt	Sant		Falskt	
Reaktivt	Sant	Falskt	Sant	Falskt
Antal	2	1	0	0

Tabell 16. Typ av förändring

Deterministisk	Sant		Falskt	
Randomiserad	Sant	Falskt	Sant	Falskt
Antal	1	0	2	0

Tabell 17. Defensiva mål

Villfarelse	Sant		Falskt	
Fördröjning	Sant	Falskt	Sant	Falskt
Antal	0	0	3	0

Tabell 18. Utvärdering av prestanda

Utvärdering	Hypotetisk	Simulering	Empirisk	Teoretisk
Sant	1	0	1	2
Falskt	2	3	2	1

Tabell 19. Utvärdering av användbarhet

Utvärdering	Hypotetisk	Simulering	Empirisk	Teoretisk
Sant	1	0	0	0
Falskt	2	3	3	3

Tabell 20. Utvärdering av effektivitet

Utvärdering	Hypotetisk	Simulering	Empirisk	Teoretisk
Sant	1	1	0	2
Falskt	2	2	3	1

Tabell 21. Frekvens av förändringar

Frekvens	Beskrivs i artikel
Sant	2
Falskt	1

C.4 Rörliga maskiner

Tabell 22. När förändringar utförs

Proaktivt	Sant		Falskt	
Reaktivt	Sant	Falskt	Sant	Falskt
Antal	2	4	0	0

Tabell 23. Typ av förändring

Deterministisk	Sant		Falskt	
Randomiserad	Sant	Falskt	Sant	Falskt
Antal	4	1	1	0

Tabell 24. Defensiva mål

Villfarelse	Sant		Falskt	
Fördröjning	Sant	Falskt	Sant	Falskt
Antal	1	0	5	0

Tabell 25. Utvärdering av prestanda

Utvärdering	Hypotetisk	Simulering	Empirisk	Teoretisk
Sant	3	1	3	3
Falskt	3	3	3	3

Tabell 26. Utvärdering av användbarhet

Utvärdering	Hypotetisk	Simulering	Empirisk	Teoretisk
Sant	2	1	0	0
Falskt	4	5	6	6

Tabell 27. Utvärdering av effektivitet

Utvärdering	Hypotetisk	Simulering	Empirisk	Teoretisk
Sant	2	2	0	2
Falskt	4	4	6	4

Tabell 28. Frekvens av förändringar

Frekvens	Beskrivs i artikel
Sant	0
Falskt	6

C.5 Rörliga nätadresser

Tabell 29. När förändringar utförs

Proaktivt	Sant		Falskt	
Reaktivt	Sant	Falskt	Sant	Falskt
Antal	2	8	1	0

Tabell 30. Typ av förändring

Deterministisk	Sant		Falskt	
Randomiserad	Sant	Falskt	Sant	Falskt
Antal	3	0	8	0

Tabell 31. Defensiva mål

Villfarelse	Sant		Falskt	
Fördröjning	Sant	Falskt	Sant	Falskt
Antal	2	0	9	0

Tabell 32. Utvärdering av prestanda

Utvärdering	Hypotetisk	Simulering	Empirisk	Teoretisk
Sant	5	5	5	4
Falskt	6	6	6	7

Tabell 33. Utvärdering av användbarhet

Utvärdering	Hypotetisk	Simulering	Empirisk	Teoretisk
Sant	6	1	1	3
Falskt	5	10	10	8

Tabell 34. Utvärdering av effektivitet

Utvärdering	Hypotetisk	Simulering	Empirisk	Teoretisk
Sant	6	5	3	4
Falskt	5	6	8	7

Tabell 35. Frekvens av förändringar

Frekvens	Beskrivs i artikel
Sant	6
Falskt	5

C.6 Kombinerad rörelse

Tabell 36. När förändringar utförs

Proaktivt	Sant		Falskt	
Reaktivt	Sant	Falskt	Sant	Falskt
Antal	1	1	0	0

Tabell 37. Typ av förändring

Deterministisk	Sant		Falskt	
Randomiserad	Sant	Falskt	Sant	Falskt
Antal	0	0	2	0

Tabell 38. Defensiva mål

Villfarelse	Sant		Falskt	
Fördröjning	Sant	Falskt	Sant	Falskt
Antal	0	0	2	0

Tabell 39. Utvärdering av prestanda

Utvärdering	Hypotetisk	Simulering	Empirisk	Teoretisk
Sant	1	0	0	1
Falskt	1	2	2	1

Tabell 40. Utvärdering av användbarhet

Utvärdering	Hypotetisk	Simulering	Empirisk	Teoretisk
Sant	1	0	0	1
Falskt	1	2	2	1

Tabell 41. Utvärdering av effektivitet

Utvärdering	Hypotetisk	Simulering	Empirisk	Teoretisk
Sant	1	1	0	1
Falskt	1	1	2	1

Tabell 42. Frekvens av förändringar

Frekvens	Beskrivs i artikel
Sant	1
Falskt	1

Historien antyder att det i de allra flesta system kan finnas IT-säkerhetsbrister, eller sårbarheter, som exponeras mot antagonister. Moving Target Defense (MTD) är ett samlingsnamn för tekniker som försöker uppnå säkra system genom att ständigt förändra var sårbarheter befinner sig istället för att motarbeta deras existens. Säkerheten för det skyddade systemet ges i och med att attackeraren inte hinner identifiera var sårbarheter för en konfiguration finns innan denna konfiguration byts ut.

Begreppet MTD är relativt nytt men den bakomliggande idén har använts inom IT-området i åtminstone ett decennium. Exempelvis har en variant av MTD funnits för Linux sedan 2001 och en annan variant studerades samma år av DARPA. Den förhållandevis intensiva forskning som idag bedrivs inom området startade dock för ett par år sedan när bland andra Department of Homeland Security riktade dedikerade forskningsmedel till det.

Denna studie kartlägger artiklar som presenterar lösningar baserade på MTD och syftar till att identifiera vilka varianter av MTD som finns, hur användbara dessa varianter är, vilka prestandaegenskaper de har och om de ger mer säkerhet. En litteratursökning identifierade 129 artiklar; av dessa artiklar var 32 relevanta för studiens syfte.

Sex områden för MTD identifierades: 1) rörlig kodtransformering, 2) rörlig minnesallokering, 3) rörliga applikationer, 4) rörliga maskiner, 5) rörliga nätadresser och 6) kombinationer av dessa fem områden. Fokus för de identifierade artiklarna är på introduktion och presentation av skydd snarare än utvärdering av deras prestanda eller hur de erbjuder ökad säkerhet. Sådana egenskaper diskuteras ofta med teori och/eller simuleringsresultat som stöd, men i de allra flesta fall utan empiri och under orealistiska förutsättningar. Detta gör att det är svårt att veta hur effektiva skydden faktiskt är och hur mycket de påverkar användarens upplevelse. Framtida arbete inom området bör fokusera mer på att utvärdera kvaliteten för förespråkade skydd, i synnerhet angående hur användaren påverkas och vilken hotmodell som är realistisk. Empiriska tester under realistiska förutsättningar, exempelvis med hjälp av så kallade cyber ranges, är ett sätt att uppnå mer valida resultat.

FOI är en huvudsakligen uppdragsfinansierad myndighet under Försvarsdepartementet. Kärnverksamheten är forskning, metod- och teknikutveckling till nytta för försvar och säkerhet. Organisationen har cirka 1000 anställda varav ungefär 800 är forskare. Detta gör organisationen till Sveriges största forskningsinstitut. FOI ger kunderna tillgång till ledande expertis inom ett stort antal tillämpningsområden såsom säkerhetspolitiska studier och analyser inom försvar och säkerhet, bedömning av olika typer av hot, system för ledning och hantering av kriser, skydd mot och hantering av farliga ämnen, IT-säkerhet och nya sensorers möjligheter.