



**UNIVERSIDADE DE BRASÍLIA  
FACULDADE DE TECNOLOGIA  
DEPARTAMENTO DE ENGENHARIA ELÉTRICA**

**HONEYHASHES E HONEYTICKETS: DETECÇÃO DE ATAQUES  
PASS-THE-HASH E PASS-THE-TICKET EM REDES DE  
DOMÍNIO ACTIVE DIRECTORY DO WINDOWS**

**FABIO LUIZ DA ROCHA MORAES**

**ORIENTADOR: FLÁVIO ELIAS GOMES DE DEUS  
CO-ORIENTADOR: ROBSON DE OLIVEIRA ALBUQUERQUE**

**DISSERTAÇÃO DE MESTRADO EM ENGENHARIA ELÉTRICA  
ÁREA DE CONCENTRAÇÃO INFORMÁTICA FORENSE E SEGURANÇA  
DA INFORMAÇÃO**

**PUBLICAÇÃO: PPGENE.DM - 636 A/16**

**BRASÍLIA / DF: DEZEMBRO/2016**




**UNIVERSIDADE DE BRASÍLIA  
FACULDADE DE TECNOLOGIA  
DEPARTAMENTO DE ENGENHARIA ELÉTRICA**

**HONEYHASHES E HONEYTICKETS: DETECÇÃO DE ATAQUES  
PASS-THE-HASH E PASS-THE-TICKET EM REDES DE DOMÍNIO  
ACTIVE DIRECTORY DO WINDOWS**

**FÁBIO LUIZ DA ROCHA MORAES**

DISSERTAÇÃO DE MESTRADO PROFISSIONAL SUBMETIDA AO DEPARTAMENTO DE ENGENHARIA ELÉTRICA DA FACULDADE DE TECNOLOGIA DA UNIVERSIDADE DE BRASÍLIA, COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE.

APROVADA POR:



---

FLAVIO ELIAS GOMES DE DEUS, Dr., ENE/UNB  
(ORIENTADOR)



---

RICARDO STACIARINI PUTTINI, Dr., ENE/UNB  
(EXAMINADOR INTERNO)

---



LAERTE PEOTTA DE MELO, BANCO DO BRASIL  
(EXAMINADOR EXTERNO)

Brasília, 16 de Dezembro de 2016.



## FICHA CATALOGRÁFICA

MORAES, FABIO LUIZ DA ROCHA MORAES

Honeyhashes e Honeytickets: Detecção de Ataques Pass-the-Hash e Pass-the-Ticket em Redes de Domínio Active Directory do Windows [Distrito Federal] 2016.

xxiii, 86p., 297 mm (ENE/FT/UnB, Mestre, Engenharia Elétrica, 2016).

Dissertação de Mestrado – Universidade de Brasília, Faculdade de Tecnologia. Departamento de Engenharia Elétrica.

1. Redes Windows      2. Furto de credenciais  
3. Detecção de intrusos   4. Honeytoken

I. ENE/FT/UnB.      II. Título (Série)

## REFERÊNCIA BIBLIOGRÁFICA

MORAES, F. L. R. (2016). Honeyhashes e Honeytickets: Detecção de Ataques Pass-the-Hash e Pass-the-Ticket em Redes de Domínio Active Directory do Windows. Dissertação de Mestrado, Publicação PPGENE.DM - 636 A/16, Departamento de Engenharia Elétrica, Universidade de Brasília, Brasília, DF, 86p.

## CESSÃO DE DIREITOS

NOME DO AUTOR: Fabio Luiz da Rocha Moraes

TÍTULO DA DISSERTAÇÃO: Honeyhashes e Honeytickets: Detecção de Ataques Pass-the-Hash e Pass-the-Ticket em Redes de Domínio Active Directory do Windows.

GRAU/ANO: Mestre/2016.

É concedida à Universidade de Brasília permissão para reproduzir cópias desta Dissertação de Mestrado e para emprestar ou vender tais cópias somente para propósitos acadêmicos e científicos. Do mesmo modo, a Universidade de Brasília tem permissão para divulgar este documento em biblioteca virtual, em formato que permita o acesso via redes de comunicação e a reprodução de cópias, desde que protegida a integridade do conteúdo dessas cópias e proibido o acesso a partes isoladas desse conteúdo. O autor reserva outros direitos de publicação e nenhuma parte deste documento pode ser reproduzida sem a autorização por escrito do autor.



---

Fabio Luiz da Rocha Moraes

SPO, Conjunto A, Lote 23, Complexo da PCDF, Ed. Sede

CEP 70.610-907 – Brasília – DF - Brasil



*Dedico esse trabalho à minha Chiquinha, o ser mais humilde que já encontrei nesse mundo.*





## AGRADECIMENTOS

Ao meu orientador Dr. Flávio Elias Gomes de Deus e co-orientador Robson de Oliveira Albuquerque por me auxiliarem nessa jornada e também ao Dr. Rafael Timóteo de Sousa Júnior, pela apresentação de artigo referente a esse trabalho na Conferência Ibero Americana de Computação Aplicada.

Aos funcionários do Laboratório de Engenharia de Redes de Comunicação – LabRedes da Universidade de Brasília, principalmente ao Luciano do Anjos, pela ajuda e esclarecimentos nos processos administrativos do curso.

Ao meu chefe, Welson Chen, ao Diretor do Instituto de Criminalística (IC), Gustavo Dalton, ao Diretor Adjunto do IC, Juscelino, ao Assessor do Departamento de Polícia Técnica (DPT), Raimundo Cleverlande, ao Diretor do DPT, Paulo Vilarins, por apoiarem a concessão da minha licença capacitação, fundamental para que eu pudesse terminar os trabalhos.

A todos os meus colegas de curso, especialmente ao Daniel Caldas, Marcelo Naves e Vinícius Lima por todo o apoio e companheirismo durante o curso.

Aos meus colegas de seção de trabalho, Marcos Soares, Modestino André, Jean Lima e Marcos Paulo por suportarem uma maior carga de trabalho durante a minha ausência.

À Ludmila, a Bida, pelos anos dedicados no início dessa empreitada, e à Minha Linda Andressa pela compreensão e apoio nas horas de estudo.

Aos meus pais pela vida, à minha família pelos laços e a Deus por todas as coisas.

A todos, os meus sinceros agradecimentos e me perdoem se esqueci alguém.

O presente trabalho foi realizado com o apoio do Departamento Polícia Federal – DPF e Instituto de Criminalística da Polícia Civil do Distrito Federal, com recursos do Programa Nacional de Segurança Pública com Cidadania – PRONASCI, do Ministério da Justiça.



## **RESUMO**

### **HONEYHASHES E HONEYTICKETS: DETECÇÃO DE ATAQUES PASS-THE-HASH E PASS-THE-TICKET EM REDES DE DOMÍNIO ACTIVE DIRECTORY DO WINDOWS**

**Autor: Fabio Luiz da Rocha Moraes**

**Orientador: Flávio Elias Gomes de Deus**

**Programa de Pós-graduação em Engenharia Elétrica**

**Brasília, dezembro de 2016**

O furto de credenciais é considerado ser uma peça em uma cadeia de eventos que englobam ataques a redes de computadores e sistemas operacionais, sendo a evasão de dados ou incidentes parte de um escopo maior no processo de verificação e resposta a incidentes. Apesar de ataques do tipo Pass-the-Hash serem conhecidos no ambiente de Redes Windows há algum tempo, a detecção desses tipos de técnicas ainda constitui um desafio em ambientes onde o processo de autenticação conta com sistemas distribuídos em redes heterogêneas e amplas. De forma a auxiliar no combate e detalhamento deste tipo de problema de segurança em redes, este trabalho estudou as medidas protetivas contra esses ataques e três linhas de detecção. As linhas de detecção foram classificadas em detecção de eventos, detecção de anomalias e detecção de honeytokens. São propostos três elementos que podem auxiliar no confronto ao problema. O primeiro elemento é uma Kill Chain para ataques Pass-the-Hash e Pass-the-Ticket que visa mostrar e explicar a anatomia desses ataques. O segundo é uma classificação para as abordagens de tratamento desses ataques como medida para organizar as diferentes formas como o problema é encarado. O último trata-se de uma arquitetura para servir de base para a implementação de uma ferramenta de detecção. Esse terceiro componente possui fim de ajudar no processo de detecção de intrusão em redes com as características mencionadas.



## **ABSTRACT**

### **HONEYHASHES & HONEYTICKETS: DETECTING PASS-THE-HASH AND PASS-THE-TICKET ATTACKS IN WINDOWS ACTIVE DIRECTORY DOMAIN NETWORKS**

**Author: Fabio Luiz da Rocha Moraes**

**Supervisor: Flávio Elias Gomes de Deus**

**Programa de Pós-graduação em Engenharia Elétrica**

**Brasília, december of 2016**

Credential theft is considered to be a piece in a chain of events that cover attacks to computer networks and operating systems, being data leaks or incidents part of a bigger scope in the process of verification and response to incidents. In spite of Pass-the-Hash and Pass-the-Ticket attacks are long known in the Windows Network environment, the detection of such techniques still constitutes a challenge in environments where the authentication process counts with distributed systems in wide and heterogeneous networks. In order to aid the combating and detailing with this kind of network security problem, this work studied the protective measures against these attacks and three lines of detection. The lines of detection were classified in event detection, anomaly detection and honeytoken detection. Three elements that can help in the confrontation to this matter were proposed. The first element is a Kill Chain for Pass-the-Hash and Pass-the-Ticket attacks, that aims to show and explain the anatomy of these attacks. The second is a classification to the approaches in dealing with the issue as a mean to organize the different ways the problem is treated. The last is about an architecture to serve as a base for the implementation of a detection tool. This third component has an end to help the process of intrusion detection in networks with the mentioned characteristics.



# SUMÁRIO

1. INTRODUÇÃO.....	1
1.1. MOTIVAÇÃO.....	2
1.2. DECLARAÇÃO DO PROBLEMA.....	4
1.3. OBJETIVOS.....	5
1.4. LIMITAÇÕES E PREMISSAS.....	5
1.5. ORGANIZAÇÃO DO TRABALHO.....	6
2. REVISÃO DA LITERATURA.....	7
2.1. REDES DE DOMÍNIO DO WINDOWS E ACTIVE DIRECTORY.....	7
2.2. PROTOCOLO KERBEROS.....	8
2.2.1. Componentes e Funcionamento do Protocolo.....	8
2.2.2. Troca de mensagens.....	10
2.2.3. Processo de autenticação.....	11
2.2.4. Tipos de Criptografia Suportados.....	15
2.3. ARMAZENAMENTO DE CREDENCIAIS NO WINDOWS.....	16
2.4. ATAQUES PASS-THE-HASH E PASS-THE-TICKET.....	17
2.4.1. Overpass-the-Hash (Pass-the-Key).....	18
2.4.2. Pass-the-Ticket.....	19
2.4.3. Silver Tickets.....	20
2.4.4. Golden Tickets.....	21
2.5. FORMAS DE COMBATE A ATAQUES PTH&T.....	22
2.5.1. Medidas de Prevenção.....	22
2.5.2. Inovações no Sistema Operacional.....	23
2.5.3. Detecção de Eventos.....	24
2.5.4. Detecção de Anomalia.....	25
2.5.5. Detecção de Honeytokens.....	26
3. ANÁLISE DE UMA ABORDAGEM DE DETECÇÃO COM HONEYTOKENS.....	28
3.1. MODELO ADVERSARIAL.....	28
3.2. ESTUDO DOS PROBLEMAS USANDO O DCEPT.....	30
3.3. ARQUITETURA E CONFIGURAÇÃO DE DETECÇÃO.....	31
3.4. VALIDAÇÃO DO AMBIENTE.....	34

3.5. DISCUSSÃO E ANÁLISE.....	40
4. PROPOSTAS DE MELHORIAS.....	43
4.1. PROPOSTA DE ARQUITETURA PARA UMA FERRAMENTA DE DETECÇÃO DE HONEYTOKENS.....	43
4.1.1. Injeção de honeyhashes e honeytickets.....	43
4.1.2. Uso de Vários Honeytokens.....	44
4.1.3. Detecção de Golden Tickets.....	44
4.1.4. Suporte a Múltiplos Tipos de Criptografia.....	44
4.1.5. Continuação do Protocolo em Caso de Detecção.....	44
4.1.6. Resumo das Funcionalidades e Arquitetura.....	45
4.2. VIABILIDADE TECNOLÓGICA DA ARQUITETURA.....	47
4.3. DESCRIÇÃO DO CENÁRIO DE TESTES.....	47
4.4. REALIZAÇÃO DOS TESTES.....	47
4.4.1. Teste de Injeção de Honeyhash e Honeyticket.....	48
4.4.2. Teste de Encaminhamento de Tráfego.....	49
4.4.3. Teste de Uso do Tipo de Criptografia Correto.....	54
4.5. IMPLEMENTAÇÕES DE INTERESSE COM FERRAMENTAS OPEN SOURCE.....	58
4.6. DISCUSSÃO.....	59
4.7. PROPOSTAS COMPLEMENTARES.....	60
4.7.1. Proposta de Kill Chain para Ataques PtH&T.....	60
4.7.2. Proposta de Classificação para as Abordagens de Tratamento aos Ataques.....	61
5. CONCLUSÕES E TRABALHOS FUTUROS.....	64
5.1. PUBLICAÇÃO DE PESQUISA.....	65
5.2. RECOMENDAÇÕES PARA TRABALHOS FUTUROS.....	65
REFERÊNCIAS BIBLIOGRÁFICAS.....	67
A – Playbook Ansible.....	73
B – Patch de Mudanças no Código da Ferramenta DCEPT.....	77
C – Ferramentas Utilizadas.....	83



## LISTA DE TABELAS

Tabela 2.1: Chaves de longo prazo e chaves de sessão usados no Protocolo Kerberos.....	9
Tabela 2.2: Tíquetes usados no Protocolo Kerberos.....	9
Tabela 2.3: Tipos de criptografia constantes na RFC 3961.....	15
Tabela 2.4: Eventos de interesse que podem ser usados na coleta e análise em busca de ataques.....	24
Tabela 3.1: Informações sobre as máquinas virtuais do experimento.....	31
Tabela 3.2: Combinações de Realm e Salt testadas.....	38
Tabela 3.3: Possíveis valores para a propriedade msDC-SupportedEncryptionTypes.....	39
Tabela 3.4: Resumo das observações sobre a ferramenta DCEPT.....	42
Tabela 4.1: Comparativo entre a proposta e a ferramenta DCEPT.....	46
Tabela 4.2: Máquinas virtuais utilizadas nos testes.....	47
Tabela 4.3: Principais protocolos trocados entre a estação de trabalho e o DC.....	51

## LISTA DE FIGURAS

Figura 1.1: Fases da Intrusion Kill Chain para segurança da informação.....	3
Figura 2.1: Uso de sistemas operacionais em computadores no segmento desktop.....	7
Figura 2.2: Composição entre o Active Directory, KDC, AS e TGS.....	9
Figura 2.3: Troca de mensagens do Protocolo Kerberos conforme definido na RFC 4120.....	10
Figura 2.4: Detalhamento das mensagens trocadas entre o cliente e os serviços no Protocolo Kerberos.....	11
Figura 2.5: Chaves tíquetes e suas respectivas localizações nos participantes do Kerberos.....	11
Figura 2.6: Detalhes das trocas de mensagens entre o cliente e o Serviço AS, sem dados de pré-autenticação.....	12
Figura 2.7: Detalhes das trocas de mensagens entre o cliente e o Serviço AS, com dados de pré-autenticação.....	13
Figura 2.8: Detalhes das trocas de mensagens entre o cliente e o Serviço TGS.....	14
Figura 2.9: Detalhes das trocas de mensagens entre o cliente e o Serviço de Aplicação.....	15
Figura 2.10: Ataque Overpass-the-Hash (Pass-the-Ticket) com o uso de uma chave obtida.....	18
Figura 2.11: Pass-the-Ticket com o uso de um tíquete TGT obtido.....	19
Figura 2.12: Pass-the-Ticket com o uso de um tíquete de serviço obtido.....	19
Figura 2.13: Silver Ticket forjado com o uso de uma chave de serviço obtida.....	20
Figura 2.14: Golden Ticket forjado a partir da chave do TGS (conta KRBTGT).....	21
Figura 3.1: Exemplo de um ambiente de Rede Windows.....	29
Figura 3.2: Funcionamento da ferramenta DCEPT.....	31
Figura 3.3: Servidor de geração de honeytokens e sniffer inicializados. O servidor de geração responde ao pedido de uma honeyword pelo cliente.....	32
Figura 3.4: Agente conecta ao serviço, recebe um honeytoken, coloca em memória e dorme por 24 horas.....	33
Figura 3.5: Uso básico da ferramenta Mimikatz para extração de credenciais em memória.....	35
Figura 3.6: Honeyword extraído da memória com a ferramenta Mimikatz, em destaque pelo pontilhado.....	35
Figura 3.7: Erro "no password hashes loaded".....	36
Figura 3.8: Chamada ao John com o formato KRB5PA-SHA1 esperado para o arquivo de hashes.....	36
Figura 3.9: timestamp cifrado com o encryption type RC4-HMAC (23) na requisição AS-REQ enviada pelo cliente.....	37
Figura 3.10: Erro "no password hashes loaded" resolvido.....	37

Figura 3.11: Editando a propriedade msDC-SupportedEncryptionTypes com o valor 0x10 para garantir o uso do AES256.....	39
Figura 3.12: Erro KRB5KDC_ERR_ETYPE_NOSUPP.....	40
Figura 4.1: Arquitetura proposta para uma ferramenta de detecção de honeytokens.....	46
Figura 4.2: Simulação de um ataque PtH.....	48
Figura 4.3: Captura de tráfego DNS com queries do tipo SRV.....	50
Figura 4.4: Seção modificada do arquivo dnsmasq.conf.....	50
Figura 4.5: Arquivo de hosts do servidor Ubuntu.....	51
Figura 4.6: Processos do utilitário Socat usado para o encaminhamento.....	51
Figura 4.7: Tráfego Kerberos passando pelo servidor Ubuntu.....	52
Figura 4.8: Tráfego LDAP trocado diretamente entre a estação de trabalho e o DC.....	53
Figura 4.9: Tráfego CLDAP passando pelo servidor Ubuntu.....	53
Figura 4.10: Módulo PyShark usado para capturar o tráfego Kerberos.....	54
Figura 4.11: Uso do depurador pdb para Python.....	55
Figura 4.12: Parte das alterações realizadas no código da ferramenta DCEPT.....	57
Figura 4.13: Inserção manual de honeytoken na base de dados.....	58
Figura 4.14: Detecção de honeytoken após alteração no código.....	58
Figura 4.15: Fases da Pass-the-Hash&Ticket Kill Chain.....	60
Figura 4.16: Proposta de classificação para as abordagens de combate a ataques PtH&T.....	62



## LISTA DE SÍMBOLOS, NOMENCLATURA E ABREVIACÕES

AD	Active Directory
AD DS	Active Directory Domain Services
ADSI	Active Directory Service Interfaces
AP	Application Service
API	Application Programming Interface
AS	Authentication Service
BDC	Backup Domain Controller
CLDAP	Connectionless LDAP
DC	Domain Controller
DCEPT	Domain Controller Enticing Password Tripwire
DHCP	Dynamic Host Configuration Protocol
DNS	Domain Name System
IDS	Intrusion Detection System
IP	Internet Protocol
JtR	John the Ripper
KDC	Kerberos Distribution Center
K <sub>TGS</sub>	Key TGS
K <sub>U</sub>	Key User
K <sub>S</sub>	Key Service
LDAP	Lightweight Directory Access Protocol
PC	Personal Computer
PDC	Primary Domain Controller
PtH	Pass-the-Hash
PtH&T	Pass-the-Hash e Pass-the-Ticket
PtH&T-KC	Pass-the-Hash&Ticket-Kill Chain
PtT	Pass-the-Ticket
SO	Sistema Operacional
SSO	Single Sign-On
ST	Service Ticket
TCP	Transmission Control Protocol
TGS	Ticket-Granting Service
TGT	Ticket-Granting Ticket

UDP	User Datagram Protocol
VM	Virtual Machine
VPN	Virtual Private Network
WEL	Windows Event Logs



# 1. INTRODUÇÃO

Considere o seguinte cenário. Um atacante explora uma vulnerabilidade em um computador participante de uma Rede de Domínio Windows e obtém privilégios de sistema nesse único computador. O intruso tem então acesso às áreas protegidas da memória desse computador. Em uma dessas áreas estão armazenadas credenciais de usuários na forma de *hashes* de senhas (chaves) e tíquetes. Ele extrai essas informações da memória do computador comprometido e passa a se *mover lateralmente* utilizando os *hashes* e tíquetes como credenciais para obter acesso em outros computadores participantes desse domínio. Com esse acesso ele passa a explorar mais vulnerabilidades nos novos sistemas, escalando privilégios e repetindo a mesma dança por várias vezes, comprometendo cada vez mais computadores no ambiente de rede. Com credenciais de contas de serviços, ele passa a ser capaz de gerar *Silver Tickets*, tendo acesso a sistemas com serviços na rede, numa hierarquia de maior valor, os servidores. Em dado momento o atacante consegue comprometer o Controlador de Domínio da rede e acessar o *hash* da senha de uma conta de domínio especial, chamada KRBTGT. Comprometendo essa conta o invasor agora é capaz de obter acesso a quaisquer recursos do domínio, por meio da criação de *Golden Tickets*. Essa sequência de acontecimentos é um exemplo de como um único computador pode dar início a uma série de eventos que levam ao comprometimento de toda uma Rede de Domínio Windows.

Neste trabalho é abordada a questão de ataques *Pass-the-Hash* (PtH) e ataques *Pass-the-Ticket* (PtT) em redes de computadores do Windows. Um dos recursos de uma rede de uma Rede Windows é tentar prover uma maior segurança aos seus usuários. Ataques desse tipo são usados conforme a história introdutória. Um invasor tem acesso à rede de computadores por meio de um ponto fraco. Pode ser uma área menos sensível e que exija menos atenção dos administradores. Porém, o atacante, por meio desse ponto menos crítico, tem acesso a novas credenciais de usuários armazenadas no computador comprometido. Ele passa a testar essas credenciais em outros computadores da rede, buscando por novos acessos. A isso dá-se o nome de movimento lateral. Ataques PtH e PtT se encaixam bem para esse tipo de finalidade. Esses ataques são conhecidos desde a década de 90 e até hoje a empresa Microsoft lança novidades em seus sistemas operacionais tentando dificultar esses ataques. Para endereçar esse problema, várias medidas são sugeridas. É possível citar o monitoramento da rede e implantação de sistemas de detecção, dentre essas sugestões de medidas especiais de segurança a serem feitas pelos administradores da rede.

Será visto nesse trabalho que a detecção é uma tarefa de certa forma árdua. A causa disso se deve à similaridade entre uma ação lícita, isto é, um usuário credenciado se autenticando e usando um



serviço na rede, e uma ação ilícita, um invasor se valendo das credenciais furtadas desse usuário para ter acesso a recursos. Será mostrado que a questão da detecção tem íntima relação com o Protocolo Kerberos, que é o principal protocolo de autenticação nesse tipo de rede.

Para contribuir no combate a esses ataques foi feita uma revisão da literatura e foi focado em um tipo de detecção específica, que fora denominada detecção por *honeytokens*. Essa abordagem implanta objetos falsos, parecidos com verdadeiros, no intuito de atrair a atenção de um atacante, servindo como uma espécie de isca. Essa abordagem parece promissora, porém pouco explorada para a detecção desses ataques. Foi possível encontrar apenas uma ferramenta que utiliza essa técnica. Por isso, essa ferramenta foi estudada como forma de compreender esse tipo de abordagem. Daquilo que foi aprendido e observado surgiram ideias que viraram propostas de melhoria para a implementação de uma ferramenta mais abrangente. Não se chegou a começar uma implementação de tal ferramenta, mas foram realizados alguns testes para confirmar a possibilidade de implementação. Além disso, a visão ampla adquirida com o estudo fez com que fossem propostos mais dois elementos que podem contribuir para a compreensão e combate a esses tipos de ataques. O primeiro elemento visa explicar e educar profissionais da área sobre as fases desses ataques, permitindo o foco em pontos específicos para a sua quebra. O segundo elemento tem a intenção de organizar as diversas condutas tomadas contra esses ataques.

## 1.1. MOTIVAÇÃO

Por ser um tipo de rede muito usada por entidades públicas e privadas, as Redes Windows são alvos constantes de *hackers*. Seus objetivos podem ser os mais diferentes como lucro, espionagem corporativa ou industrial, espionagem governamental, ativismo (hacktivismo) político, ativismo ideológico, curiosidade, vandalismo, terrorismo, etc. Nesse sentido faz-se mister que o sistema seja seguro de modo a não comprometer informações. O comprometimento de informações pode gerar perdas e danos de proporções diversas. Para atingir seus objetivos, atacante mais preparados executam uma série de ações preparatórias e de execução. Essas ações chegam a ser comparadas com operações militares e deram origem a termos oriundos daquele meio. Um termo comumente usado no meio é a *Cyber Kill Chain*, que tenta mostrar a anatomia de um ataque. Inicialmente proposta por Hutchins, Cloppert e Amin (2011) como *Intrusion Kill Chain*, é defendido que ataques podem ocorrer em fases e que podem ser parados a partir da implantação de medidas de controle em cada um desses estágios. A Figura 1.1 mostra os estágios da Kill Chain, conforme proposta por Hutchins, Cloppert e Amin. Um exemplo de uso da Kill Chain é o documento elaborado pelo Comitê em Comércio, Ciência e Transporte (*Committee on Commerce, Science and Transportation*) do Senado dos Estados Unidos (2014) que analisa a evasão de dados da empresa Target, uma das

maiores empresas de venda no varejo nos Estados Unidos. Na ocasião, atacantes usaram credenciais furtadas de uma empresa de aquecimento, ventilação e ar-condicionado (AVAC), chamada Fazio Mechanical Services, que possuía uma fraca segurança da informação e que tinha acesso à rede interna da Target, para ter acesso à rede da empresa. A partir dali os atacantes se moveram lateralmente de áreas menos sensíveis para áreas com dados de usuários. Além disso, aparentemente a companhia Target falhou ao responder aos múltiplos avisos de seus sistemas de detecção de intrusos (*Intrusion Detection System – IDS*). Embora não tão claro, uma das ações mais comuns dentro de uma cadeia de eventos de ataque é o furto de credenciais.

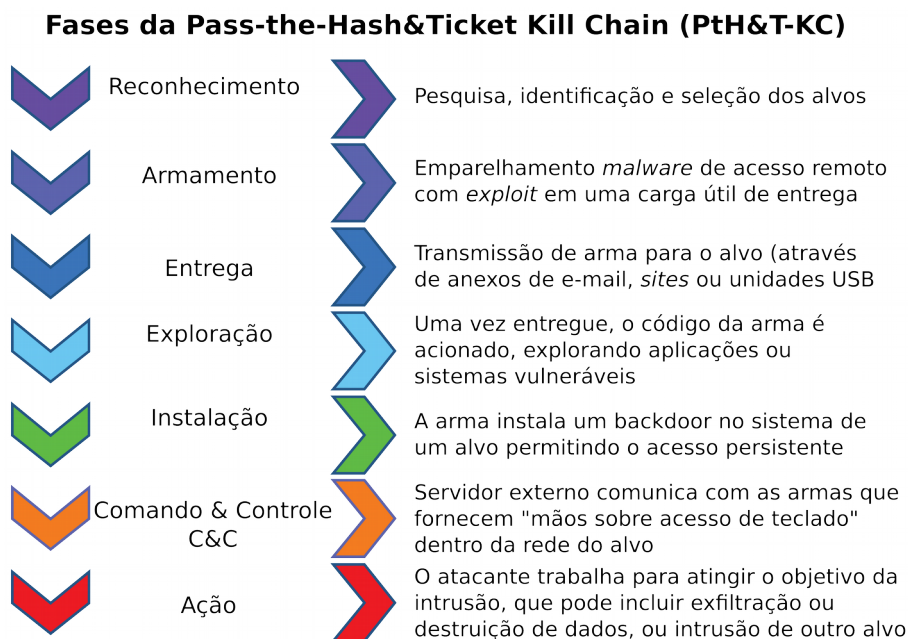


Figura 1.1: Fases da *Intrusion Kill Chain* para segurança da informação.

Fonte: modificado a partir de figura do *site* Wikimedia Commons<sup>1</sup>.

Segundo a companhia de notícias USA TODAY (ERIN KELLY, 2015), hackers chineses usaram o furto de credenciais de uma contratada do Escritório de Gerenciamento de Pessoal (*Office of Personnel Management – OPM*) para consumir um dos maiores ataques ao Governo Norte-Americano em 2015, que levou ao vazamento de pelo menos 4 milhões de registros de informações sobre empregados federais. Este é um caso dentre diversos outros. Até novembro de 2016 já houve 858 violações de dados, segundo o ITRC (2016)– *Identity Theft Resource Center*, expondo quase 30 milhões de registros. De acordo com a empresa Verizon (2016), no seu relatório de violações de dados, 63% de violação de dados confirmadas envolveram a obtenção de senhas fracas, padrão ou usurpadas. O furto de credenciais é usado em muitos padrões de incidentes, de infecção de malware

<sup>1</sup> Disponível em <[https://upload.wikimedia.org/wikipedia/commons/1/1d/Intrusion\\_Kill\\_Chain\\_-\\_v2.png](https://upload.wikimedia.org/wikipedia/commons/1/1d/Intrusion_Kill_Chain_-_v2.png)>. Acesso em dez. 2016.

a ataques altamente direcionados como os coordenados por uma Ameaça Persistente Avançada, em inglês *Advanced Persistent Threat – APT* (FOSTER, 2014).

Conforme o exposto, o furto de credenciais possui papel importante no âmbito da segurança da informação e na prevenção de ataques. Por se tratar de um sistema operacional (SO) muito utilizado, o Windows é constante alvo de ataques. Nesse sentido, o estudo, pesquisa e elaboração de medidas que auxiliem na detecção de intrusos a partir da detecção do furto de credenciais em Redes Windows possui grande relevância.

## 1.2. DECLARAÇÃO DO PROBLEMA

Uma Rede de Domínio Windows é um contexto de segurança provido por software fornecendo funcionalidades como *Single Sign-On – SSO* – e autenticação mútua. A autenticação é provida por uma entidade central e é utilizada por todos os computadores associados a essa rede em nível de aplicação. Essa entidade central é conhecida como Controlador de Domínio (*Domain Controller – DC*) fornecido pela suíte de software *Active Directory – AD* (MICROSOFT CORPORATION, 1999).

O protocolo escolhido como padrão para prover autenticação mútua foi o Kerberos (NEUMAN *et al.*, 2005), cujo princípio é formado pelo protocolo Needham-Schroeder (NEEDHAM; SCHROEDER, 1978). A base do protocolo Kerberos é usar uma chave simétrica, conhecida por ambas as partes na comunicação (senha do usuário), para estabelecer uma chave de sessão e autenticar ambas as partes. A chave de sessão é utilizada para cifrar mensagens trocadas. Isto é feito por meio de um serviço no controlador de domínio chamado *Kerberos Distribution Center – KDC* – (Centro de Distribuição Kerberos), que também é responsável pela distribuição de tíquetes para permitir que usuários acessem recursos e serviços na rede.

Domínios do Active Directory do Windows têm sido marcados por anos por ataques do tipo *Pass-the-Hash (PtH)* e *Pass-the-Ticket (PtT)*. Ataques PtH têm sua existência conhecida pelo menos desde 1997 (BASHAR, 2010; SECURITYFOCUS, 1997). PtH são técnicas de ataque onde um atacante utiliza um *hash* de senha como um equivalente da senha para personificar o usuário portador da senha e acessar recursos permitidos a ele.

De forma similar ao PtH, na técnica PtT um atacante também personifica um usuário, mas ao invés de usar um *hash* de senha o atacante utiliza um tíquete do Kerberos, gerado pelo KDC ou forjado. O tíquete pode ser válido por um período de tempo preestabelecido, como 10 horas (validade padrão para tíquetes de usuários e serviços) (MICROSOFT CORPORATION, 2002), ou o atacante pode ser capaz de gerar tíquetes para quaisquer serviços (caso conhecido como *Golden*

*Ticket*), dependendo do *hash* obtido ou nível de infiltração (EXORCYST, 2014; PILKINGTON, 2014).

Como um problema intrínseco, não é possível evitar completamente estes tipos de ataques uma vez que o atacante tenha obtido acesso a *hashes* ou tíquetes válidos. A resposta até então tem sido minimizar a possibilidade de um atacante ganhar privilégios administrativos em computadores participantes de um domínio, diminuindo assim a possibilidade de acesso a *hashes* e tíquetes, e limitar o escopo de uso de credenciais (JUNGLES *et al.*, 2012).

Técnicas recentes lidam com a detecção de intrusos buscando atividades anômalas na rede, principalmente pela análise de Registros de Eventos do Windows (*Windows Event Logs – WEL*) (HSIEH *et al.*, 2015; SAPEGIN *et al.*, 2015; YU, 2015). A suposição é de que as ações de um intruso em uma rede serão diferentes das de um usuário regular e então essas ações são passíveis de serem examinadas e descobertas, gerando alarmes para administradores. Em uma outra direção está o uso de *honeypots* para se detectar intrusos (SPITZNER, 2002). Quando comparados com a detecção de anomalias, *honeypots* são mais simples e diretos do que aquelas, que podem gerar falsos positivos e vice-versa. Com uma abordagem de *honeypot* sabe-se que o seu uso não fora aprovado. Além disso, ambas as técnicas devem ser usadas em conjunto para diminuir o problema, afinal, um *honeypot* não seria suficiente para, por exemplo, detectar o furto de uma credencial válida.

### 1.3. OBJETIVOS

Este trabalho tem por objetivo geral o estudo de técnicas de intrusão baseada em ataques do tipo PtH&T e as respectivas técnicas de detecção de tais ataques. Como forma de atingir esse objetivo geral, foi definido o seguinte conjunto de objetivos específicos:

1. Montar um ambiente para estudo e análise da ferramenta DCEPT;
2. Mostrar como uma detecção de *honeypot* pode combater o furto de credenciais;
3. Elucidar os problemas existentes em uma abordagem de detecção de *honeypots*; e
4. Propor melhorias para uma para a detecção de *honeypots*.

### 1.4. LIMITAÇÕES E PREMISSAS

Uma das limitações do trabalho é que não se teve acesso a um ambiente de produção onde a ferramenta DCEPT poderia ter sido instalada e configurada para que, a partir daí, fossem obtidas estatísticas referentes a acessos, tráfegos e demandas de recursos de processamento. Como maneira

de superar essa limitação foi desenvolvido uma arquitetura de simulação composta por todos os elementos básicos em uma Rede Windows com o Active Directory.

É assumido um cenário de pós-exploração (*post-exploitation*) em que um atacante já obteve algum tipo de privilégio em uma estação de trabalho cliente de um Domínio Windows. Portanto, não é utilizada nenhuma técnica de exploração de vulnerabilidade para se ter acesso a despejos (*dumps*) de memória ou ações que necessitam de privilégios de administrador.

## **1.5. ORGANIZAÇÃO DO TRABALHO**

Esta dissertação é composta por 5 capítulos. O Capítulo 1 apresenta a introdução ao tema desta pesquisa. No Capítulo 2 passa-se pela revisão da literatura, onde é feita uma revisão sobre os principais conceitos e tecnologias envolvidos na compreensão deste trabalho. Alguns detalhes mais pertinentes à detecção de ataques PtH&T do Protocolo Kerberos são estudados. São vistos alguns locais onde *hashes* e tíquetes podem ser obtidos por atacantes. Nesse capítulo são vistos os diferentes tipos de ataques PtH&T, bem como as medidas de combate a esses ataques. No Capítulo 3 é feito o experimento que visa ajudar a atingir o objetivo de entender como *honeytokens* podem ser usados para a detecção. No Capítulo 4 são feitas propostas de melhorias com base nos resultados observados no capítulo anterior. No Capítulo 5 são expostas as conclusões a que se chegou a partir da revisão da literatura e dos resultados.

## 2. REVISÃO DA LITERATURA

Para o correto entendimento de ataques do tipo PtH&T é importante revisar alguns conceitos na literatura, entre eles, o funcionamento básico do protocolo Kerberos e a troca de mensagens correspondentes. Também é fundamental o conhecimento das diferentes ações que são tomadas para se lidar com o problema, no âmbito da mitigação e detecção de tais ataques.

### 2.1. REDES DE DOMÍNIO DO WINDOWS E ACTIVE DIRECTORY

O Windows, em suas mais variadas versões, ainda hoje é um dos sistemas operacionais mais utilizados no mundo para computadores pessoais (PC), incluindo *desktops* e *laptops*, em diferentes fontes de pesquisa. A Figura 2.1 mostra o uso de sistemas operacionais no mundo no período de janeiro de 2015 a novembro de 2016, segundo o *site* StatCounter (2016). Além disso, ele possui edições tanto para utilização em casa quanto para utilização em ambientes de trabalho e também por profissionais liberais. Redes de Domínio do Windows com o Active Directory são amplamente utilizadas em redes internas corporativas. Essas redes possuem os recursos necessários para prover funcionalidades para administradores da rede, facilitando o seu uso e adoção. São providas funcionalidades como gerenciamento de configuração, instalação automatizada de aplicativos, criação de perfis para usuários e adoção de medidas de segurança. Alguns exemplos de medidas de segurança são a implantação de autenticação e autorização, identificando os utilizadores dos recursos da rede e limitando o acesso àquilo que os usuários devem ter, segundo as políticas estabelecidas pela direção da organização. Esse dois controles fazem parte das principais funcionalidades de um Domínio Windows.

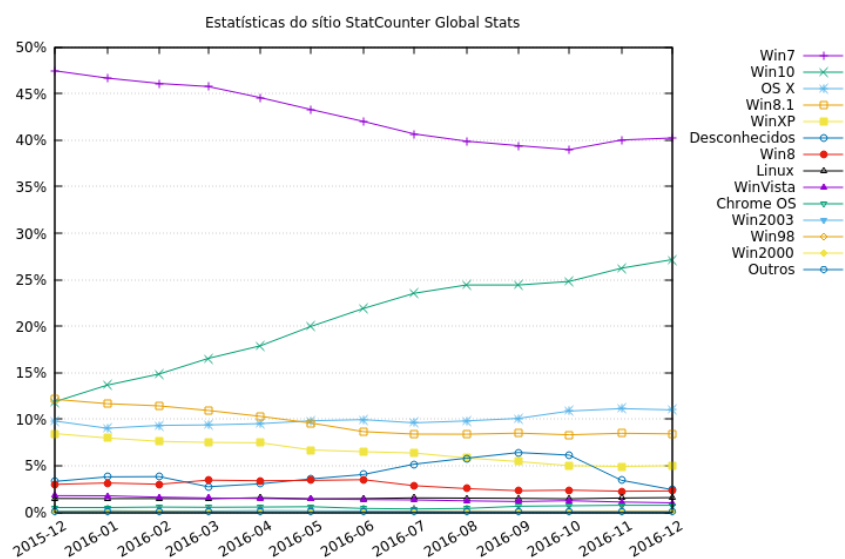


Figura 2.1: Uso de sistemas operacionais em computadores no segmento *desktop*.

Um Domínio Windows é uma rede de computadores no nível de aplicação. Descrevendo de maneira ampla, algumas das principais funcionalidades de um Domínio Windows são prover serviços de diretórios e autenticação única (*Single Sign-On – SSO*) (MICROSOFT CORPORATION, 1999). O domínio é gerenciado por uma suíte de software chamada Active Directory – AD – que é instalada em um servidor central chamado de Controlador de Domínio (*Domain Controller – DC*). Podem existir outros controladores de domínio, conhecidos por Controladores de Domínio de Backup (*Backup Domain Controllers – BDC*). O AD fornece o serviço de diretórios e mantém uma base de dados de usuários, computadores, recipientes, grupos, políticas e vários objetos na rede. Para autenticar usuários o AD usa a implementação da Microsoft para o protocolo Kerberos. Essa implementação possui alguma variação da especificação do protocolo e é conhecida como MS-KILE (MICROSOFT CORPORATION, 2016).

## **2.2. PROTOCOLO KERBEROS**

O Serviço de Autenticação de Rede Kerberos (versão 5) é descrito pela RFC 4120 (NEUMAN *et al.*, 2005). A nomenclatura entre chamar o Kerberos de serviço ou protocolo se confunde na literatura aparentemente e é usada no texto de forma intercambiável. Ele serve para prover autenticação mútua entre duas partes em uma conversa e comunicação segura em um meio não seguro.

### **2.2.1. Componentes e Funcionamento do Protocolo**

É usado o conhecimento prévio de uma chave simétrica (senha) como base para identificar participantes e envolve a participação de um terceiro de confiança. Além disso é previsto suporte para múltiplos tipos de criptografia. A partir da senha e dos múltiplos tipos de criptografia são derivadas chaves de longo prazo (*hashes*), que serão usadas como entradas para algoritmos criptográficos (*encryption types*). O Protocolo Kerberos também evita o envio da senha pela rede e limita o seu uso, por meio do emprego de chaves de sessão (*session keys*), a serem usadas nas trocas de mensagens. A Tabela 2.1 mostra as diferentes chaves usadas no protocolo.

Brevemente, um cliente se autentica em um Serviço de Autenticação (*Authentication Service – AS*), que é um dos dois serviços que compõem o Serviço chamado de Centro de Distribuição Kerberos (*Kerberos Distribution Center – KDC*), que é o terceiro de confiança entre as partes que desejam autenticar-se. O outro serviço componente do KDC é o Serviço de Concessão de Tíquetes (*Ticket Granting Service – TGS*), conforme mostra a Figura 2.2.

Tabela 2.1: Chaves de longo prazo e chaves de sessão usados no Protocolo Kerberos

Chave	Descrição	Local
$K_U$	Chave do Usuário (User key, derivada da senha)	Cliente e KDC
$K_{TGS}$	Chave do TGS (TGS key)	KDC
$K_S$	Chave do Serviço (Service key )	Aplicação e KDC
$SK_{TGS}$	Chave de Sessão do TGS (TGS Session Key)	Cliente e KDC <sup>2</sup>
$SK_S$	Chave de Sessão do Serviço (Service Session Key)	Cliente e Aplicação <sup>2</sup>

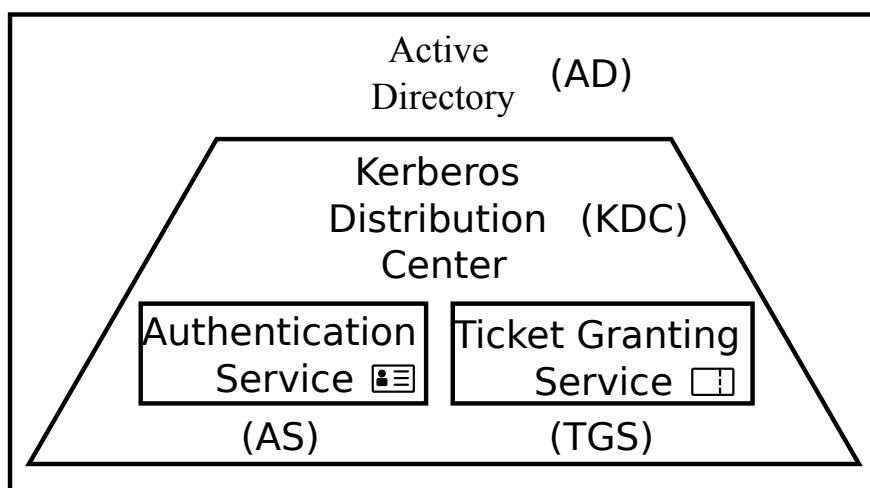


Figura 2.2: Composição entre o Active Directory, KDC, AS e TGS.

Após a autenticação o cliente recebe um Tíquete de Concessão de Tíquetes (*Ticket Granting Ticket – TGT*), que será usado para requisitar ao TGS outros tíquetes, conhecidos por Tíquetes de Serviço (*Service Tickets – ST*). A Tabela 2.2 apresenta esses tíquetes e seus respectivos emissores. Ressalta-se que tíquetes de serviço também são conhecidos na nomenclatura como TGS, o que pode causar confusão com o Serviço de Concessão de Tíquetes, TGS. Esses, por sua vez, serão usados para acessar recursos na rede, como um compartilhamento ou uma impressora. A implementação da Microsoft, também conhecida na literatura como MS-KILE, inclui extensões ao protocolo para prover informações de autorização adicionais como participação em grupos, informação de logon interativo e níveis de integridade (MICROSOFT CORPORATION, 2016a).

Tabela 2.2: Tíquetes usados no Protocolo Kerberos

Tíquete	Descrição	Emissor
TGT	Tíquete de Concessão de Tíquetes (Ticket-Granting Ticket)	AS
ST	Tíquete de Serviço (Service Ticket ou TGS <sup>3</sup> )	TGS

<sup>2</sup> Obtidas pelo TGS (KDC) e aplicação por meio de decifragem do tíquete pertinente.

<sup>3</sup> Os tíquetes de serviço também são conhecidos como TGS, o que pode gerar confusão com o serviço TGS do KDC.



### 2.2.2. Troca de mensagens

Resumidamente o Kerberos troca mensagens na rede para fins de autenticação. Os principais conjuntos de trocas de mensagens do cliente são de três tipos:

1. Com o Serviço de Autenticação (AS);
2. Com o Serviço de Concessão de Tíquetes (TGS); e
3. Com algum dos serviços de aplicação (AP).

Esse grupo de troca de mensagens é apresentado na Figura 2.3.

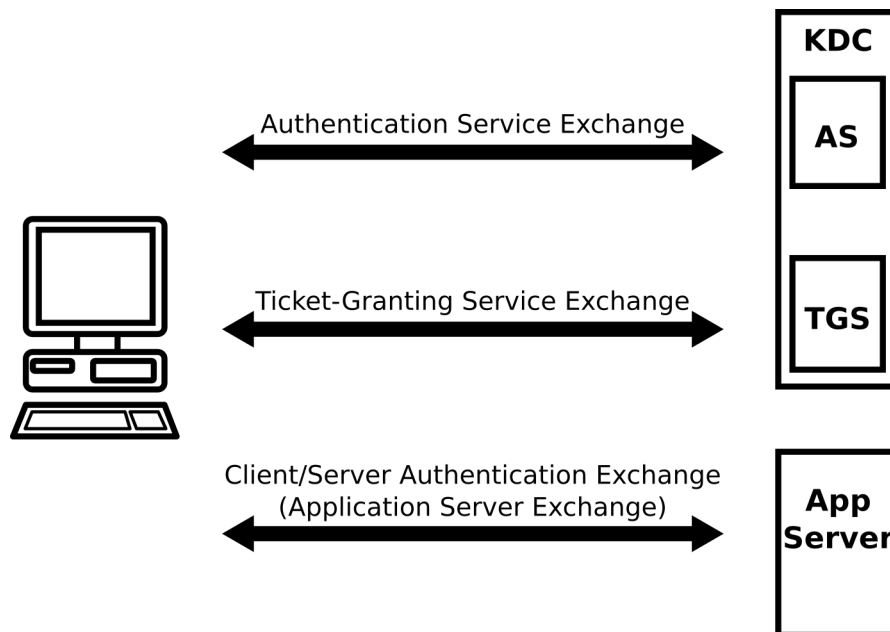


Figura 2.3: Troca de mensagens do Protocolo Kerberos conforme definido na RFC 4120.

Sem se levar em consideração as possíveis mensagens de erro, no tocante ao protocolo Kerberos, tem-se a divisão em dois tipos de mensagens para cada um dos casos anteriores, sendo as mensagens de requisição (*request*) e resposta (*reply*). Dessa forma para a comunicação com o AS se tem um AS-REQ e um AS-REP. Para comunicação com o TGS tem-se um TGS-REQ e um TGS-REP. Analogamente, para as aplicações existe um AP-REQ e um AP-REP. O detalhamento dessas mensagens entre o cliente e os serviços é mostrado na Figura 2.4

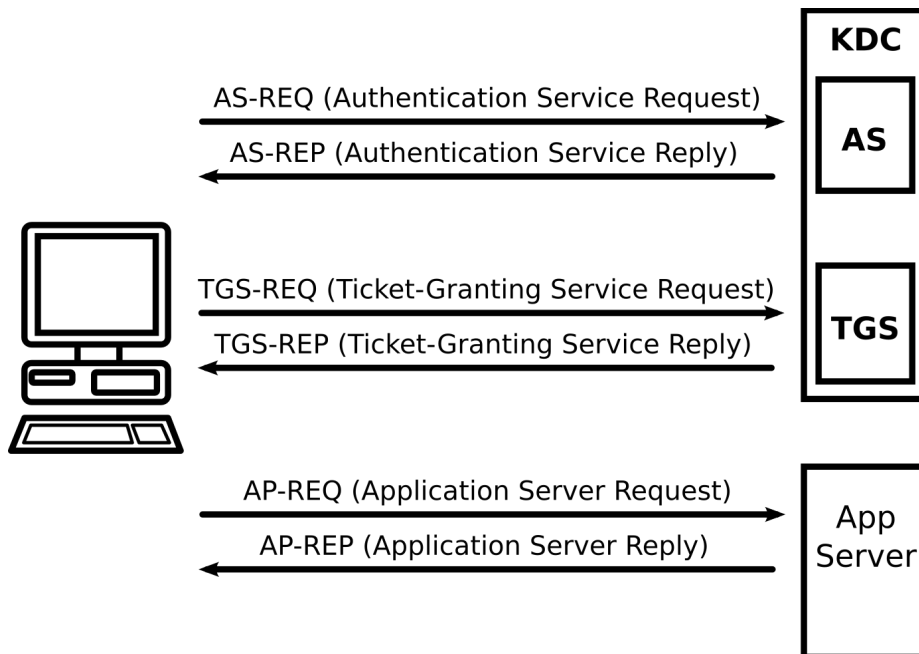


Figura 2.4: Detalhamento das mensagens trocadas entre o cliente e os serviços no Protocolo Kerberos.

### 2.2.3. Processo de autenticação

Cabe lembrar que são usadas chaves de longo prazo, chaves de sessão e tíquetes nas trocas de mensagens que irão compor o processo de autenticação, mostrados na Figura 2.5.

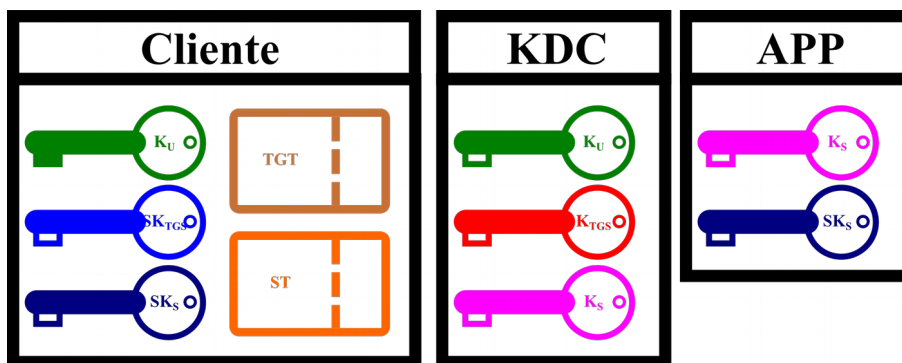


Figura 2.5: Chaves tíquetes e suas respectivas localizações nos participantes do Kerberos.

O processo de autenticação com o AS, por meio da AS-REQ, pode envolver uma pré-autenticação, que é o comportamento padrão, ou não (MICROSOFT CORPORATION, 2016b). Se a requisição é feita sem dados de pré-autenticação, a chamada vai completamente em texto claro, conforme mostra a como mostra a Figura 2.6. Na resposta AS-REP há dois blocos de informação. O primeiro bloco é cifrado com uma das chaves de longo prazo do usuário ( $K_U$ ), dependendo do tipo de criptografia negociada, de forma que só ele tenha acesso a essa informação. Ali dentro estão informações sobre o tíquete recebido, o TGT, presente no segundo bloco, e uma chave de seção

( $SK_{TGS}$ ), que será usada para cifrar a comunicação com o serviço referente ao tíquete, o TGS. O segundo bloco de informação se trata de um tíquete, o TGT. Esse bloco é cifrado com uma das chaves de longo prazo do TGS ( $K_{TGS}$ ), de forma que só o TGS possa acessá-lo. Ali dentro está presente a chave de sessão que o TGS usará para estabelecer comunicação segura com o cliente e as credenciais do usuário. O tíquete TGT será apresentado pelo usuário ao TGS para solicitar acesso a serviços.

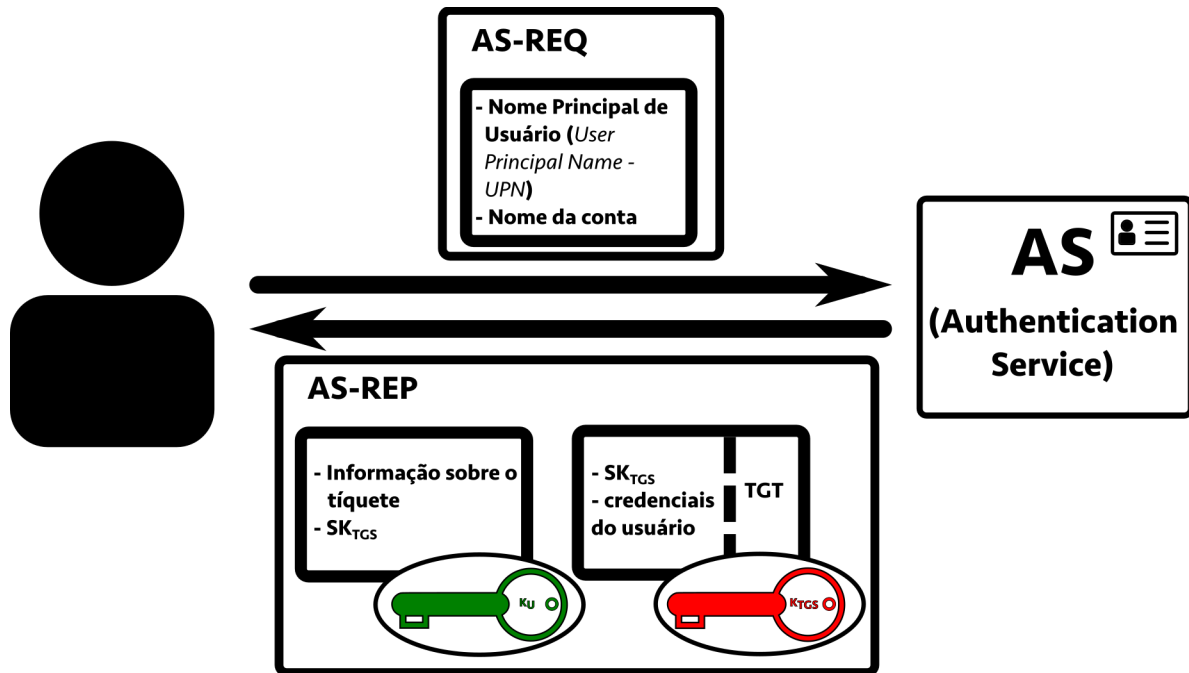


Figura 2.6: Detalhes das trocas de mensagens entre o cliente e o Serviço AS, sem dados de pré-autenticação.

Como já fora frisado, a configuração padrão para um domínio da Microsoft requer o envio de pré-autenticação por parte do cliente na requisição AS-REQ. Esta política pode ser alterada para clientes específicos. Se o processo requerer pré-autenticação mas nenhum dado de pré-autenticação for enviado na AS-REQ, o AS responde com uma mensagem de erro, do tipo `KRB_ERROR` e informará a necessidade de envio de pré-autenticação com a mensagem `KDC_ERR_PREAUTH_REQUIRED`. Nesse caso, o cliente envia uma nova requisição AS-REQ com dados de pré-autenticação presentes, como mostrado na Figura 2.7.

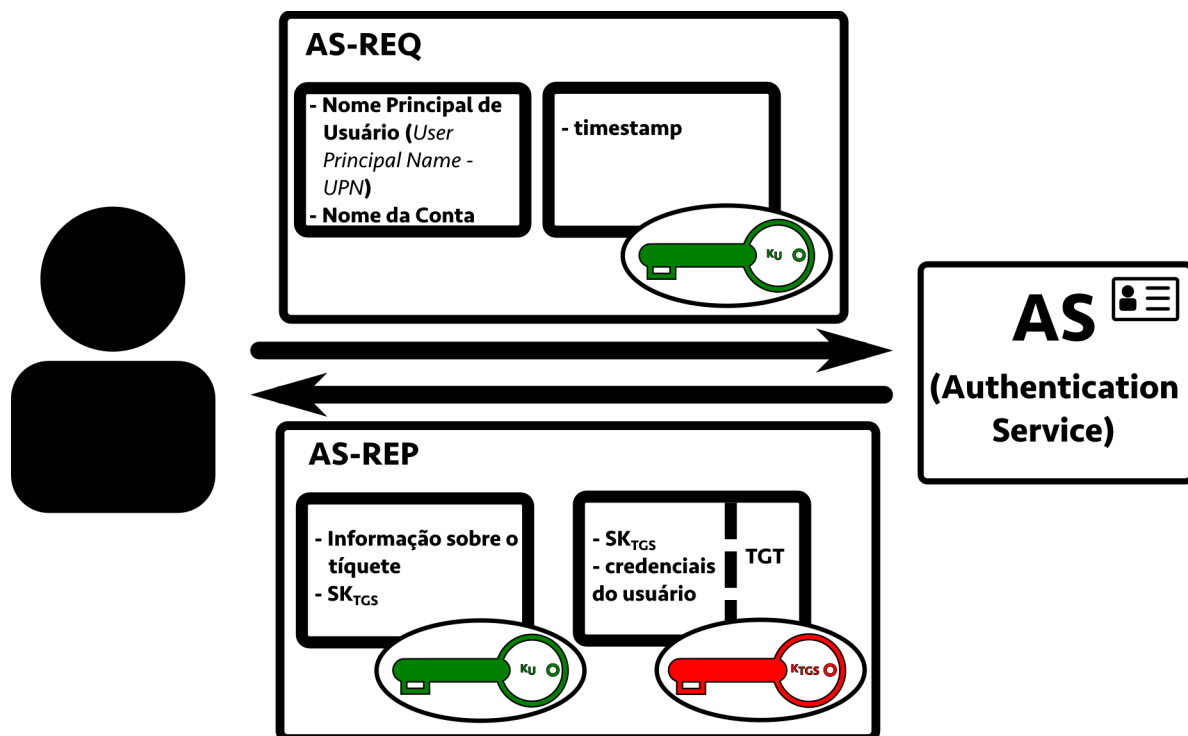


Figura 2.7: Detalhes das trocas de mensagens entre o cliente e o Serviço AS, com dados de pré-autenticação.

A pré-autenticação é composta por um carimbo de tempo (*timestamp*), que é cifrado usando uma das chaves de criptografia de longo prazo do cliente ( $K_U$ ), lembrando, um *hash* derivado da senha do usuário a partir do tipo de criptografia escolhido (PILKINGTON, 2014). O carimbo de tempo é usado para prevenir ataques *replay*, limitando a janela em que esses ataques possam ocorrer (2 minutos por padrão) (MICROSOFT CORPORATION, 2009). Daí vem a importância de manter o relógio do cliente sincronizado com o dos controladores de domínio, de forma a não interferir no processo de autenticação. Geralmente isso é feito por meio de servidores de tempo do próprio domínio. Se o servidor for capaz de decifrar o carimbo de tempo e o carimbo de tempo estiver em um intervalo de tempo válido a pré-autenticação será bem-sucedida e a resposta AS-REP será enviada ao cliente com os mesmos dados já vistos na troca de mensagens sem pré-autenticação. Com o TGT em posse o cliente passa a poder requisitar tíquetes para os serviços desejados por meio de troca de mensagens com o TGS.

O TGT é cifrado com chave do TGS ( $K_{TGS}$ ). Esta chave é compartilhada entre todos os controladores de domínio. Se trata da chave da conta KRBTGT do domínio (DUCKWAL; DELPY, 2014), de forma que só o TGS pode decifrá-lo. O TGT é apresentado ao TGS pelo cliente para requisitar tíquetes para aplicações (serviços). As adições da Microsoft incluem mais elementos no TGT, do que o especificado na RFC 4120, como o Certificado de Atributo de Privilégio (*Privilege Attribute Certificate* – PAC). O PAC é assinado usando também a chave do TGS e a chave do

serviço alvo ( $K_s$ ). No caso do PAC presente no TGT, ele recebe somente a assinatura usando a chave do TGS (da conta KRBTGT), uma vez que o serviço alvo é o próprio TGS. Esse detalhe faz diferença na criação de *Silver Tickets* e *Golden Tickets* como será visto à frente. No PAC estão informações como nome do usuário, domínio, identificador e grupos a que o usuário pertence.

A requisição ao TGS por tíquetes de serviço ocorre de forma parecida com a requisição ao AS. O cliente apresentando o TGT junto com um autenticador ao TGS. O autenticador é cifrado usando a chave de sessão compartilhada entre o cliente e o TGS ( $SK_{TGS}$ ). O TGS, ao receber a requisição, decifra o TGT, usando sua chave ( $K_{TGS}$ ), para ter acesso a essa chave de sessão ( $SK_{TGS}$ ) e com ela acessar o autenticador. O TGS responde com uma chave de sessão para a comunicação entre o cliente e o serviço ( $SK_s$ ), e um tíquete de serviço, cifrado com uma chave derivada da senha da conta do serviço ( $K_s$ ), conforme mostra a Figura 2.8.

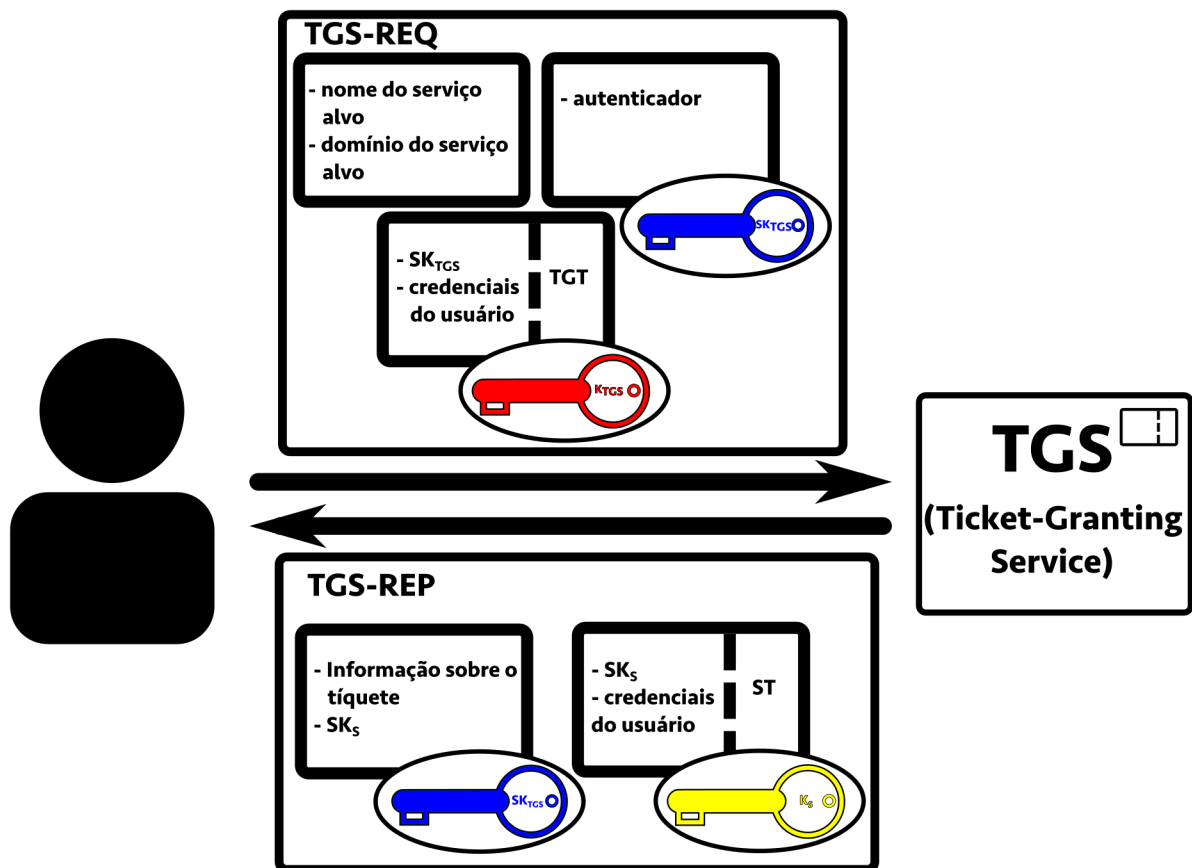


Figura 2.8: Detalhes das trocas de mensagens entre o cliente e o Serviço TGS.

De forma análoga, o cliente apresenta o tíquete de serviço junto com um autenticador à aplicação que autenticará o usuário. Caso necessário o cliente pode solicitar uma autenticação mútua para autenticar a aplicação. Essa é uma configuração feita a partir de políticas do domínio, o cliente não chega a ser solicitado sobre a autenticação mútua. Desta forma a resposta AS-REP da aplicação é

opcional. Se houver necessidade de autenticação mútua, a aplicação enviará um carimbo de tempo cifrado com a chave de sessão compartilhada entre o cliente e o serviço ( $SK_s$ ), obtida pela aplicação através do tíquete, provando ao cliente que foi capaz de decifrar o tíquete que fora cifrado com a chave do serviço ( $K_s$ ). Assim mostra a Figura 2.9.

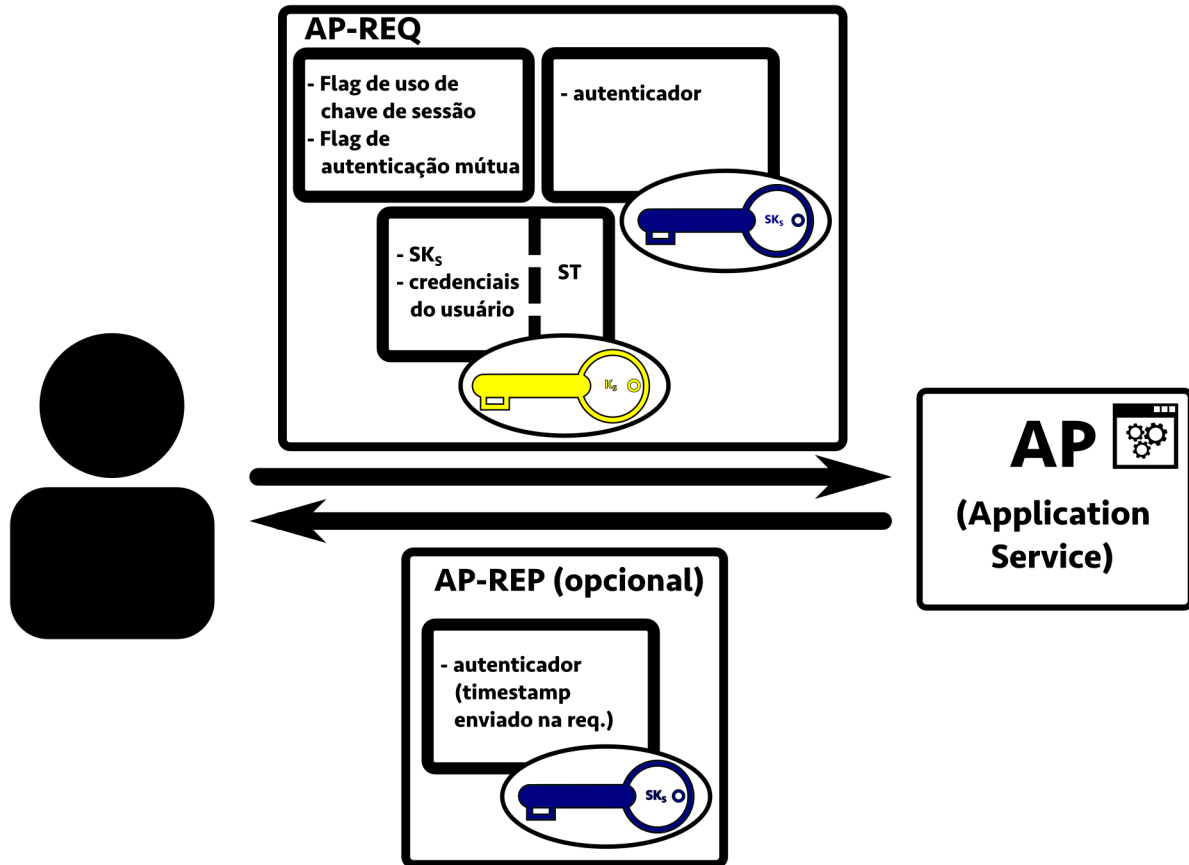


Figura 2.9: Detalhes das trocas de mensagens entre o cliente e o Serviço de Aplicação.

#### 2.2.4. Tipos de Criptografia Suportados

O Protocolo Kerberos prevê o suporte a múltiplos tipos de criptografia (*encryption types* ou *etypes*). Os etypes são mencionados na RFC 3961 (RAEBURN, 2005b). Lá são definidos mecanismos de criptografia e soma de verificação (*checksum*) a serem usados nas comunicações. Como alguns exemplos de tipos de criptografia (e seus números associados) é possível citar o DES-CBC-MD5 (com um número associado, *etype*, de 3), o AES128-CTS-HMAC-SHA1-96 (*etype* 17), o AES256-CTS-HMAC-SHA1-96 (*etype* 18) e o RC4-HMAC (*etype* 23). Dessa listagem pode-se observar que existem tanto cifradores em bloco como cifradores de fluxo.

Tabela 2.3: Tipos de criptografia constantes na RFC 3961.

Tipos de criptografia	<i>etype</i>	Seção ou comentário
des-cbc-crc	1	6.2.3

<b>Tipos de criptografia</b>	<i>etype</i>	<b>Seção ou comentário</b>
des-cbc-md4	2	6.2.2
des-cbc-md5	3	6.2.1
[reserved]	4	
des3-cbc-md5	5	
[reserved]	6	
des3-cbc-sha1	7	
dsaWithSHA1-CmsOID	9	(pkinit)
md5WithRSAEncryption-CmsOID	10	(pkinit)
sha1WithRSAEncryption-CmsOID	11	(pkinit)
rc2CBC-EnvOID	12	(pkinit)
rsaEncryption-EnvOID	13	(pkinit from PKCS#1 v1.5)
rsaES-OAEP-ENV-OID	14	(pkinit from PKCS#1 v2.0)
des-ede3-cbc-Env-OID	15	(pkinit)
des3-cbc-sha1-kd	16	6.3
aes128-cts-hmac-sha1-96	17	[KRB5-AES]
aes256-cts-hmac-sha1-96	18	[KRB5-AES]
rc4-hmac	23	(Microsoft)
rc4-hmac-exp	24	(Microsoft)
subkey-keymaterial	65	(opaque; PacketCable)

Durante a requisição AS-REQ o cliente informa ao AS os tipos de criptografia suportados por ele. Se não houver pré-autenticação o servidor deve escolher o primeiro tipo de criptografia forte válida. O que ocorre no caso da pré-autenticação é que cliente faz a primeira requisição AS-REQ sem dados de pré-autenticação e na resposta de erro AS-REP (*preauthentication required*) o servidor AS informa os tipos de criptografia que ele, o servidor, suporta. Essa resposta pode incluir a especificação de *salt* (sal) para os *etypes* que requerem isso. O cliente então deverá escolher o primeiro tipo de criptografia forte válida, suportada mutuamente pelo cliente e o serviço (NEUMAN *et al.*, 2005; PILKINGTON, 2014), que servirá para cifrar o carimbo de tempo a ser enviado na próxima AS-REQ.

### 2.3. ARMAZENAMENTO DE CREDENCIAIS NO WINDOWS

Credenciais do Windows podem ser armazenadas de diferentes formas. Podem ser encontradas em uma base de dados no disco do computador cliente como a Security Account Manager (SAM) (MARTIN; TOKUTOMI, 2012).

Um componente chamado *LSA secrets* armazena *cache* de credenciais do domínio que são usadas pela *Local Security Authority* (LSA). É uma espécie de armazenamento protegido. A LSA é responsável, dentre outras coisas, pelo logon de usuários no sistema. Credenciais podem ser

armazenadas na memória do computador por meio do processo LSASS (DUCKWALL; DELPY, 2014; METCALF, 2014a). Senhas podem ser armazenadas usando *hashes* LM ou NTLM. Cabe ressaltar que o *hash* NTLM é a mesma chave de entrada para o tipo de criptografia RC4-HMAC, *etype* 23. Aparentemente isso foi feito por questões de compatibilidade. LSASS é um acrônimo que vem de *Local Security Authority Subsystem Service* (Serviço de Subsistema de Autoridade de Segurança Local). Nesse caso, pode haver armazenadas senhas com criptografia reversível (*reversibly encrypted passwords*), *hashes* de senhas (NT ou LM) ou tíquetes do Kerberos (TGT e outros tíquetes de serviço).

*Hashes* também podem ser achadas na base de dados dos controladores de domínio do AD, que retém todas as credenciais do domínio, por acesso via serviços de diretórios ou no arquivo NTDS.DIT (HUMMEL, 2009; JUNGLES *et al.*, 2012; METCALF, 2016).

Há também a possibilidade de que credenciais de domínio sejam armazenadas localmente de forma que usuários possam fazer logon de forma interativa mesmo se o controlador de domínio não estiver disponível. É o caso de laptops e notebooks que são colocados como participantes de um domínio e precisam que seus usuários sejam capazes de fazer logon mesmo que não estejam na estrutura do trabalho. Este mecanismo é conhecido como *cached credentials* (JUNGLES *et al.*, 2012).

Estes são alguns exemplos de onde credenciais podem ser furtadas. Estes locais são visados por atacantes que buscam extrair credenciais. Atacantes geralmente, após comprometer um sistema, escalam privilégios e tentam obter credenciais desses locais, procurando no disco rígido ou com alguma ferramenta que leia a memória ou faça despejo dela.

#### **2.4. ATAQUES PASS-THE-HASH E PASS-THE-TICKET**

PtH e PtT são técnicas que utilizam um *hash* de senha, ou um tíquete do Kerberos, para personificar o usuário que detém a senha para aquele *hash* ou o tíquete. Nesses casos tanto o *hash* como o tíquete funcionam como equivalentes de senha.

O protocolo Kerberos especifica operações a serem aplicadas na senha de usuário para transformar uma senha de tamanho variável em uma chave de longo prazo de tamanho fixo, que servirá como entrada para as várias formas de algoritmos de criptografia suportados pela especificação (AES128, AES256, DES, 3DES, RC4...). Exemplos de tais operações podem ser encontradas na RFC 4757 (ZHU; JAGANATHAN; BREZAK, 2006), com o tipo de criptografia RC4-HMAC, e na RFC 3962 (RAEBURN, 2005a), com o tipo de criptografia AES com chaves de 128 e 256 bits de tamanho. As chaves geradas podem ser armazenadas no processo LSASS para



serem usadas pelo usuário durante a sessão (sessão LSA). Como essas chaves servem como entrada para os diferentes tipos de algoritmos de criptografia, elas são usadas nesse sentido quando furtadas, como equivalentes de senha. Há várias ferramentas para executar esses ataques. Mimikatz (DUCKWALL; DELPY, 2014), Psh-toolkit, Msvctl, Metasploit PSEXEC module, Tenable smbshell e JoMo-kun (BASHAR, 2010) são algumas delas. Nas seções seguintes são apresentadas alguns diferentes tipos desses ataques. Elas apresentam principalmente a abordagem tomada pela ferramenta Mimikatz, que usa a injeção de chaves e tíquetes na memória por meio do processo LSASS.

### 2.4.1. Overpass-the-Hash (Pass-the-Key)

Um modelo de autenticação padrão, largamente conhecido, é aquele em que o servidor armazena em sua base de dados os *hashes* da senha de usuário, Isso é feito para que, caso a base de dados seja comprometida, o atacante não tenha acesso imediato ao texto claro das senhas de usuários. Em uma arquitetura cliente-servidor, em um processo de autenticação, é comum o cliente passar a senha para o servidor, por meio de um canal seguro preestabelecido, o servidor calcular o *hash* da senha e compará-lo com o armazenado em sua base de dados, autenticando o usuário caso os *hashes* coincidam. Imagine uma aplicação em que o *hash* da senha de usuário seja calculado no cliente e enviado por meio de um canal inseguro para o servidor, que comparará os *hashes* e o autenticará caso coincidam. Um atacante capturando o tráfego na rede pode obter esse *hash* e repetir a chamada do cliente ao servidor, personificando o usuário e obtendo a autenticação do servidor. Esse exemplo é o tipo de ataque conhecido como *Pass-the-Hash*.

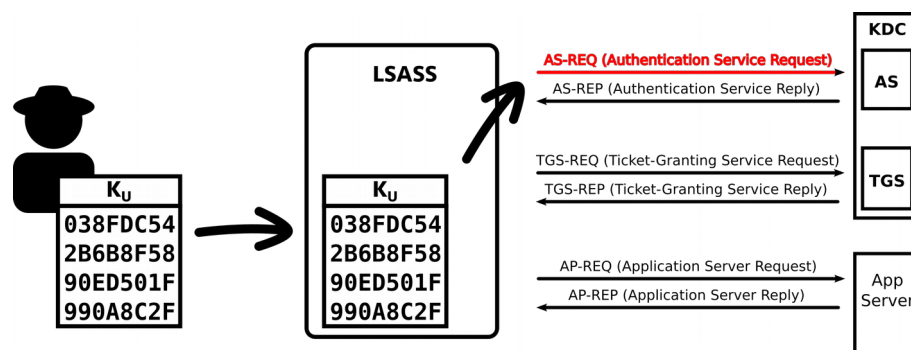


Figura 2.10: Ataque *Overpass-the-Hash* (*Pass-the-Ticket*) com o uso de uma chave obtida.

Apresentado na Figura 2.10, como em uma subcategoria ou uma especialização, na técnica *Overpass-the-Hash* ou *Pass-the-Key* o atacante obtém uma das chaves de longo prazo de um usuário, que são derivadas da senha dele, e a injeta em memória, no processo LSASS. Assim quando a requisição AS-REQ é feita, essa chave será usada para criar um carimbo de tempo cifrado como autenticador.

## 2.4.2. Pass-the-Ticket

Nesse tipo de ataque o invasor obtém um tíquete de serviço válido, gerado pelo AS ou TGS, em face a uma requisição do usuário. O tíquete obtido pode ser tanto um TGT ou tíquete de serviço. O tíquete é injetado na LSASS e usado para requisitar outros tíquetes ou acessar o serviço correspondente ao tíquete.

Apesar de amplamente conhecido na literatura como PtT, esse tipo de ataque, quando usando o TGT, mereceria um nome especial dentro das subcategorias de ataques PtT. Defende-se isso por ele se tratar de um tíquete especial quando comparado com os tíquetes de serviço e dar mais poder ao atacante, o poder de requisitar acesso a serviços em nome daquele usuário a quem o tíquete pertence. Esse ataque é apresentado na Figura 2.11. O nome poderia ser algo como *Pass-the-TGT*, fazendo um paralelo com a nomenclatura do Pass-the-Key. O outro ataque, que utiliza um tíquete de serviço obtido, apresentado na Figura 2.12, poderia se chamar *Pass-the-TGS* ou *Pass-the-ST*.

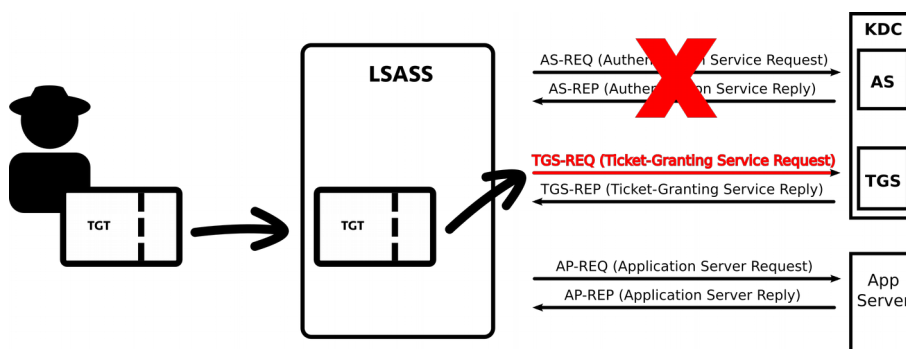


Figura 2.11: *Pass-the-Ticket* com o uso de um tíquete TGT obtido.

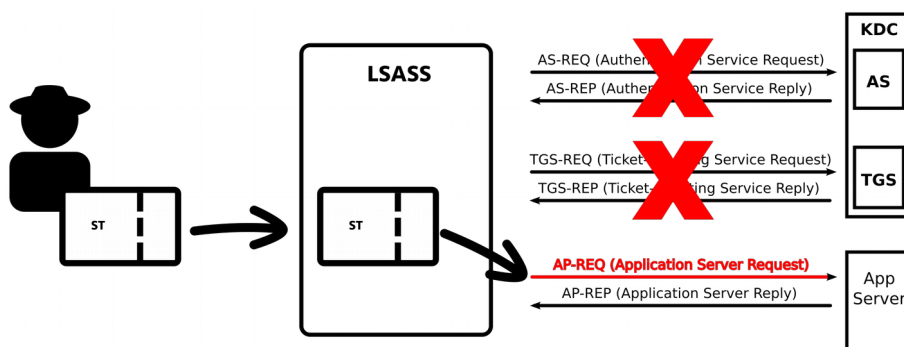


Figura 2.12: *Pass-the-Ticket* com o uso de um tíquete de serviço obtido.

Essa diferenciação na nomenclatura possui a vantagem também de diferenciar esses ataques do seu conjunto pai, PtT, que englobaria o Pass-the-TGT, Pass-the-TGS e *Silver Tickets*.

### 2.4.3. Silver Tickets

Um atacante que tem acesso a uma chave de um serviço ( $K_S$ ) na rede é capaz de usar essa chave para forjar tíquetes para acesso àquele serviço. Esse tíquetes são conhecidos como *Silver Tickets* (METCALF, 2015b). O atacante pode escolher o usuário que ele deseja usar para acessar o serviço como, por exemplo, o usuário *Administrator*. Pode escolher até mesmo um usuário que não exista no domínio. O atacante também é capaz de manipular os grupos aos quais aquele usuário pertence. Assim o usuário escolhido pode ser colocado como participante de um grupo de administradores. O processo do ataque é mostrado na Figura 2.13.

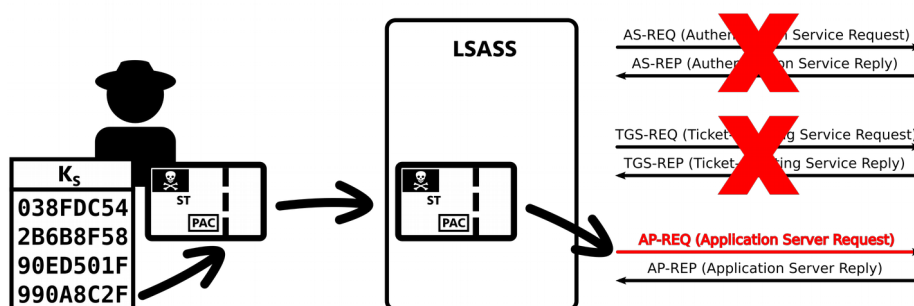


Figura 2.13: Silver Ticket forjado com o uso de uma chave de serviço obtida.

Cabe lembrar que um tíquete de serviço é cifrado com a chave da conta do serviço alvo ( $K_S$ ). Dentro de qualquer tíquete de serviço (ST), seja ele válido ou forjado, estão informações como a identificação do usuário, serviço alvo (no caso do tíquete TGT o serviço alvo é o TGS, da conta KRBTGT), período de validade e chave de sessão a ser compartilhada com o serviço alvo ( $SK_S$  ou  $SK_{TGS}$ ). Além disso, existe uma estrutura chamada *Privilege Attribute Certificate* – PAC. Nessa estrutura estão informações sobre o usuário e grupos a que ele pertence. Essa estrutura é assinada usando a chave do TGS ( $K_{TGS}$ ) e a chave do serviço alvo ( $K_S$ ).

É justamente no PAC que um atacante irá atuar para se incluir em grupos com privilégios elevados. Isso é possível porque geralmente as aplicações não validam a informação presente no PAC com o KDC, confiando no que está ali, assinado com a sua chave, isto é, a chave do serviço. Além de não validarem as informações presentes no PAC, as informações presentes no tíquete também não costumam ser validadas (DUCKWALL; DELPY, 2014), como existência ou habilitação da conta do usuário, permitindo que atacantes impersonifiquem usuários desabilitados ou inexistentes.

Esse tíquete é limitado ao serviço específico cuja chave fora obtida pelo atacante. Tíquetes de serviço tem uma validade padrão de 10 horas.

#### 2.4.4. Golden Tickets

Quando a chave comprometida é chave para a conta KRBTGT do domínio, o atacante tem a possibilidade de criar os seus próprios tiquetes TGT, usando como alvo qualquer serviço da rede (CERT-EU, 2014; METCALF, 2014b, personificando qualquer usuário e com os privilégios que ele desejar. Esses tiquetes são conhecidos como *Golden Tickets*. O processo de funcionamento de *Golden Tickets* é mostrado na Figura 2.14.

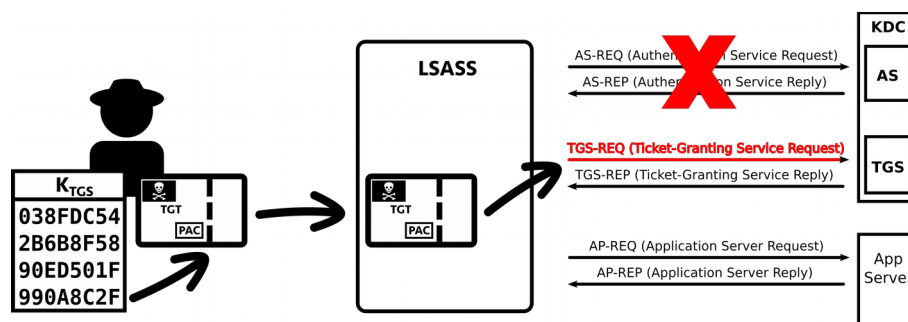


Figura 2.14: Golden Ticket forjado a partir da chave do TGS (conta KRBTGT).

O processo é relativamente parecido com o da criação de *Silver Tickets*. O TGS, na implementação do Windows, somente valida a existência e/ou habilitação da conta do usuário se o TGT for mais velho que 20 (vinte) minutos. Isso permite que sejam usados usuários que estão desabilitados ou que não existam no domínio. Mas isso também não impede a criação de um novo *Golden Ticket* TGT. Além disso, qualquer tiquete de serviço (ST) obtido nesse intervalo de 20 minutos terá uma validade padrão de 10 horas.

Quanto ao PAC, a única chave necessária para assiná-lo é a chave do TGS ( $K_{TGS}$ ), já que o serviço alvo é o próprio TGS, que já foi comprometida. Vale lembrar que, no caso dos *Silver Tickets*, para assinar um PAC seriam necessárias a chave do serviço e a chave do TGS. Naquele caso (*Silver Tickets*), o atacante depende do serviço não validar o PAC com o KDC. Além disso, para esse caso (*Golden Tickets*) assim como para os serviços (*Silver Tickets*), o TGS confia nas informações do PAC, no tocante aos grupos para os quais o usuário está registrado.

Outro ponto interessante é que a validade padrão da senha da conta KRBTGT é de 10 anos e são poucos os eventos que disparam a necessidade de mudança dessa senha. Tais eventos podem ser:

1. Atualização do nível funcional do domínio, por exemplo, do NT5 para o NT6;
2. Recuperação do Sistema Operacional do servidor a partir de mídias de recuperação;
3. Mudança manual;

## 2.5. FORMAS DE COMBATE A ATAQUES PTH&T

Foram estudadas as medidas aplicadas a combater ataques PtH&T., focando o estudo nas fases de prevenção e detecção, que são de maior interesse para o estudo, sem se aprofundar no tocante a medidas de resposta aos ataques e recuperação em um evento desses.

### 2.5.1. Medidas de Prevenção

Como o uso e armazenamento de um *hash* de senha como chave está no núcleo do Kerberos, não é fácil superar esse problema. Assim consultores sugerem medidas para mitigar esses ataques (JUNGLES *et al.*, 2012). Como exemplo de algumas técnicas de mitigação, Jungles et al. (2014) sugerem ações em cinco categorias de estratégias:

1. Identificar ativos de alto valor;
2. Proteção dos ativos;
3. Detecção;
4. Resposta; e
5. Recuperação dos ataques.

Alguns exemplos de ativos de valor podem ser controladores de domínio, servidores de arquivos, contas e grupos de administrador ou equivalentes. Um exemplo mencionado da proteção de ativos é a implementação de diferentes níveis de camadas para administradores de domínio, administradores de aplicações e administradores de estações de trabalho, como forma de prover controle administrativo e evitar uma escalada de privilégios. Outras medidas protetivas são implementar restrições de logon, segmentação de rede e autenticação de múltiplos fatores. Na categoria de detecção é afirmado que o foco no uso de credenciais furtadas é mais efetivo para a detecção do que tentar identificar o furto delas. A detecção pode ser feita baseada na coleta e análise de registros de eventos buscando eventos específicos. Existem opções para centralizar a coleta de registros como o *Windows Event Collector*, *Audit Collection Services* ou soluções SIEM – *Security Information and Event Management* – de terceiros. Atividades de resposta são compostas por endereçar vetores de ataque e investigação. Recuperação trata de retomar o controle de recursos comprometidos. Pode haver a necessidade de mudança de senhas de contas ou computadores comprometidos, ou restaurar completamente um domínio exposto. Deve-se ter atenção especial à senha da conta KRBTGT e, se necessário, alterá-la duas vezes (CERT-EU, 2014).

Três ações de mitigação são destacadas por Jungles *et al.* (2012, 2014) e Thomlinson (2012):

1. Restringir e proteger contas de domínio com altos privilégios;
2. Restringir e proteger contas locais com privilégios de administrador; e
3. Restringir o tráfego de entrada usando o firewall do Windows.

Mais algumas ações de mitigação são sugeridas como a remoção de usuários padrão do grupo de administradores locais e a limitação do número e uso de contas de domínio com privilégios, dentre outras.

### 2.5.2. Inovações no Sistema Operacional

De tempos em tempos as novas versões do Windows trazem alguma novidade que visa endereçar o problema do furto de credenciais.

Uma delas foi a criação do *Protected Users Group* – Grupo de Usuários Protegidos – (MICROSOFT CORPORATION, 2014), que esteve disponível desde o Windows 7 e do Windows Server 2008 R2. Aos usuários adicionados a esse grupo são empregadas automaticamente medidas de segurança adicionais. Algumas dessas medidas são:

- Desabilitada autenticação NTLM, Digest ou CredSSP;
- Senhas não são armazenadas no *cache* em memória;
- Não serão usadas os tipos de criptografia DES ou RC4 no Protocolo Kerberos.

Claramente, essas medidas tentam dificultar o acesso de *crackers* a *hashes* de senhas.

Uma outra funcionalidade é a *LSA Protection* (proteção LSA), disponível desde o Windows 8.1 e Windows Server 2012 R2 (JUNGLES *et al.*, 2014), que transforma o processo LSASS em um processo protegido do sistema operacional. Isso tenta impedir que processos que não sejam assinados pela Microsoft, por uma Autoridade Certificadora aprovada, adulterem o processo protegido. Apesar disso, existem ferramentas que são capazes de passar por cima dessa medida protetiva, como o Mimikatz (METCALF, 2015a).

As versões mais recentes do sistema operacional, o Windows 10 e Windows Server 2016, trazem como novidade os recursos *Device Guard* e *Credential Guard* (BRIAN LICH, 2016; MADDIE EGAN, 2016; ZYLVA, 2016). O recurso Credential Guard usa tecnologia de virtualização para tirar o processo LSASS da memória do sistema operacional e isolá-lo em outra área (*Isolated LSA*), como se pertencesse a um sistema operacional convidado (*Virtual Secure Mode*). A grosso modo pode-se imaginar um *hypervisor*, do tipo-1 (bare-metal) e acima deles dois sistemas operacionais convidados, um sendo o SO padrão em utilização pelo usuário e o outro com o processo protegido.

Embora essa alegoria não esteja completamente precisa, ele serve bem para explicar como a tecnologia funciona.

### 2.5.3. Detecção de Eventos

Uma das formas mais comuns para a tentativa de identificação de ataques PtH&T é a coleta e a análise de *logs* de eventos. Esse *logs* podem vir de vários computadores e podem ser concentrados e gerenciados em uma entidade central usando, por exemplo, uma solução do tipo *SIEM – Security Information and Event Management* – para produzir alertas para eventos específicos.

Jungles *et al.* (2014) citam uma coleção de 15 eventos de interesse que podem ser usados para uma análise em busca de ataques PtH&T e ainda sugere a coleta de outros que possam vir a interessar. Esses eventos estão listados na Tabela 2.4.

Tabela 2.4: Eventos de interesse que podem ser usados na coleta e análise em busca de ataques.

<b>ID do evento</b>	<b>Descrição</b>
4688	Um novo processo foi criado.
4648	Um logon foi tentado usando credenciais explícitas.
4624	Uma conta foi logada com sucesso.
4769	Um tíquete de serviço do Kerberos foi requisitado.
4768	Um tíquete de autenticação do Kerberos (TGT) foi requisitado.
4776	O controlador de domínio tentou validar credenciais para uma conta.
100	Tentativa de uso de NTLM.
104	Tentativa com DES ou RC4 para Autenticação Kerberos.
101	Tentativa de uso de NTLM.
105	Autenticação Kerberos de um dispositivo em particular não foi permitida.
106	Ao usuário ou dispositivo não foi permitido autenticar no servidor.
305	TGT do Kerberos não preencheu as restrições de controle de acesso.
306	Usuário, dispositivo ou ambos não preencheram as restrições de controle de acesso.
3065	A checagem de integridade de código determinou que um processo tentou carregar um driver particular que não preencheu os requisitos de segurança para Seções Compartilhadas. Entretanto, diante da política de sistema que está em vigor, foi permitido que a imagem fosse carregada.
3066	Este evento registra uma checagem de integridade de código que determina que um processo (geralmente lsass.exe) tentou carregar um driver particular que não preencheu os requisitos de nível de assinatura da Microsoft. Entretanto, diante da política de sistema que está em vigor, foi permitido que a imagem fosse carregada.

Um alerta com algum desses eventos não determina a existência de um ataque, mas deve ser analisado e investigado por tal. Não está claro até onde esse processo pode ser automatizado para diminuir o tempo gasto por administradores de rede na análise e investigação desses eventos. Além disso, alguns desses eventos parecem por demais genéricos, como, por exemplo, o 4688 e 4624. Outros são mais específicos, como os de ID de evento 3065 e 3066, que têm a ver com o carregamento de um driver não autorizado por parte do processo lsass.exe. No entanto, esses eventos só serão disparados se o modo de auditoria estiver habilitado para a LSASS.

#### **2.5.4. Detecção de Anomalia**

Já foi afirmado que é difícil diferenciar um logon legítimo de um não legítimo (JUNGLES *et al.*, 2012). Entra em cena a detecção de anomalia que normalmente usa algum método de aprendizado de máquina para realizar sua tarefa (*Support Vector Machines*, Redes Neurais, Modelos de Markov, etc.). Embora parte do trabalho não esteja focada especificamente na detecção de ataques PtH ou PtT em Redes Windows, esses tipos de modelos são elegíveis para isso porque lidam com o uso de credenciais furtadas em cenários de pós-exploração.

A detecção de anomalias tenta capturar atividades anormais quando comparadas um conhecimento prévio, isto é, a um padrão esperado (CHANDOLA; BANERJEE; KUMAR, 2009). Elas se concentram na maneira incomum que um usuário pode estar usando ou tentando acessar recursos na rede. A ideia principal é criar um perfil das atividades comuns de um usuário e tocar um sino quando esse usuário não estiver se comportando de maneira natural ou esperada.

Compreendendo a área de detecção de anomalia, Hsieh *et al.* (2015) modelam o comportamento do usuário e estimam o uso provável de certas operações para detectar anomalias. Eles visam principalmente ameaças internas (inside threats). Isso é feito analisando um conjunto de dados de registros do Active Directory de uma organização real, usando a probabilidade do modelo de Markov para modelar as contas, centrando nas ações do usuário em eventos sequenciais de tempo. Foi utilizada uma validação cruzada *k-fold* para avaliar o método.

Sapegin *et al.* (2015) propõem um algoritmo de detecção de anomalias baseado na distribuição de Poisson para detectar usuários maliciosos que não causam uma violação de acesso. O trabalho se concentra em atividades de logon com um grande volume de eventos a serem analisados. Eles produziram um experimento e usaram um conjunto de dados de treinamento e um conjunto de dados de teste em seu cenário. Eles concluíram que o algoritmo funcionou bem, embora sua configuração tenha escopo limitado e pudesse usar mais alguns cenários e validação.



Yu (2015) introduz um modelo para detectar usuários disfarçados usando valores de peso para diferentes eventos, a fim de construir um perfil de usuário e novas atividades são comparadas a perfis de eventos passados usando lógica difusa (*fuzzy*) para avaliar o nível de ameaça.

### 2.5.5. Detecção de Honeytokens

Nos trabalhos anteriores envolvendo detecção de anomalia, ainda é preciso se preocupar com taxas de precisão e sensibilidade (*recall*). Neste sentido, um sistema de alarme pode ocupar um valioso tempo de administradores/equipes de resposta investigando eventos fora do interesse. Quando se trata de *honeytokens*, é mais provável que algo esteja errado (DE BARROS, 2003), e talvez seja por isso que os falsos positivos geralmente não aparecem na discussão neste campo específico.

*Honeytokens* derivam do conceito de *honeypots*, que são serviços ou sistemas isca, planejados para chamar a atenção de um atacante e serem explorados por ele. Para serem explorados, os sistemas devem possuir alguma vulnerabilidade. Existe também o conceito de *honeynets* que são agregações, conjuntos ou redes compostas de *honeypots* visando ser um cenário heterogêneo para um atacante (THE HONEYNET PROJECT, 2006).

Um *honeypot* pode ser classificado pela perspectiva de implantação como sendo um *honeypot* de produção ou um *honeypot* de pesquisa (LÓPEZ; RESÉNDEZ, 2008). O primeiro, como o nome sugere, é instalado em um ambiente de produção, ou seja, sistemas de negócio verdadeiros e ativos, propostos a detectar e mitigar ataques em tempo real, dando a oportunidade de uma resposta rápida a um intruso. O último, são direcionados a estudar o atacante e entender a sua motivação. São geralmente relacionados a sistemas em universidades. *Honeypots* também podem ser classificados baseados no critério de nível de interação com um atacante (LÓPEZ; RESÉNDEZ, 2008). Um *honeypot* de baixa interação (*low-interaction*) pode simular apenas algumas partes de um sistema, como a ferramenta *honeyd* que simula uma pilha de rede e não um sistema operacional completo. Ao contrário de *honeypots* de baixa interação, *honeypots* de alta interação (*high-interaction*) simulam sistemas inteiros com o intuito de um atacante interagir com o sistema. Um exemplo de ferramenta nessa categoria é o projeto Honeywall CDROM (THE HONEYNET PROJECT, 2008).

Como se pode ver, o termo *honey* é comumente usado para se referir a objetos de engodo (JUELS, ARI, 2014). Comparados a *honeynets* e *honeypots*, *honeytokens* são relacionados a coisas com menores granularidades que uma rede, sistema ou serviço. Eles podem ser arquivos, entradas de bancos de dados, contas de usuários, endereços de e-mail ou senhas (DE BARROS, 2003). No caso das senhas é comum observar uma especialização do termo, sendo usada a palavra *honeyword*. Ao longo do texto a palavra *honey* é usada nesse sentido e daí obtém-se os termos *honeyhashes* e

*honeytickets*, como expressão especializada para denotar *hashes* e tíquetes de engodo, guardando o termo *honeypoken* para referir-se a ambos.

Em 2016 uma ferramenta foi lançada baseada no conceito de usar *honeytokens* de senhas para detectar atacantes. Essa ferramenta é chamada DCEPT, um acrônimo para *Domain Controller Enticing Password Tripwire* (STEWART; BETKE, 2016). Ela gera senhas forjadas, as armazena em uma base de dados em um servidor, as coloca na memória de estações de trabalho Windows e monitora o tráfego do controlador de domínio esperando detectar um desses *honeytokens* gerados para disparar um alarme.

Existem trabalhos recentes em torno de usar *honeywords* para confundir um invasor de forma que ele não seja capaz de diferenciar entre um *hash* de senha válido e um que não é. Esses trabalhos tratam o uso de *honeywords* de formas gerais e não voltados para Redes Windows e ataques PtH&T. Isso reforça a ideia de que haja um *gap* no uso de uma abordagem de *honeytokens* para a detecção.

Juels e Ristenpart (2014) estudaram uma técnica denominada *honey encryption*, que é uma forma de cifrar uma mensagem com a característica de que quando esta mensagem seja decifrada com uma chave errada, o texto claro se pareça com uma mensagem válida mas falsa, confundindo o *cracker*, em vez de gerar um erro quando o processo de decodificação falhar, o que beneficiaria o atacante.

Khedkar *et al.* (2016) propõem armazenar o *hash* da senha do usuário junto com diversos outros *hashes* provindos de *honeywords*, assim um atacante não pode ter certeza de ele usar a senha real ou uma falsificada, no caso de um vazamento do banco de dados de *hashes*. O uso de uma senha falsa dispararia um alarme.

### **3. ANÁLISE DE UMA ABORDAGEM DE DETECÇÃO COM HONEYTOKENS**

Detectar atacantes usando técnicas PtH&T é difícil por causa das similaridades entre um processo de autenticação e concessão de tíquetes normal e um anormal. A detecção de eventos em busca de registros que apontem tais ataques não garantem ou determinam que ele esteja ocorrendo. Um registro de evento de autenticação pode se tratar de apenas um usuário se autenticando e precisaria de uma investigação e análise para descobri-lo. A detecção de anomalia precisa se preocupar com falsos positivos e falsos negativos.

Apesar da técnica de honeytokens ser uma indicação mais imediata de que algo ilícito está ocorrendo, ela não é livre de problemas e deve ser usado em conjunto com outras técnicas. Nesse sentido foi montada um cenário com máquinas virtuais e diversas ferramentas. A ferramenta de detecção de intruso DCEPT foi instalada e configurada. Um logon interativo usando um honeypoken foi tentado para que ele fosse detectado pela ferramenta. Ao final do capítulo as descobertas, problemas e aprendizados são discutidos.

Para a montagem, execução e entendimento do experimento e das discussões e análises seguintes, faz-se necessário pelo menos um conhecimento básico de algumas ferramentas e sistemas operacionais como o VirtualBox, Ubuntu Linux, Windows Server, Active Directory, Visual Studio Express, Ansible, SQLite, John the Ripper, Mimikatz. Mais detalhes sobre as ferramentas e o uso delas estão presentes no Anexo C.

#### **3.1. MODELO ADVERSARIAL**

Um ambiente típico de uma Rede Windows em uma organização A é aquele existente em uma rede corporativa interna (*intranet*) e é composto basicamente por 3 tipos de componentes:

1. Estações de trabalho cliente;
2. Servidores de recursos; e
3. Controladores de domínios.

Um exemplo de um ambiente de Rede Windows é apresentado na Figura 3.1. As estações de trabalho podem fazer parte do Domínio Windows ou não. Elas podem ter acesso aos recursos corporativos de um local de dentro da organização provedora do ambiente ou remotamente, por meio de Rede Privada Virtual (*Virtual Private Network – VPN*). É possível também que essas estações de trabalho que acessam recursos pertençam a outras organizações, uma organização B por

exemplo, que compartilhem recursos com a organização A. Os servidores de recursos podem ser compartilhamentos de arquivos do Windows, impressoras, servidores web, servidores de backup, etc. Se tratam de serviços que desejam autenticar um usuário ou computador cliente em nome de um usuário. Os controladores de domínio desempenham o papel de terceiro de confiança na autenticação dos usuários perante os serviços.

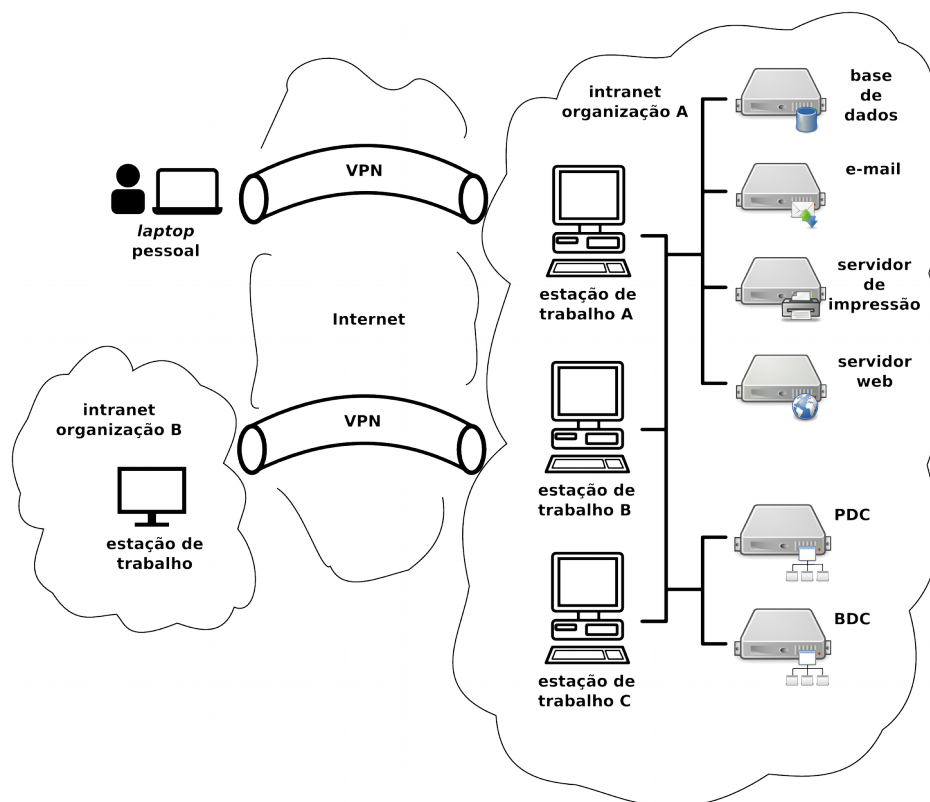


Figura 3.1: Exemplo de um ambiente de Rede Windows.

Nesse cenário, vislumbra-se dois tipos de atacantes:

1. Atacante externo;
2. Atacante interno.

O atacante externo é um adversário típico, não participante da organização, que busca acesso a recursos dentro da organização. Ele pode ter acesso ao ambiente interno da organização A por meio de técnicas de *phishing*, e-mails com anexos maliciosos, *keyloggers*, etc. Ressalta-se que o uso de *keyloggers* para a obtenção de senhas não configura um ataque PtH&T, objeto do estudo. Esse acesso pode se iniciar pelo comprometimento de um computador de dentro da organização ou da organização B.

O atacante interno é um funcionário da organização com privilégios limitados. Ele tem acesso físico a certa quantidade de computadores da organização A e, portanto, uma maior capacidade lesiva, de acordo com a terceira lei imutável da segurança proposta por Culp (2000).

Deixa-se de fora do modelo atacantes internos com privilégios elevados, como administradores de domínio, que seriam capazes de obter acesso a recursos sem a utilização de ataques PtH&T.

Ambos os atacantes são capazes de explorar um sistema assim que obtenham acesso a ele (como por meio de um terminal de comando – *command prompt*), através da exploração de alguma vulnerabilidade presente, obtendo assim privilégio de sistema e permitindo acesso às áreas protegidas do sistema, ainda que residam em um compartimento seguro, como o usado na tecnologia *Credential Guard*, o que poderia ser conseguido por meio do comprometimento do *hypervisor* ou sistema de inicialização (*boot*). Essa propriedade possui o intuito de preservar a premissa de um cenário de pós-exploração.

Após comprometer um computador, ambos os atacantes usam os *hashes* e tíquetes encontrados e tentam executar ataques PtH&T em novos computadores, realizando movimentos laterais, escalando áreas mais sensíveis da rede.

### **3.2. ESTUDO DOS PROBLEMAS USANDO O DCEPT**

Como não há ou não foi possível encontrar na literatura, além da ferramenta DCEPT, uma outra ferramenta que use a abordagem da detecção de *honeytokens* para a detecção de intruso usando técnicas PtH&T em Redes Windows, essa ferramenta é tomada como referência ao estado da arte para esse tipo de abordagem. O estudo e a análise da ferramenta DCEPT visa mostrar como uma abordagem de detecção de *honeytokens* pode ser feita para combater ataques PtH&T e elucidar os problemas existentes.

O DCEPT é uma ferramenta desenvolvida em um projeto de pesquisa da DELL e colocada publicamente a disposição como prova de conceito em benefício de administradores de sistema (*sysadmins*) de Redes Windows. Como é colocado na seção LEIA-ME, na página inicial do sítio *web* do projeto no Github, “a meta do projeto é prover uma ferramenta de detecção de *honeypoken simples e grátis, além de educar administradores de sistemas Windows sobre a natureza desses ataques*” (SECUREWORKS, 2016). Pode-se notar que a ferramenta não constitui solução definitiva, mas sim um terreno a ser construído.

### 3.3. ARQUITETURA E CONFIGURAÇÃO DE DETECÇÃO

Para avaliar o DCEPT e verificar seus detalhes de funcionamento, foi preparada uma configuração com 3 máquinas virtuais conforme mostra a Tabela 3.1. Uma para o controlador de domínio do Active Directory (DC), outra para a função de uma estação de trabalho do Domínio Windows e a terceira para ter o DCEPT instalado e em execução. O DC foi instalado em um Windows Server 2008. A versão do Windows usada para a estação de trabalho foi a versão 8.1. O DCEPT foi instalado em um Ubuntu Server 14.04.

Tabela 3.1: Informações sobre as máquinas virtuais do experimento.

Nome de <i>host</i>	Sistema Operacional	Destinação	Endereço IP
vm-windows-81	Windows 8.1	Estação de trabalho cliente	192.168.15.4
win-pdc	Windows Server 2008	Controlador de Domínio AD	192.168.15.14
Ubuntu	Ubuntu Server 14.04	Servidor com o DCEPT	192.168..15.6

A ferramenta DCEPT baseia-se no conceito de implantação de honeytokens, ou mais precisamente *honeywords*, na memória de estações de trabalho clientes de um Domínio Windows, para que os invasores possam roubar essa senha falsa e usá-la para se autenticar na rede. Quando um logon com um honeytoken é detectado um alarme é gerado para avisar os administradores. A Figura 3.2 mostra o funcionamento da solução.

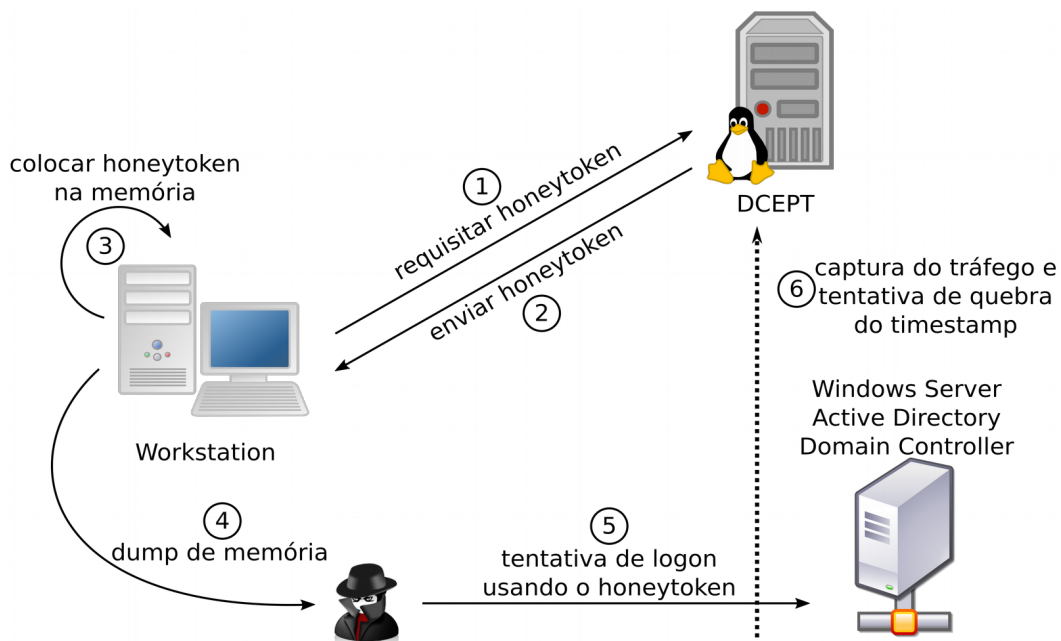


Figura 3.2: Funcionamento da ferramenta DCEPT.

A ferramenta DCEPT é composta por 3 componentes:

1. Servidor de geração de *honeypwords*;
2. Agente;
3. *Sniffer*.

O servidor de geração de *honeypwords* e o *sniffer* são codificados em Python e são instalados no servidor Linux. A principal chamada é feita ao arquivo `dcept.py` que inicia o serviço. É carregada a configuração presente no arquivo `dcept.cfg`, verificado o modo de operação (master ou slave), iniciado o servidor de geração de *honeypwords* (no caso de nó master ou conexão com o nó master no caso de nó slave) e iniciado o *sniffer*. A Figura 3.3 mostra a inicialização do serviço.

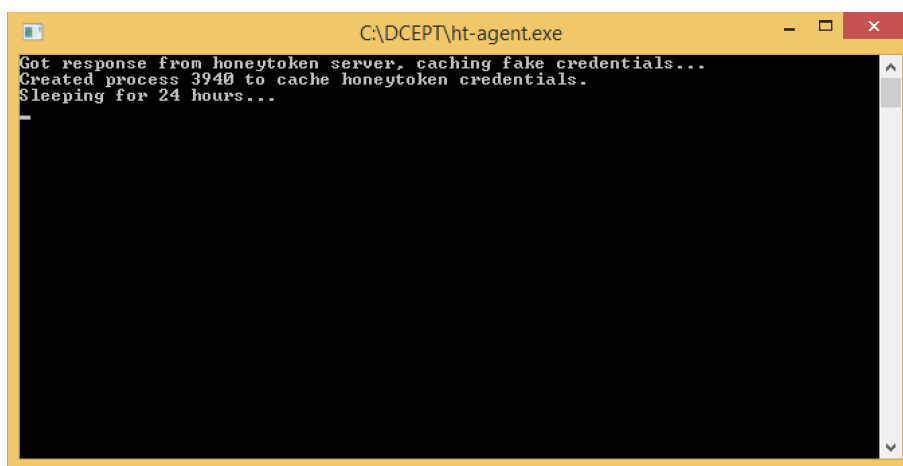
O servidor de geração é um serviço *web* que utiliza uma base de dados *SQLite* para armazenar informações sobre os *tokens*. O servidor é responsável por responder requisições HTTP do tipo GET de clientes solicitando *honeypwords*. As *honeypwords* são geradas, armazenadas na base de dados *SQLite* e enviadas ao cliente. A localização da base é fixada em código (*hardcoded*) no *script* Python e encontra-se em `/opt/dcept/var/honeytoken.db`. Durante a inicialização, se a base de dados não existir ela será criada. A base de dados possui duas tabelas apenas, a tabela `db_version`, que armazena a versão da base de dados (default 1.0), e a tabela `logs`, que armazena as informações sobre as *honeypwords*. A tabela de *logs* possui os campos `date`, `domain`, `username`, `machine` e `password`. A primeira entrada gravada na tabela de *logs* é uma *honeypword* de teste. O *password* gerado contém caracteres alfanuméricos, maiúsculos e minúsculos, e um comprimento de 10 caracteres.

```
root@ubuntu:/etc/dcept# cd
root@ubuntu:~# /opt/dcept/dcept.py

Server configured as master node
Starting DCEPT...
Database contains 4 generated passwords
Starting honeypword generation server HTTP daemon 0.0.0.0:80
kerbsniff: Looking for IC.LOCAL\Administrator on eth0
Using selector: EpollSelector
192.168.15.4 - - [05/Jul/2016 15:09:17] "GET /backup/?machine=UM-WINDOWS-81 HTTP
/1.1" 200 -
Request from IP: 192.168.15.4 workstation: UM-WINDOWS-81
Generated - NBUJQN0JhJ
Sent: {'d': 'IC.LOCAL', 'u': 'Administrator', 'p': 'NBUJQN0JhJ'}
```

Figura 3.3: Servidor de geração de *honeypwords* e *sniffer* inicializados. O servidor de geração responde ao pedido de uma *honeypword* pelo cliente.

Na estação de trabalho Windows é instalado o agente, um executável. O agente se conecta via HTTP ao módulo de geração de *honeypwords* do servidor DCEPT, solicita uma *honeypword*, injeta a *honeypword* em memória e dorme por 24 horas, quando então reinicia o processo. O agente é codificado em C# e deve ser compilado antes de ser distribuído para as estações de trabalho. No agente é codificada uma URL base, que será usada para a conexão com o servidor de geração de *honeypwords* e que deve ser alterada para apontar para o endereço IP ou entrada de DNS do servidor de geração de *honeypwords*. Também é codificado um parâmetro para a solicitação GET do HTTP, cujo padrão é o nome da máquina (pode ser deixado dessa forma). O código do agente é simples e não é necessário um conhecimento C# avançado para inspeção. A Figura 3.4 mostra a execução do agente.



```
C:\DCEPT\ht-agent.exe
Got response from honeypword server, caching fake credentials...
Created process 3940 to cache honeypword credentials.
Sleeping for 24 hours...
```

Figura 3.4: Agente conecta ao serviço, recebe um honeypword, coloca em memória e dorme por 24 horas.

O módulo *sniffer* do servidor DCEPT procura por requisições para o Serviço de Autenticação (requisições do tipo AS-REQ) do protocolo Kerberos, com dados de pré-autenticação. Assim, para que ele funcione, o servidor DCEPT tem que estar perto do DC, e ser capaz de capturar a comunicação dos clientes com o DC. Nesse experimento isso foi alcançado colocando a máquina virtual em modo promíscuo, podendo assim capturar as comunicações com o DC.

Os dados de pré-autenticação de um AS-REQ são compostos de um carimbo de tempo cifrado. Para cada AS-REQ com dados de pré-autenticação, o DCEPT enfileira um trabalho que cria uma lista de palavras (um dicionário) a partir do banco de dados de *honeypwords* geradas previamente (e distribuídas) salvos na base de dados SQLite e usa outra ferramenta, John the Ripper (JtR), para decifrar o carimbo de tempo. Quando a cifra é quebrada, é sabido que uma *honeypword* foi usada na tentativa de autenticação (supondo, razoavelmente, que *honeypwords* não irão colidir com a senha de administrador de rede) e enviado um e-mail de alerta para administradores. Como na base de dados



de *honeypots* há a informação sobre data e nome de *host* é possível saber qual estação de trabalho foi comprometida e o período de tempo em que a credencial foi extraída.

### 3.4. VALIDAÇÃO DO AMBIENTE

Foi relativamente difícil conseguir colocar o DCEPT funcionando em um ambiente de máquina virtual simples e controlado. Pode ser difícil conseguir que ele funcione corretamente em uma rede heterogênea, ampla ou de produção.

Inicialmente, a ferramenta é fornecida com um *container* Docker preparado para facilitar o processo de implantação. Infelizmente, a compilação e inicialização do *container* apresentaram alguns erros. Depois de um tempo, foi decidido fazer a instalação do DCEPT em uma máquina virtual usando o VirtualBox, assim como o DC e a estação de trabalho. Além disso, dessa forma, é possível aproveitar a rede compartilhada para que o DCEPT possa capturar o tráfego através do DC de forma promíscua. Um *playbook* da ferramenta de gerenciamento de configuração *Ansible* foi usado para instalar o servidor DCEPT, para que este processo possa ser repetível. O código do *playbook* encontra-se no Anexo A. Um arquivo importante a ser observado no servidor é o `dcept.cfg`. Editando este arquivo se é capaz de personalizar a configuração do servidor e editar parâmetros como domínio, depuração, etc. O nível de *log* DEBUG foi usado para esta experiência, que imprime algumas mensagens na saída padrão do terminal.

Houve um problema com a compilação do agente ao tentar usar o pacote mono da distribuição Linux para fazê-lo. Um *issue* foi aberto no *site* repositório de controle de versão da ferramenta (Github) pelo usuário yoshi314 sugerindo alterar as instruções de compilação para superar o erro. No caso em questão, foi utilizado o Visual Studio Express no Windows para fazê-lo.

Depois que o servidor e o agente foram iniciados, foi realizado o testes do cenário. A *honeypot* injetada na memória da estação de trabalho é conhecida anteriormente. Ela é mostrada na saída do servidor DCEPT, quando um agente requisita uma *honeypot*, se o servidor estiver sendo executado no modo de depuração. Uma *honeypot* também pode ser obtida a partir do arquivo de banco de dados SQLite que o DCEPT usa para armazenar suas *honeypots*. Apesar disso, uma ferramenta foi usada para extrair a *honeypot* da memória, para obter um pouco mais da perspectiva do atacante. A ferramenta escolhida foi o Mimikatz. A Figura 3.5 mostra um uso básico do Mimikatz para se extrair credenciais em memória. A ferramenta apresentou a *honeypot* em texto claro, como mostra a Figura 3.6 (em destaque pelo pontilhado), que sugere que foi armazenada cifrada de forma reversível na LSA.

```
mimikatz 2.1 x64 (oe.eo)

.#####.  mimikatz 2.1 (x64) built on May  6 2016 01:28:44
.## ^ ##.  "A La Vie, A L'Amour"
## / \ ##  /* * *
## \ / ##   Benjamin DELPY 'gentilkiwi' ( benjamin@gentilkiwi.com )
'## v ##'   http://blog.gentilkiwi.com/mimikatz             (oe.eo)
'#####'                                     with 19 modules * * */

mimikatz # log
Using 'mimikatz.log' for logfile : OK

mimikatz # privilege::debug
Privilege '20' OK

mimikatz # sekurlsa::logonpasswords
```

Figura 3.5: Uso básico da ferramenta Mimikatz para extração de credenciais em memória.

```
mimikatz 2.1 x64 (oe.eo)

Domain      : UM-WINDOWS-81
Logon Server : <null>
Logon Time  : 05/07/2016 15:09:17
SID         : S-1-5-21-2952204455-1605887975-571632165-1001

msv :
[00000003] Primary
* Username : Administrator
* Domain   : IC.LOCAL
* NTLM     : ed6dcb6ebb4f5b04722c05591e8ebee1
* SHA1     : 74675c4d2204d1912fb24ad168714351f7183b6c
tspkg :
* Username : Administrator
* Domain   : IC.LOCAL
* Password : N8UJQN0JhJ
wdigest :
* Username : Administrator
* Domain   : IC.LOCAL
* Password : <null>
livessp :
kerberos :
* Username : Administrator
* Domain   : IC.LOCAL
* Password : N8UJQN0JhJ
ssp : RO
credman :
```

Figura 3.6: *Honeyword* extraído da memória com a ferramenta Mimikatz, em destaque pelo pontilhado.

Com a *honeyword* extraída, um logon interativo foi tentado. A saída do serviço mostra que o módulo DCEPT *cracker* não foi capaz de decifrar o carimbo de tempo codificado e terminou com uma mensagem “no password hashes loaded”, apresentada na Figura 3.7.

```

kerb-as-req for domain user IC\administrator
PA-ENC-TIMESTAMP: None
Ignoring kerberos packet - Not kerb-as-req
kerb-as-req for domain user IC\administrator
PA-ENC-TIMESTAMP: 2d649777bf943b0ea9815634996eecab43c28bd5aeea53f7b21bf a2c9d9d39
965ba4a2b529fd98decf738fdd775b6cf06baea774
Cracker enqueued 1 encrypted timestamp. Queue size: 1
Ignoring kerberos packet - Not kerb-as-req
Recovering password from encrypted timestamp...
Testing 5 password(s)
Ignoring kerberos packet - Not kerb-as-req
Ignoring kerberos packet - Not kerb-as-req
Cracking job completed
No password hashes loaded (see FAQ)

```

Figura 3.7: Erro "no password hashes loaded".

Inspecionando o código do módulo de quebra de cifra (*cracker*), descobriu-se que a mensagem vem da chamada ao JtR. O trecho de código está na Figura 3.8. A mensagem de erro quer dizer que a lista de palavras (dicionário) gerado pelo módulo *cracker* do DCEPT não foi reconhecido pelo John. No módulo em questão a chamada para o JtR espera que o formato do arquivo de *hashes* seja KRB5PA-SHA1 (opção `--format` do JtR). O formato KRB5PA-SHA1 no JtR é responsável pela decodificação de tipos de criptografia (*etypes*) Kerberos 17 (AES128-CTS-HMAC-SHA1-96) e 18 (AES256-CTS-HMAC-SHA1-96). Uma entrada num arquivo de *hash* para o formato KRB5PA-SHA1 do JtR deve possuir o seguinte formato: `$formato$etype$usuário$domínio:$salt$cifra`. Um exemplo desse formato, extraído do DCEPT após uma leve alteração do código é: `$krb5pa$18$Administrator$IC.LOCAL$be4d2148dfb5f998ac5a367e2c341b6676463df9a8dd770634f584bb5e7def6c72cab6efb52f006d4a84b53adc801df0c504956c`. Nesse exemplo o *salt* não foi especificado.

```

result = subprocess.check_output("/opt/dcept/john --wordlist=%s --pot=%s --format
=krb5pa-sha1 %s %s" % (wordPath, potPath, passPath, redirectStr), shell=True)

```

Figura 3.8: Chamada ao John com o formato KRB5PA-SHA1 esperado para o arquivo de hashes.

Capturando a comunicação entre o DC e a estação de trabalho foi possível ver que o *etype* enviado pela estação de trabalho era o 23 – RC4-HMAC, como visto na Figura 3.9. O código do DCEPT foi alterado manualmente mudando o *etype* de 18 (AES256) para 23 (RC4-HMAC). Também foi removido o parâmetro `--format` da chamada ao JtR, para que o JtR reconhecesse o formato do arquivo de *hashes* automaticamente, conforme sugerido no *FAQ* da ferramenta. Para que as alterações surtam efeito é preciso remover os arquivos com a extensão `.pyc` e reiniciar o serviço, terminando o processo (`dcept.py`) e reexecutando-o.

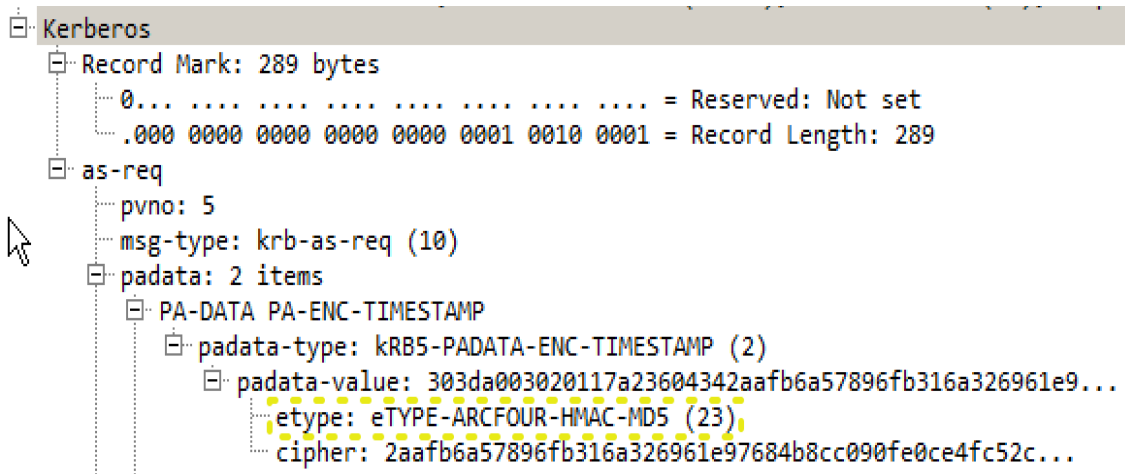


Figura 3.9: *timestamp* cifrado com o *encryption type* RC4-HMAC (23) na requisição AS-REQ enviada pelo cliente.

As alterações foram suficientes para fazer com que a mensagem “no password hashes loaded” desaparecesse. Apesar disso, JtR não foi capaz de quebrar o carimbo de tempo cifrado. Conforme mostra a Figura 3.10. Na figura é possível ver que o serviço identificou uma AS-REQ do usuário IC\Administrator, seguido do valor do carimbo de tempo. O *etype* 23 no arquivo de *hashes* é identificado corretamente pelo JtR. São usadas 7 *honeypwords* para a quebra do carimbo de tempo. A chamada ao John completa, mas sem a quebra esperada do carimbo de tempo.

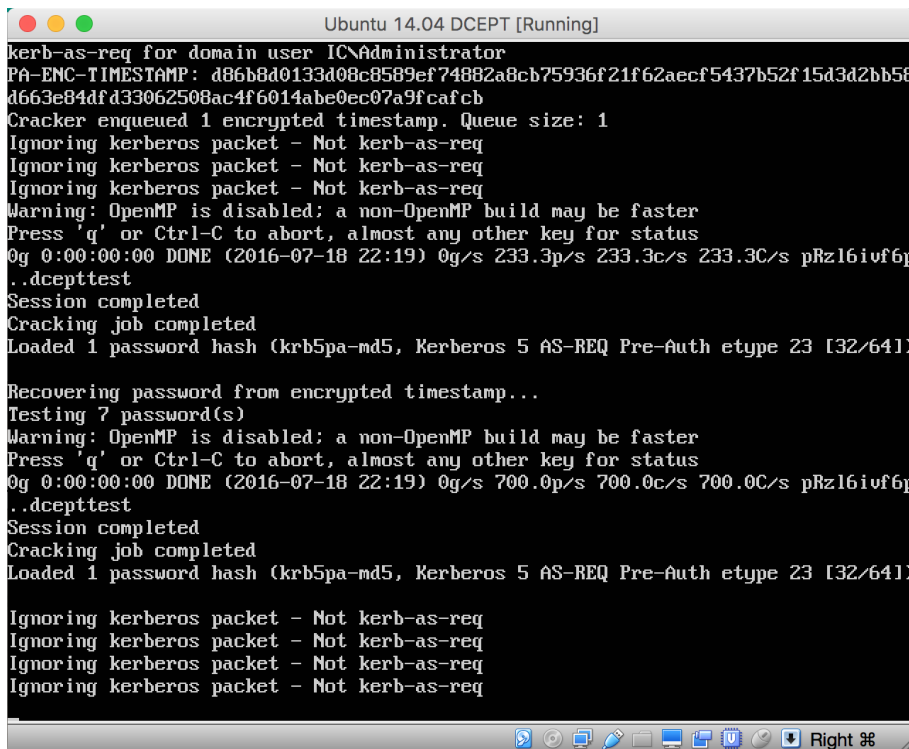


Figura 3.10: Erro "no password hashes loaded" resolvido.

Algumas hipóteses para o JtR não ter quebrado o carimbo de tempo cifrado foram levantadas:

1. algum problema desconhecido com a ferramenta JtR;
2. o *salt* e/ou *realm* estarem incorretos (embora nenhum *salt* fora especificado pelo DC durante a troca de mensagens);
3. usar o *hash* ntlm no lugar da senha.

Não foi possível confirmar nenhuma dessas hipóteses. Para endereçar o problema 1 foi feita uma tentativa de usar outra ferramenta para a quebra da cifra, o Hashcat. Não foi obtido sucesso em usá-lo em diferentes sistemas operacionais e computadores (Ubuntu, OS X e Windows). No segundo caso tentou-se usar algumas combinações de *salt* e *realm*, como mostrado na Tabela 3.2, sem resultados. Usar o *hash* NTLM das *honeypwords* no lugar das próprias *honeypwords* também se mostrou infrutífero.

Tabela 3.2: Combinações de *Realm* e *Salt* testadas.

<b>Realm</b>	<b>Salt</b>
IC	(vazio)
IC	ICAdministrator
IC.LOCAL	(vazio)
IC.LOCAL	IC.LOCALAdministrator

Para contornar este problema, foi tentado configurar o DC e a estação de trabalho para usarem apenas o tipo de criptografia AES256 (*etype* 18), como é esperado pelo código fonte do DCEPT. O AD armazena informações sobre tipos de criptografia em diver lugares, tais como:

1. Tipo de criptografia da conta do usuário;
2. Tipo de criptografia da conta do computador;
3. Tipos de criptografia permitidos na política de grupo local.

Todos esses locais foram verificados para se tentar garantir o uso do AES256. A Figura 3.11 mostra um exemplo de configuração alterada usando a ferramenta `adsis.exe`. A propriedade `msDC-SupportedEncryptionTypes` recebeu o valor hexadecimal `0x10`, conforme explica Sun (2011). Outros possíveis valores para a propriedade são mostrados na Tabela 3.3. Os valores podem ser somados para suportar múltiplos *etypes*.

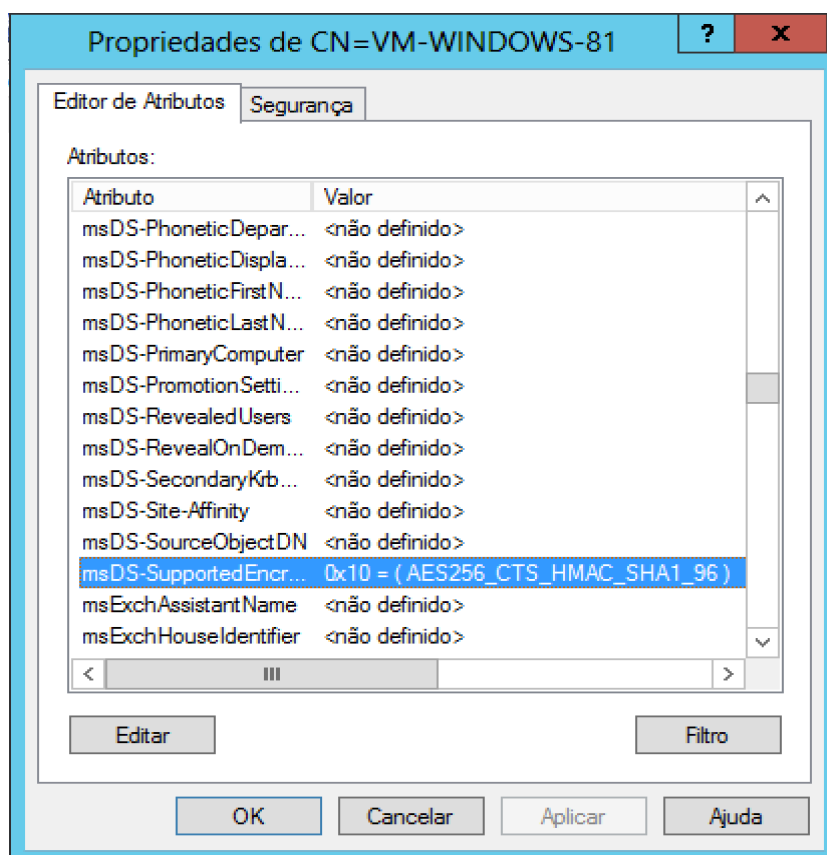


Figura 3.11: Editando a propriedade msDC-SupportedEncryptionTypes com o valor 0x10 para garantir o uso do AES256.

Tabela 3.3: Possíveis valores para a propriedade msDC-SupportedEncryptionTypes.

Tipo de Criptografia	Valor
DES-CBC-CRC	0x01
DES-CBC-MD5	0x02
RC4-HMAC	0x04
AES128-CTS-HMAC-SHA1-96	0x08
AES256-CTS-HMAC-SHA1-96	0x10

Depois de algum tratamento com as diretivas de grupo do AD para a estação de trabalho e conta de usuário, foi observado o erro `KRB5KDC_ERR_ETYPE_NOSUPP`, na resposta dada pelo AS à requisição AS-REQ. É possível ver o erro na captura mostrada na Figura 3.12, na entrada de número 10.

Após uma redefinição de senha de administrador do domínio (conta utilizada pelo *honeypot*), uma nova tentativa de logon com a *honeyword* atingiu o servidor do DCEPT, foi quebrada pelo JtR e acionou o alerta de e-mail para administradores.

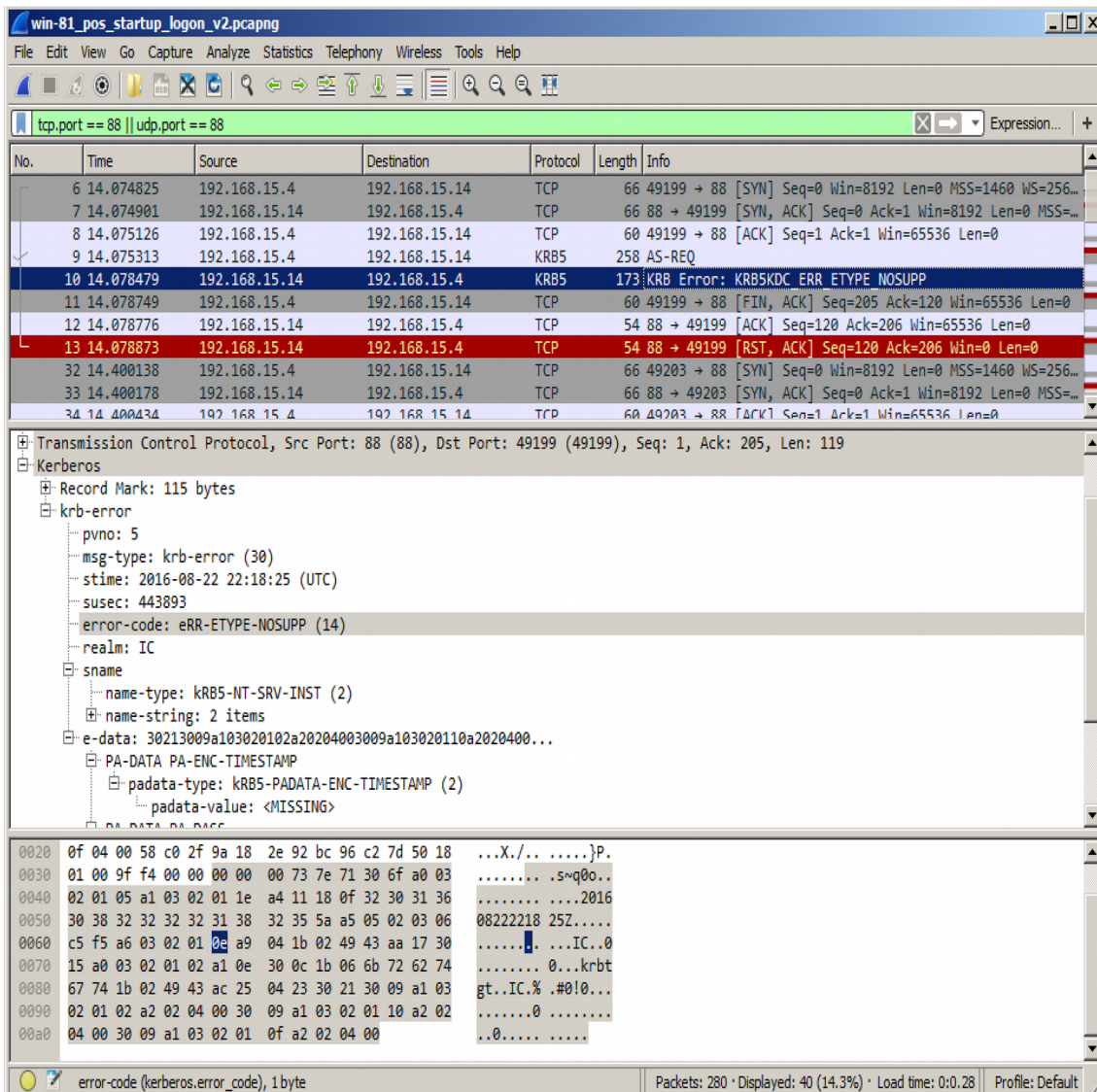


Figura 3.12: Erro KRB5KDC\_ERR\_ETYPE\_NOSUPP.

### 3.5. DISCUSSÃO E ANÁLISE

Uma vantagem do DCEPT é que ele é simples e direto. No lado do cliente um agente pré-compilado é capaz de fazer o trabalho. É livre e de código aberto sob a GNU *General Public License*. A ideia de um *honeypot* para capturar intrusos parece sólida. Sendo uma prova de conceito, pode atrair administradores de sistema a repensar os problemas de ataques PtH&T. Outra vantagem é que ele é capaz de identificar a máquina comprometida, aquela de onde a senha foi furtada, e também o período de tempo em que foi furtado, por meio da data de geração da *honeyword*, ao custo de poder de processamento, já que é usado um banco de dados de *honeywords* para decifrar um carimbo de tempo. Este recurso é interessante a partir de um ponto de vista da análise forense e também de recuperação do recurso exposto.

Quanto a classificar o DCEPT como uma ferramenta de *honeypot* de produção ou de pesquisa, por design, o *honeypot* não permite que o atacante se autentique no domínio. Isso pode limitar a capacidade de entender a motivação (o que o atacante estava procurando) e estudar o comportamento dele. Esta característica pode não ser desejável em um ambiente de produção, mas sim em um de pesquisa. Esta peculiaridade sugere que é mais direcionada a sistemas de produção. De forma adversa, o fato de que trata apenas o tipo de criptografia AES256 (*etype* 18) na sua forma atual (*hardcoded*) e que por isso pode ser difícil de implementá-lo em um ambiente heterogêneo sugerem que seria mais apropriado para uso em um ambiente controlado como uma *honeynet*, em direção a um cenário de pesquisa.

Algo notado é um poder de processamento extra, necessário para o trabalho de decifragem usando o JtR. relativamente ao tamanho da rede de clientes, o crescimento do banco de dados de *honeypots* e os pedidos para o Serviço de Autenticação (AS) no KDC. Em uma análise simplista, o banco de dados teria um aumento linear de uma entrada por estação de trabalho cliente por dia, uma vez que o tempo padrão de geração de palavras-chave é de 24 horas. Assim, para uma rede cliente com 300 computadores executando os agentes, as entradas do banco de dados de *s* aumentaria de 300 no dia um para 2100 entradas no dia 7. Em um ano ter-se-ia 109.500. Esse banco de dados é usado para criar o arquivo de dicionário que é usado pelo processo enfileirado para decodificação do carimbo de tempo nas AS-REQ com dados de pré-autenticação.

Uma desvantagem é que o texto em claro apresentado pela ferramenta Mimikatz dá uma visível indicação para um atacante que pode se tratar de uma armadilha, uma *honeypot*. A *honeypot* tem um comprimento fixo de dez pseudoaleatórios caracteres alfanuméricos, algo que dá a sensação de que não é uma senha comum para um usuário, talvez até mesmo para um administrador de domínio.

Adicionalmente, usar somente uma conta como *honey* pode limitar a capacidade de detecção. Em verdade, é uma conta que atrairia um atacante (conta *Administrator*). A colocação de vários *honeypots* no meio de credenciais verdadeiros poderia dificultar a vida de um invasor.

Ao analisar os detalhes internos de DCEPT nota-se que ele não é capaz de tratar ataques PtT, ou melhor, ele não visa atacantes que usam tíquetes uma vez que não é projetado para injetar tíquetes falsos em memória.

Outras características menores podem melhorar DCEPT de algumas maneiras. Definitivamente, seria desejável carregar o tipo de criptografia apropriado, dependendo daquele escolhido pelo cliente na troca de mensagem com o AS. Poderia ser estendido para integrar com ferramentas externas de monitoramento. Além disso, seria interessante ter a possibilidade de interface com



diferentes programas de quebra de senha (*cracking software*), diferente do John, como o Hashcat, embora não seja difícil fazer isso alterando o código. Talvez a comunicação entre o servidor DCEPT e as estações de trabalho pudesse ser cifrada para que ela não seja capturada por um invasor. Alguma forma de remoção de credenciais de autenticação anteriores da memória do computador de um cliente poderia ser usada em conjunto com DCEPT.

A Tabela 3.4 apresenta um breve resumo dos pontos observados:

Tabela 3.4: Resumo das observações sobre a ferramenta DCEPT.

<b>Item</b>	<b>Vantagem</b>	<b>Desvantagem</b>	<b>Neutro</b>
Identifica a máquina comprometida	X		
Identifica o intervalo de tempo de comprometimento	X		
<i>Honeytoken</i> identificável em inspeção		X ( <i>honeyword</i> em texto claro)	
Custo extra de processamento (ataque de dicionário para quebra da criptografia)			X
Suporte a múltiplos tipos de criptografia		X	
Quantidade de <i>honeytokens</i> implantados			1
Visa ataques PtH	X (via <i>honeywords</i> )		
Visa ataques PtT		X	
Detecção gera falha na autenticação/acesso ao serviço para o atacante.		X	

## 4. PROPOSTAS DE MELHORIAS

Mirando o objetivo de propor melhorias para a detecção de ataques PtH&T com uma abordagem de uso de *honeytokens*, foram pensadas melhorias para os problemas observados no Capítulo 3. Com isso é feita uma proposta de arquitetura e requisitos para a implementação de uma ferramenta mais ampla para a detecção de *honeytokens*. Alguns testes são feitos para verificar a viabilidade tecnológica para tal ferramenta, isto é, se um projeto com os requisitos e arquitetura ora apresentados serão exequíveis segundo o que existe. Além disso, são propostos dois elementos que, quando estudados, podem auxiliar no combate a ataques PtH&T. Esses elementos são uma *Kill Chain* para esse tipo de ataques e uma classificação das abordagens estudadas conforme observadas na literatura.

Além das ferramentas utilizadas no Capítulo 3, outras ferramentas foram acrescentadas ao cenário de testes deste capítulo. Essas ferramentas foram necessárias em testes que visam garantir um encaminhamento de tráfego e edição de código. As ferramentas adicionais são o Dnsmasq, Socat, Git e SSHFS. Mais detalhes sobre essas ferramentas podem ser encontrados no Anexo C.

### 4.1. PROPOSTA DE ARQUITETURA PARA UMA FERRAMENTA DE DETECÇÃO DE HONEYTOKENS

A proposta de uma abordagem com o uso de *honeytokens* para a detecção de ataques PtH&T parece uma ideia sólida. Porque a localização das senhas, na memória das estações de trabalho, é um local de difícil acesso. Portanto é necessário um esforço relativamente grande para obtê-las. Algo que um usuário normal não faria. Além disso, o uso de credenciais falsas, como *honeytokens*, mostra um aspecto de uma conduta ilícita, típica de um intruso. Por isso, o intuito é usá-la e onde possível acrescentar melhorias que possam ampliar a capacidade de confundir um atacante e detectá-lo.

#### 4.1.1. Injeção de honeyhashes e honeytickets

Um ponto que chamou a atenção durante a análise foi como a *honeyword* era de fácil identificação, quando um despejo de memória trouxe as senhas em texto claro, devido a ela ser uma *string* de tamanho fixo composta de caracteres alfanuméricos de comprimento 10. Como contribuição para a melhoria disso é sugerido que o agente trabalhe de forma parecida com a ferramenta Mimikatz, que injeta *hashes* e tíquetes em memória. No lugar da injeção de *honeywords* possam ser injetados *honeyhashes* para melhorar o disfarce do engodo. De forma complementar, também podem ser injetados tíquetes, aumentando a quantidade de objetos que servirão como isca para um atacante.

#### 4.1.2. Uso de Vários Honeytokens

Outra escolha que pode ser tomada é usar apenas um *honeyhash* por conta, invés de serem usados vários. Assim evita-se a criação de diversos processos de decifragem com um dicionário de palavras que tende a crescer com o tempo. Desta forma, quando um autenticador atinge o módulo apropriado para detecção ele é capaz já a partir da conta de saber qual *honeyhash* deve ser usado para decifrar o carimbo de tempo do autenticador. Além disso, é importante que haja um trabalho de pré-configuração no sentido de clonar os nomes de usuários, como mais uma medida para despistar, dificultando que um atacante diferencie usuário que são da organização dos não existentes.

#### 4.1.3. Detecção de Golden Tickets

Há uma ideia interessante para a detecção de *Golden Tickets* que estejam em vigor na rede. A ideia é usar a chave da conta KRBTGT do verdadeiro DC no falso KDC. Melhor colocando, a chave do verdadeiro DC deve ser exportada e em seguida redefinida. A chave exportada pode ser usada no falso KDC de forma que, quando um *Golden Ticket* TGT atinja o módulo de detecção, o módulo seja capaz de decifrar o tíquete, gerando um alarme. Deve-se ter atenção aí para que tíquetes válidos não sejam incorretamente detectados devido a intervalo de validade (padrão 10 horas).

#### 4.1.4. Suporte a Múltiplos Tipos de Criptografia

Ainda, observa-se que o uso de apenas um tipo de criptografia pode limitar a ferramenta e impedi-la de trabalhar com outras estações que não ofereçam suporte àquele tipo. Por isso é importante que uma ferramenta de detecção trabalhe com múltiplos *etypes*.

#### 4.1.5. Continuação do Protocolo em Caso de Detecção

Ao estudar os trabalhos sobre *honeywords* na Seção 2.5.5 viu-se o conceito de *honey encryption*. Na *honey encryption* a ideia é não gerar um erro quando um processo de decifragem falhar. Ao contrário, é gerado um texto em claro como se o processo tivesse dado certo, confundindo assim o atacante. Nessa mesma linha, seria interessante que uma requisição não gerasse um erro, mas sim, uma resposta engodo com outro *honeyticket*. Para isso, é preciso que o módulo de captura de pacotes (*sniffer*), responsável pela filtração de requisições, não seja passivo. Ele precisa ativamente receber o pacote com a requisição, detectar um *honeytoken* e enviar outro *honeytoken* como resposta, isto é, um *honeyticket*. Já foi visto que isso é possível no experimento de encaminhamento de mensagens Kerberos na Seção 4.4.2. Nesse sentido, a ferramenta se comporta como um falso (pseudo/honey) KDC. Repare que nesse caso não basta enviar somente o *honeyticket*. É necessário

enviar a resposta apropriada, dependendo da requisição (AS ou TGT), junto com o *honeyticket*. Se o *honeytoken* não for detectado, a requisição deve ser encaminhada para o KDC verdadeiro no Controlador de Domínio. Esse mesmo mecanismo de encaminhamento pode ser usado para as requisições AP-REQ. Uma parte do sistema pode ser responsável por se integrar com outras ferramentas de monitoramento.

#### 4.1.6. Resumo das Funcionalidades e Arquitetura

Conforme exposto, segue um breve resumo dos requisitos importantes para uma ferramenta de detecção, visando atender os pontos fracos estudados:

1. Devem ser injetados em memória tanto *honeyhashes* como *honeytickets*;
2. Deve usar somente o *honeyhash* do usuário para decifrar o carimbo de tempo;
3. A ferramenta não deve gerar falha em caso de detecção, se comportando como um sistema verdadeiro;
4. Deve suportar múltiplos tipos de criptografia do Kerberos;
5. Devem ser usados vários *hashes* e *tíquetes*;

Desta forma, baseado nas propostas dos capítulos anteriores, uma possível arquitetura pode ter os seguintes componentes:

1. Base de dados para armazenamento de contas de usuários, *honeyhashes* e *honeytickets*;
2. Agentes nas estações de trabalho do domínio, responsáveis por injetar *honeyhashes* e *honeytickets* em memória;
3. Módulo que responde a requisições de *honeytokens* (*honeyhashes* e *honeytickets*);
4. Módulo de filtragem, responsável por filtrar requisições AS-REQ, TGS-REQ e AP-REQ e encaminhamento, colocado “no meio do caminho” do KDC ou AP;
5. Módulo para fazer a detecção de HH&T nos pacotes filtrados e encaminhamento:
  - a) se for HH&T, devolver à filtragem para encaminhar resposta ao cliente (AS-REP, TGS-REP ou AP-REP) e enviar mensagem de alarme ao módulo de tratamento;
  - b) se não for HH&T, devolver à filtragem para encaminhar para o KDC ou AP;
6. Módulo para tratar o evento de alarme quando um HH&T é detectado: possui o propósito de integração com outras ferramentas de monitoramento.

A Figura 4.1 mostra a arquitetura com seus componentes em um cenário simples, composto por uma estação de trabalho, um controlador de domínio e um serviço.

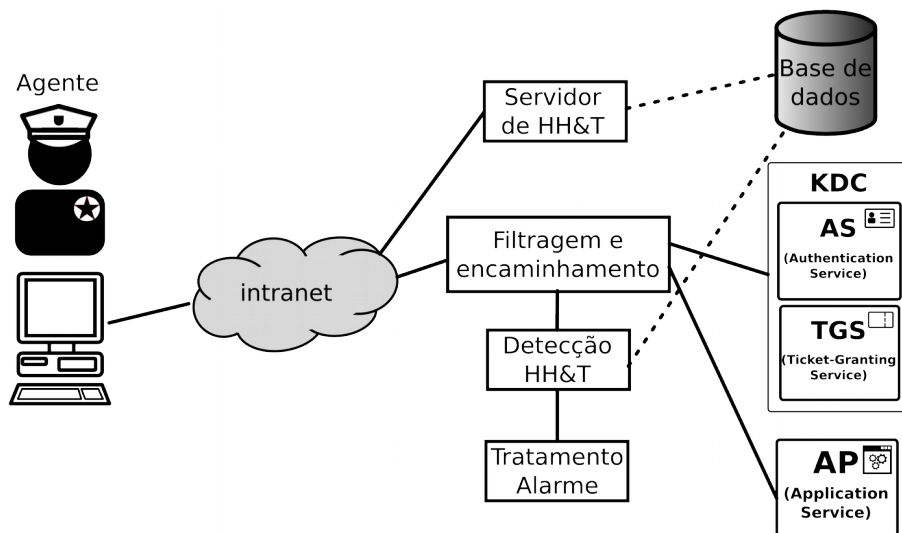


Figura 4.1: Arquitetura proposta para uma ferramenta de detecção de honeytokens.

A Tabela 4.1 apresenta um comparativo com o que foi encontrado no estudo do DCEPT e o que é colocado na proposta.

Tabela 4.1: Comparativo entre a proposta e a ferramenta DCEPT.

Id	Item	Vantagem		Desvantagem		Neutro	
		DCEPT	Prop	DCEPT	Prop	DCEPT	Prop
1	Identifica a máquina comprometida	sim			não		
2	Identifica o intervalo de tempo de comprometimento	sim	Sim para tíquetes		não para hashes		
3	Honeytoken identificável em inspeção		não	sim (honey word em texto claro)			
4	Exige custo extra de processamento (ataque de dicionário para quebra da criptografia)		não			sim	
5	Suporte a múltiplos tipos de criptografia		sim	não			
6	Quantidade de honeytokens implantados					1	vários
7	Visa ataques PtT		sim	não			
8	Detecção gera falha na autenticação/acesso ao serviço para o atacante.		não	sim			

## 4.2. VIABILIDADE TECNOLÓGICA DA ARQUITETURA

Como forma de avaliar a possibilidade de execução de tal projeto, isto é, se há formas de implementar aquilo que se deseja nos requisitos e arquitetura, escolheu-se aplicar alguns testes. Optou-se por testar:

1. A injeção de *honeypashes* e *honeytickets*;
2. O encaminhamento de tráfego Kerberos;
3. O uso do tipo de criptografia correto.

## 4.3. DESCRIÇÃO DO CENÁRIO DE TESTES

Basicamente, foi reutilizada a mesma arquitetura mostrada na Seção 3.3, entretanto, alguns componentes foram removidos e outros colocados. O controlador de domínio com o sistema operacional Windows Server 2008 foi removido. No lugar dele foi instalado e configurado um servidor com o sistema operacional Windows Server 2012. Além disso, foi acrescentada uma máquina virtual com o sistema operacional Windows 7. Essa máquina foi configurada como não participante do Domínio Windows. A Tabela 4.2 mostra informações sobre as máquinas virtuais utilizadas no estudo.

Tabela 4.2: Máquinas virtuais utilizadas nos testes.

Nome de <i>host</i>	Sistema Operacional	Destinação	Endereço IP
izanagi	Windows 7		192.168.15.9
vm-windows-81	Windows 8.1	Estação de trabalho cliente	192.168.15.4
win-pdc	Windows Server 2008	Controlador de Domínio AD	192.168.15.8
ubuntu	Ubuntu Server 14.04	Servidor com o DCEPT	192.168..15.6

## 4.4. REALIZAÇÃO DOS TESTES

Na sequência estão a realização dos testes de viabilidade. No teste de injeção de *honeypashes* e *honeytickets* é utilizada a ferramenta Mimikatz para colocar as estruturas na memória de um computador não participante do domínio para obter acesso a uma listagem de diretório (*directory listing*) de um dos compartilhamentos administrativos do controlador de domínio do experimento. No teste de encaminhamento de tráfego quis-se garantir que fosse possível que as mensagens do Protocolo Kerberos passassem pelo servidor de detecção. No teste do tipo de criptografia queria-se que a ferramenta detectasse o *etype* e o usasse automaticamente.

#### 4.4.1. Teste de Injeção de Honeyhash e Honeyticket

Como forma de cumprir o requisito de injeção de *hashes* e tíquetes em memória escolheu-se fazer um teste utilizando a ferramenta Mimikatz. Essa ferramenta, além de outras coisas, possui a capacidade de produzir ataques PtH e PtT. Para isso ela usa a injeção de *hashes* e tíquetes em memória.

Nesse experimento foi simulado um ataque PtH, usando a ferramenta Mimikatz. Para dar acrescentar um algo a mais no experimento, foi utilizada uma máquina virtual que não fazia parte do domínio. Usou-se uma VM com o sistema operacional Windows 7 por já haver uma disponível para uso imediato. Foi usado o *hash* NTLM da senha do usuário `LAB\Administrador`. O ataque é mostrado na Figura 4.2.

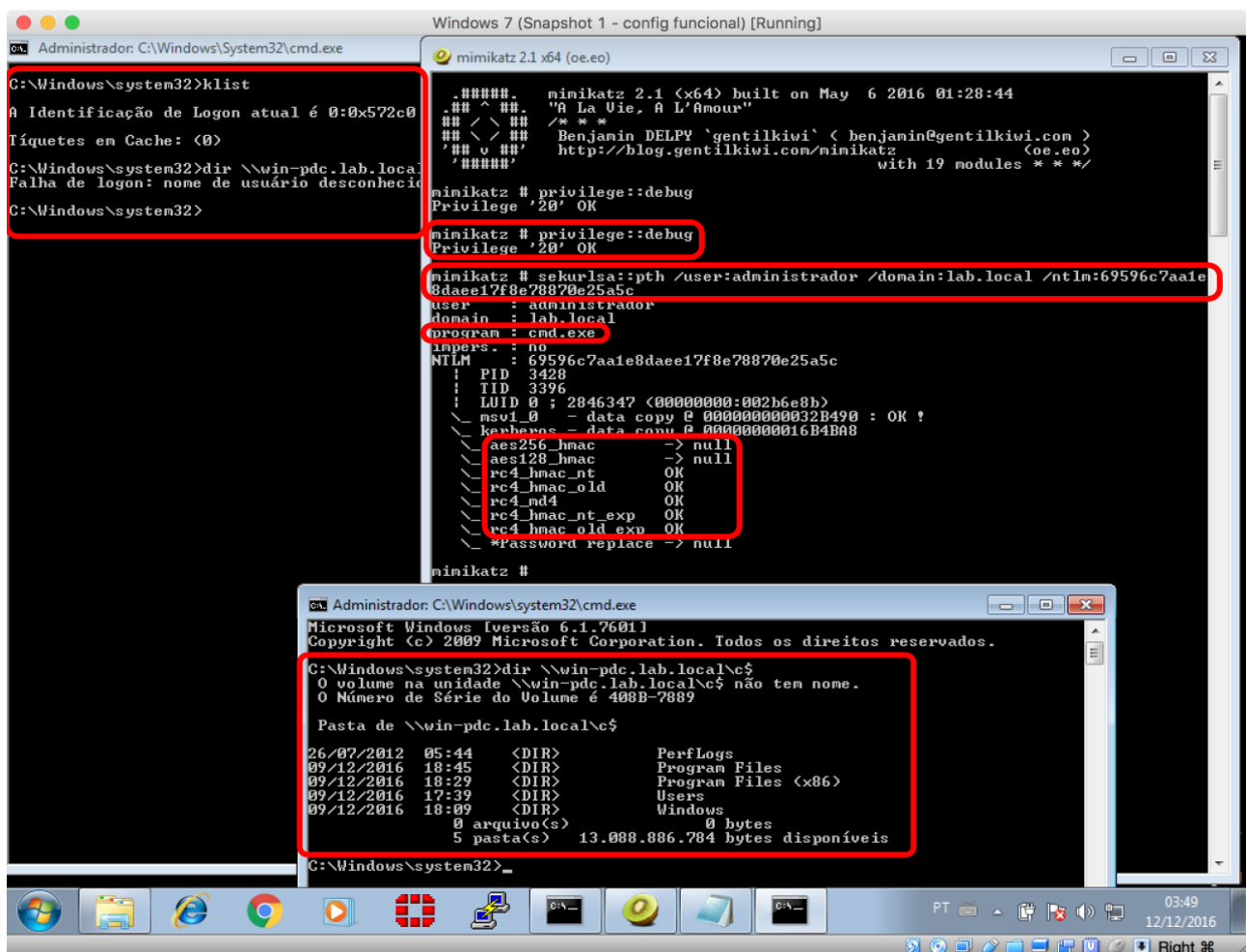


Figura 4.2: Simulação de um ataque PtH.

Pelo prompt de comando do Windows tentou-se fazer uma listagem do compartilhamento administrativo C\$, usando o comando `dir` (janela no último plano). Foi retornado um erro de falha de logon. No prompt da ferramenta Mimikatz (janela no plano do meio), é concedido privilégio de

depuração ao processo do Mimikatz, por meio do comando `privilege::debug`. O privilégio de depuração permite à ferramenta depurar um processo que, se não houvesse adquirido o privilégio, ela não teria acesso. Em seguida foi utilizado o módulo `sekurlsa` com o comando `pth`. Na figura é possível observar o carregamento das chaves RC4. O comando padrão que é utilizado (quando não especificada a opção `/run`) é um prompt (janela no primeiro plano). No prompt aberto foi executado novamente o comando `dir` e, dessa vez, foi obtido acesso ao drive C.

#### 4.4.2. Teste de Encaminhamento de Tráfego

Baseado na proposta de um componente ativo na inspeção do tráfego Kerberos, foi elaborado um experimento para verificar a possibilidade de se fazer um encaminhamento (*forwarding*) desse tráfego, na camada de transporte. Isto é feito pensando na possibilidade de se colocar um *man-in-the-middle* entre uma estação de trabalho cliente e um controlador de domínio. O intuito dessa configuração é possibilitar que o tráfego Kerberos seja filtrado, inspecionado e encaminhado conforme uma lógica de decisão.

A resolução de nomes feitas pelas estações de trabalho para descobrirem quem é o servidor de domínio da rede utilizam o protocolo DNS. É possível também que o mesmo seja feito via Netbios sobre TCP, mas o uso do DNS é o preferencial na Rede Windows. São usadas entradas de DNS do tipo SRV. As entradas são do tipo `_Service._Protocol.DnsDomainName` e há uma listagem delas no *site* Technet da Microsoft (2016c). Portanto, se quiser-se que o tráfego passe pelo servidor Ubuntu, é preciso que essas entradas sejam apontadas para ele, ao invés delas serem apontadas para o DC. A Figura 4.3 mostra *queries* DNS do tipo SRV sobre os serviços do domínio.



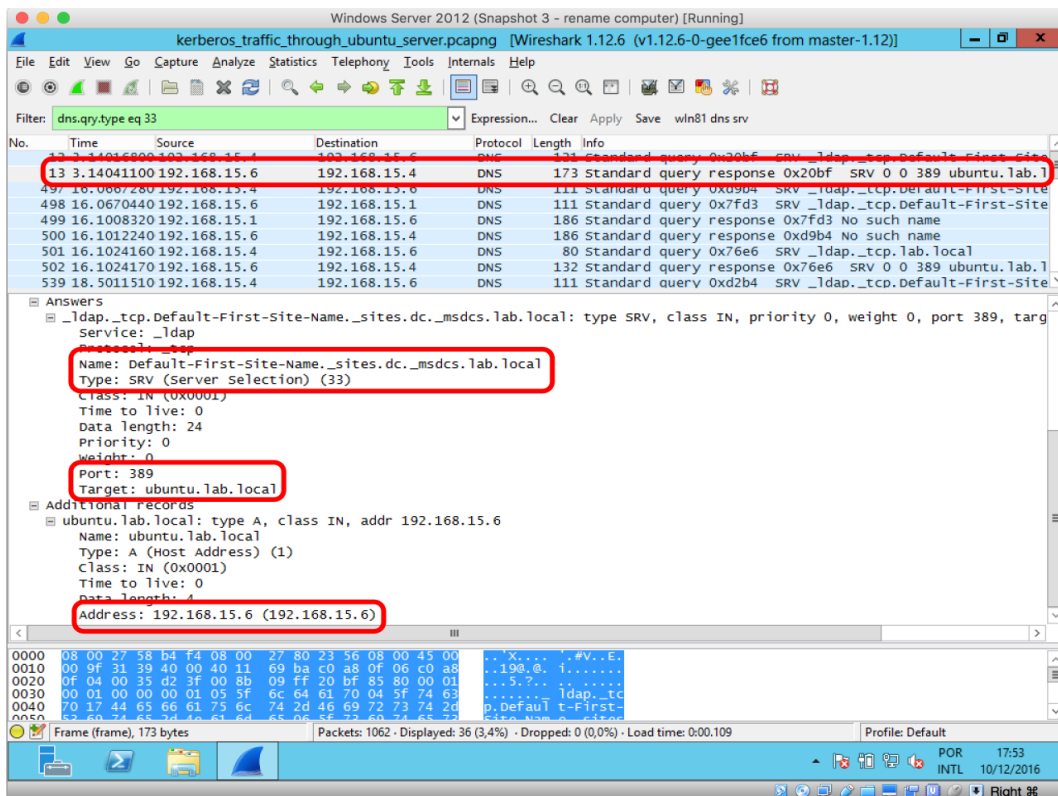


Figura 4.3: Captura de tráfego DNS com queries do tipo SRV.

Para provar a funcionalidade de servidor DNS foi utilizada a ferramenta Dnsmasq, instalada com o gerenciador de pacotes apt, da distribuição Ubuntu. A Figura 4.4 mostra as linhas acrescentadas ao final do arquivo `/etc/dnsmasq.conf`. A Figura 4.5 mostra o conteúdo do arquivo de hosts, com destaque para as entradas referentes às máquinas virtuais da rede NAT.

```
#srv-host=_kerberos._udp.lab.local,ubuntu.lab.local,88
srv-host=_kerberos._tcp.lab.local,ubuntu.lab.local,88
srv-host=_kerberos._tcp.Default-First-Site-Name._sites.dc._msdcs.lab.local,ubuntu.lab.local,88
srv-host=_kerberos-master._tcp.lab.local,ubuntu.lab.local,88
#srv-host=_kerberos-master._udp.lab.local,win-pdc.lab.local,88
#srv-host=_kpasswd._tcp.lab.local,win-pdc.lab.local,88
#srv-host=_kpasswd._udp.lab.local,win-pdc.lab.local,88
srv-host=_ldap._tcp.ubuntu.lab.local,ubuntu.lab.local,389
srv-host=_ldap._tcp.lab.local,ubuntu.lab.local,389
srv-host=_ldap._tcp.Default-First-Site-Name._sites.dc._msdcs.lab.local,ubuntu.lab.local,389
srv-host=_ldap._tcp.dc._msdcs.lab.local,ubuntu.lab.local,389
#txt-record=_kerberos.lab.local,"LAB.LOCAL"
```

Figura 4.4: Seção modificada do arquivo dnsmasq.conf.

Ajustada a questão de resolução de nomes, resta resolver a questão do encaminhamento. Para compreender melhor as mensagens trocadas entre o controlador de domínio e a estação de trabalho cliente, foi feita uma captura de tráfego e analisados os protocolos. A Tabela 4.3 mostra os principais protocolos observados.

Tabela 4.3: Principais protocolos trocados entre a estação de trabalho e o DC.

Transporte	Porta	Protocolo
UDP	53	DNS
TCP	88	Kerberos
TCP	135	EPM
TCP	389	LDAP
UDP	389	CLDAP
TCP	445	SMB

```

127.0.0.1    localhost
127.0.1.1    ubuntu

192.168.15.4 vm-windows-81.lab.local
192.168.15.6 ubuntu.lab.local dcepth.lab.local
192.168.15.8 win-pdc.lab.local
192.168.15.8 win-sep5r15fqh.lab.local

192.168.99.1 macbook.lab.local macbook

# The following lines are desirable for IPv6 capable hosts
::1        localhost ip6-localhost ip6-loopback
ff02::1    ip6-allnodes
ff02::2    ip6-allrouters
    
```

Figura 4.5: Arquivo de hosts do servidor Ubuntu.

Foi interessante perceber o uso de UDP para o Protocolo LDAP (*Lightweight Directory Access Protocol*) feita pela implementação da Microsoft, algo que é tipicamente feito sobre TCP. Esse protocolo é conhecido como Connectionless LDAP, daí CLDAP. Foi observado que esse tipo de tráfego também deve passar pelo servidor Ubuntu, além do tráfego Kerberos (que é o de fato desejado), para o correto funcionamento dos serviços do domínio.

Para realizar o encaminhamento, foi usada uma ferramenta chamada Socat, mostrado na Figura 4.6. Com ele foi feito o encaminhamento da porta TCP 88 (Kerberos) e da porta UDP 389 (CLDAP).

```

[root@ubuntu:/etc# ps axjf | grep socat
3163 3912 3911 3147 pts/0    3911 R+   0 0:00 | \ grep --color=auto socat
1 2921 2921 1301 ?        -1 S    0 0:00 socat UDP4-RECVFROM:389,fork UDP4-SENDTO:192.168.15.8:389
1 3052 3052 1301 ?        -1 S    0 0:00 socat TCP-LISTEN:88,fork TCP:192.168.15.8:88
root@ubuntu:/etc#
    
```

Figura 4.6: Processos do utilitário Socat usado para o encaminhamento.

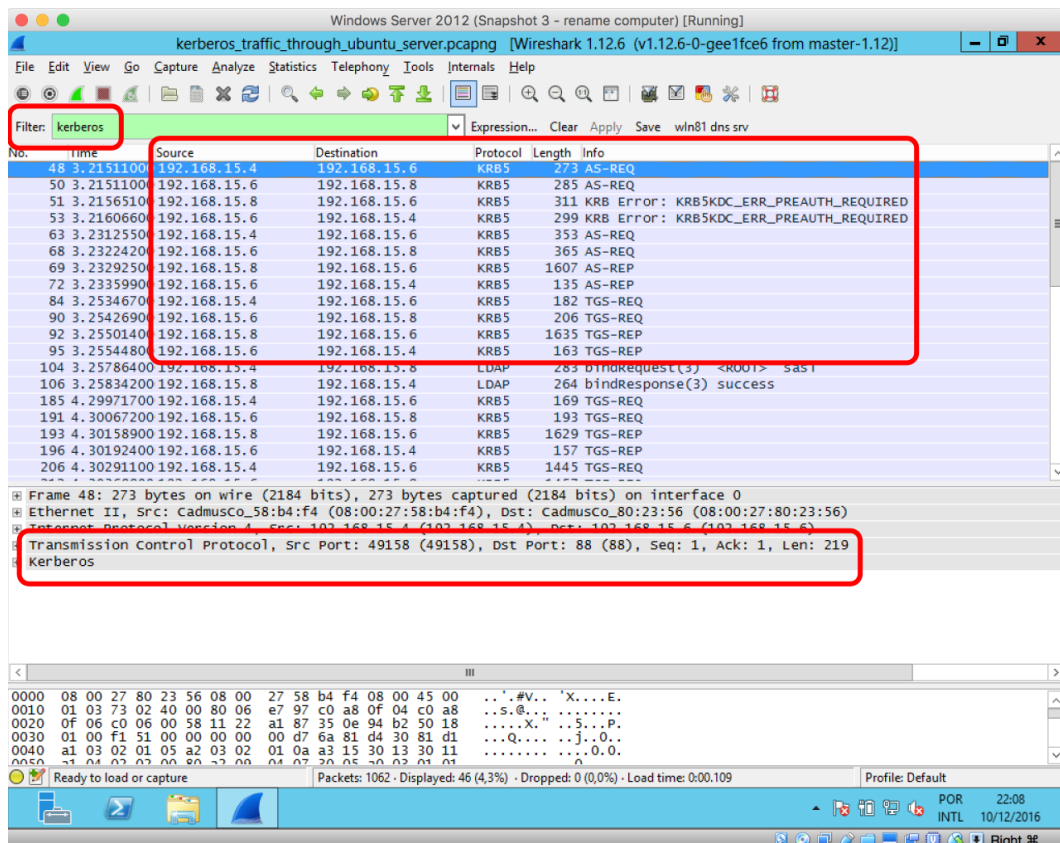


Figura 4.7: Tráfego Kerberos passando pelo servidor Ubuntu.

Com o serviço DNS e o encaminhamento prontos, foi capturado o tráfego referente ao processo de inicialização da estação de trabalho e foi possível observar o encaminhamento das mensagens Kerberos passando por dentro do servidor Ubuntu, e do servidor para o controlador de domínio. Pode-se observar as requisições AS-REQ e TGS-REQ ocorrendo na Figura 4.7. O mesmo ocorre no caso do Protocolo LDAP. Note que as mensagens TCP do Protocolo LDAP não precisam passar pelo servidor Ubuntu.

Na figura acima há destaque para o filtro usado, `kerberos`, no canto superior esquerdo. Um pouco acima do centro estão os endereços IP de origem e destino dos pacotes. É de se reparar que as mensagens estão duplicadas justamente pelo trânsito dentro do servidor Linux. É possível ver a requisição AS-REQ (números 48 e 50), partindo do cliente, de endereço IP 192.168.15.4, para o servidor, de endereço IP 192.168.15.8, que passam pelo servidor Linux, com endereço IP 192.168.15.6. As mensagens AS-REQ são seguidas de uma mensagem de erro na resposta do servidor KDC (números 51 e 53). O erro `KRB5KDC_ERR_PREAUTH_REQUIRED` quer dizer que o envio de dados de pré-autenticação faz-se necessário. O cliente requisita novamente enviando outra AS-REQ (números 63 e 68). Dessa vez o servidor responde com um AS-REP e a troca de mensagens segue conforme o esperado.

De maneira parecida ocorre nas Figura 4.8 e Figura 4.9.

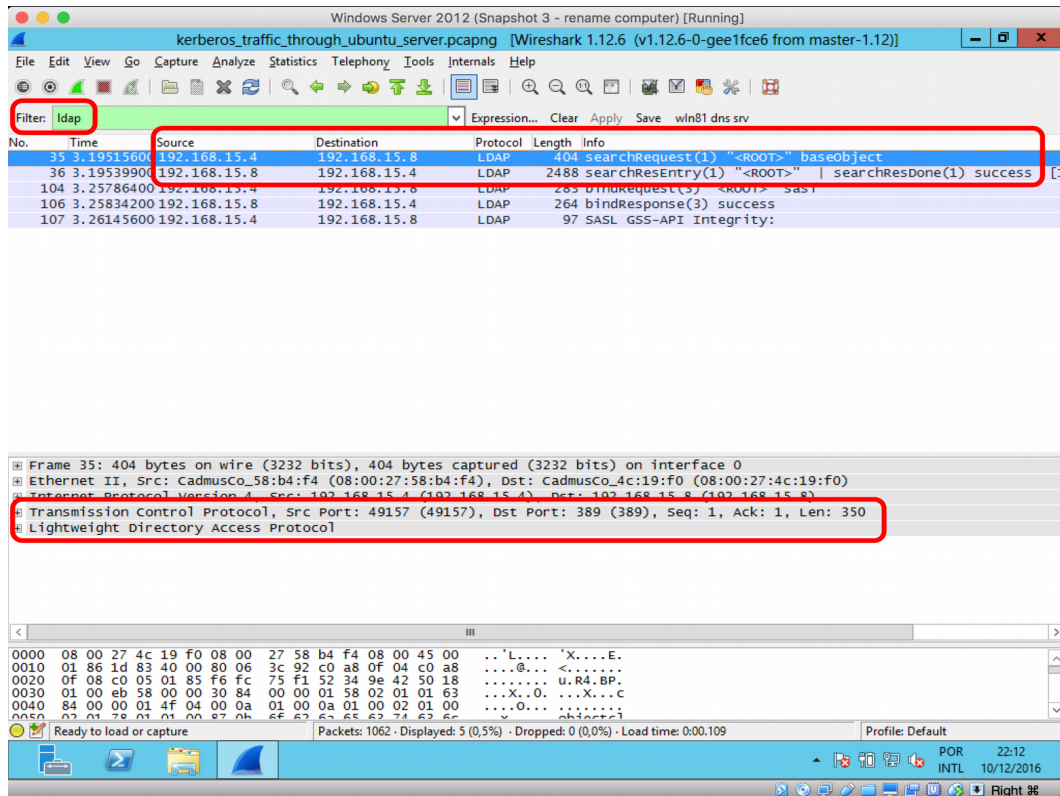


Figura 4.8: Tráfego LDAP trocado diretamente entre a estação de trabalho e o DC.

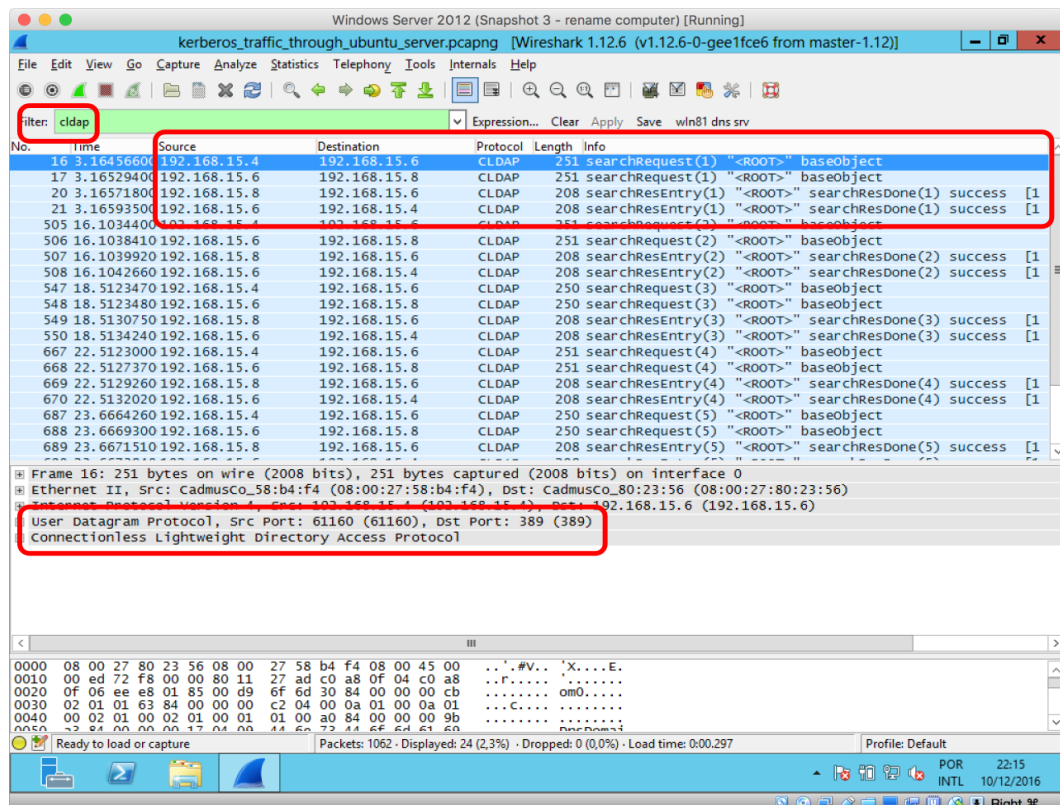
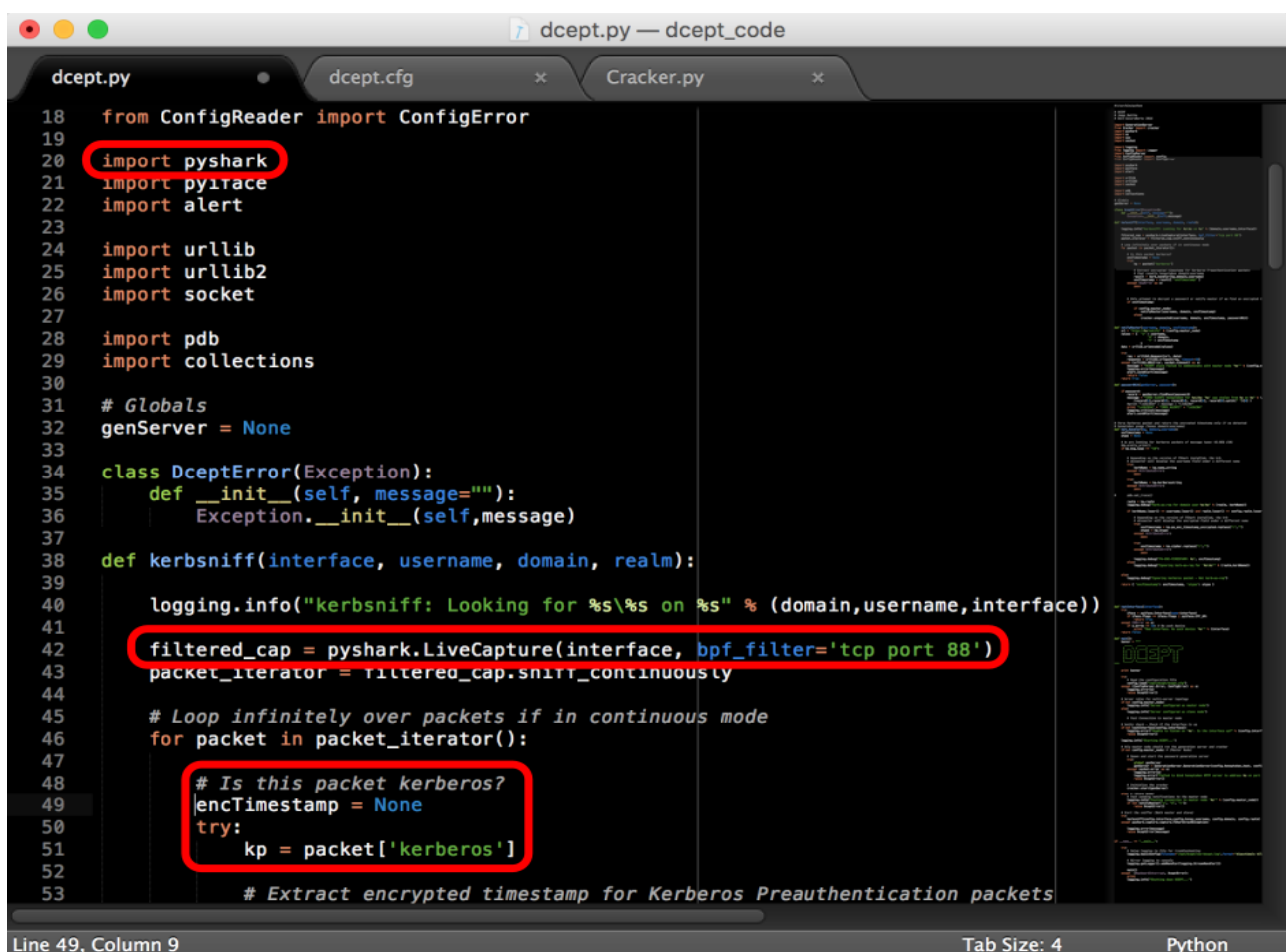


Figura 4.9: Tráfego CLDAP passando pelo servidor Ubuntu.

### 4.4.3. Teste de Uso do Tipo de Criptografia Correto

Nesse experimento o código do DCEPT é modificado para que o tipo de criptografia (*etype*) apropriado seja carregado. Um ponto importante é fazer as alterações necessárias no arquivo `dcept.cfg` para refletir as alterações de nome de usuário, domínio, etc. Outro ponto é verificar no controlador de domínio quais são os tipos de criptografia configurados para a conta a ser utilizada (LAB\Administrador no caso), por meio da ferramenta de Usuários e Computadores do Active Directory, e para o computador (VM-WINDOWS-81), com o utilitário ADSI (*Active Directory Service Interfaces*), conforme visto na Seção 3.4.

Percebe-se que é utilizado um módulo Python chamado PyShark para fazer a captura de pacotes. A Figura 4.10 mostra o trecho do código analisado. A tarefa é alterar de forma que seja possível capturar também o *etype*.



```
18 from ConfigReader import ConfigError
19
20 import pyshark
21 import pytrace
22 import alert
23
24 import urllib
25 import urllib2
26 import socket
27
28 import pdb
29 import collections
30
31 # Globals
32 genServer = None
33
34 class DceptError(Exception):
35     def __init__(self, message=""):
36         Exception.__init__(self, message)
37
38 def kerbsniff(interface, username, domain, realm):
39
40     logging.info("kerbsniff: Looking for %s%s on %s" % (domain, username, interface))
41
42     filtered_cap = pyshark.LiveCapture(interface, bpf_filter='tcp port 88')
43     packet_iterator = filtered_cap.sniff_continuously
44
45     # Loop infinitely over packets if in continuous mode
46     for packet in packet_iterator():
47
48         # Is this packet kerberos?
49         encTimestamp = None
50         try:
51             kp = packet['kerberos']
52
53             # Extract encrypted timestamp for Kerberos Preauthentication packets
```

Figura 4.10: Módulo PyShark usado para capturar o tráfego Kerberos.

Antes porém de modificar o código é preciso saber a estrutura da variável que contém o pacote Kerberos. Para isso foi utilizado o depurador pdb para Python. Foi utilizada o método `set_trace()`

para introduzir um *breakpoint* na linha anterior à atribuição da variável `kp`. Cabe lembrar que, por padrão, arquivos com a extensão `.pyc` estão sendo gerados. Esses arquivos precisam ser apagados a cada alteração no código. Para não haver preocupação com isso foi utilizada a opção `-B` e a chamada ao *script* foi feita com o comando `python -B -m pdb dcept.py`. O uso do depurador é mostrado na Figura 4.11.

Por meio do depurador pode-se ter um melhor entendimento da estrutura da variável e o valor desejado pode ser obtido por meio da propriedade `etype`. Pode-se observar que existem vários campos a disposição para a manipulação do objeto. Campos como `cname`, `hostadress`, etc.

```

[...erver --- fabio@ubuntu: ~ -- -bash      root@ubuntu: /opt/dcept --- ssh d...      fabio@ubuntu: /usr/share/dcept...  +
[root@ubuntu:/opt/dcept# python -B -m pdb dcept.py
> /opt/dcept/dcept.py(7)<module>()
-> import GenerationServer
[(Pdb) c

      D / O C E P T
    _ _ _ _ _ _ _ _ _ _

Server configured as master node
Starting DCEPT...
Database contains 7 generated passwords
Starting honeytoken generation server HTTP daemon 0.0.0.0:80
kerbsniff: Looking for LAB.LOCAL\Administrador on eth0
Using selector: EpollSelector
> /opt/dcept/dcept.py(124)kerb_handler()
-> realm = kp.realm
[(Pdb) p kp
<KERBEROS Layer>
[(Pdb) dir()
['domain', 'encTimestamp', 'kerbName', 'kp', 'username']
[(Pdb) locals()
{'username': 'Administrador', 'kerbName': 'administrador', 'domain': 'LAB.LOCAL', 'kp': <KERBEROS Layer>, 'encTimestamp': None}
[(Pdb) p dir(kp)
['_', 'DATA_LAYER', '__class__', '__delattr__', '__dict__', '__dir__', '__doc__', '__format__', '__getattribute__', '__getattribute__', '__getstate__', '__hash__', '__init__', '__module__', '__new__', '__reduce__', '__reduce_ex__', '__repr__', '__setattr__', '__setstate__', '__sizeof__', '__str__', '__subclasshook__', '__weakref__', '_all_fields', '_field_prefix', '_get_all_field_lines', '_get_all_fields_with_alternates', '_layer_name', '_sanitize_field_name', 'addr_nb', 'addr_type', 'ber_bitstring_padding', 'cname', 'etype', 'etypes', 'field_names', 'get_field', 'get_field_by_showname', 'get_field_value', 'hostaddress', 'hostaddresses', 'kdc_req_body', 'kdcoptions', 'kdcoptions_allow_postdate', 'kdcoptions_canonicalize', 'kdcoptions_constrained_delegation', 'kdcoptions_disable_transited_check', 'kdcoptions_enc_tkt_in_skey', 'kdcoptions_forwardable', 'kdcoptions_forwarded', 'kdcoptions_s_opt_hardware_auth', 'kdcoptions_postdated', 'kdcoptions_proxiable', 'kdcoptions_proxy', 'kdcoptions_renew', 'kdcoptions_renewable', 'kdcoptions_renewable_ok', 'kdcoptions_validate', 'layer_name', 'msg_type', 'name_string', 'name_type', 'nonce', 'pac_request_flag', 'padata', 'padata_type', 'padata_value', 'pretty_print', 'pvno', 'raw_mode', 'realm', 'rm_length', 'rm_reserved', 'rtime', 'sname', 'till']
[(Pdb) pp dir(kp)
['_',
 'DATA_LAYER',
 '__class__',
 '__delattr__',
 '__dict__',
 '__dir__',
 '__doc__',
 '__format__',
 '__getattribute__',
 '__getattribute__',
 '__getstate__',
 '__hash__',

```

Figura 4.11: Uso do depurador pdb para Python.

A parte relativa ao salt é um pouco mais complicada. Isso porque a informação de salt não está presente na requisição AS-REQ. Essa informação é passada pelo servidor na resposta AS-REP. Além disso, não são todos os tipos de criptografia que exigem um salt. Um mecanismo de processamento de respostas AS-REP pode ser pensado para se buscar o salt e atrelá-lo ao *principal name* da resposta, de forma que esse possa ser usado na próxima AS-REQ. Por ora, é suficiente se contentar com usar o salt padrão enviado pelo servidor, que é a concatenação do nome do domínio com o nome do usuário. Nesse caso por exemplo é `LAB.LOCALAdministrador`. Será feita uma breve checagem pelo *etype* 23, RC4-HMAC, que não requer salt. Foram utilizadas as propriedades de configuração presentes no arquivo `dcept.cfg`. O comando `git diff 3edb23b2 552a3bf2` foi utilizado para gerar as mudanças no código, apresentado na Figura 4.12. Note que o commit do *branch* (ramo) *master* estava em `3edb23b28f2af2310bffd74709f92ad27e29e56` antes das alterações. O arquivo `my.patch` encontra-se no Anexo B.

```

dcept — fabio@ubuntu: ~ — less ◀ git diff 3edb23b2 552a3bf2 — 99x55
fabio@ubuntu: ~ — less ◀ git diff 3edb23b2 552a3bf2      ...al/dcept_code — root@ubuntu: /opt/dcept — ssh dcept +
diff --git a/server/Cracker.py b/server/Cracker.py
index 9e10e6f..c97ca3a 100644
--- a/server/Cracker.py
+++ b/server/Cracker.py
@@ -12,15 +12,18 @@ import shutil
import subprocess
import time

+from ConfigReader import config
+
class Cracker:

    def __init__(self):
        self.passwordQueue = Queue.Queue(maxsize=100)
        self.thread = threading.Thread(target = self._run, args = ())
        self.thread.daemon = True
+        config.load("/opt/dcept/dcept.cfg")

-    def enqueueJob(self, username, domain, encTimestamp, callback):
-        self.passwordQueue.put((username, domain, encTimestamp, callback))
+    def enqueueJob(self, username, domain, encTimestamp, callback, etype):
+        self.passwordQueue.put((username, domain, encTimestamp, callback, etype))
        logging.debug("Cracker enqueued 1 encrypted timestamp. Queue size: %d" % (self.passwordQueue.qsize()))

@@ -30,7 +33,7 @@ class Cracker:
    # It should crack the most recent passwords working backward. In practice the
    # only time this subroutine is called is when someone uses the honeypot
    # domain\username.
-    def recoverPassword(self, username, domain, encTimestamp, callback):
+    def recoverPassword(self, username, domain, encTimestamp, callback, etype):

        tmpDir = tempfile.mkdtemp("--dcept")
@@ -38,14 +41,17 @@ class Cracker:
        wordPath = tmpDir + "/wordlist.tmp"
        passPath = tmpDir + "/encPass.tmp"
        potPath = tmpDir + "/john.pot"
+        salt = ""
+        if etype != "23":
+            salt = config.domain + config.honey_username

        logging.debug("Recovering password from encrypted timestamp...")

        # Create password file for cracking tool
        fh = open(passPath, 'w')
-        fh.write("$$$$$$$" % ("krb5pa",18,username, domain, encTimestamp))
+        fh.write("$$$$$$$" % ("krb5pa", etype, username, domain, salt, encTimestamp))
        fh.close()

+        # Create word list of the generated passwords ordered by most recent
:

```

Figura 4.12: Parte das alterações realizadas no código da ferramenta DCEPT.

Por fim, foi adicionada manualmente uma entrada na base de dados de honeypot. O intuito disso é que seja disparado um alarme ao entrar com essa senha durante um logon interativo. Relembrando, o arquivo com a base de dados está localizado em `/opt/dcept/var/honeypot.db`. O comando usado foi `INSERT INTO logs VALUES ( '2016-12-11', 'LAB.LOCAL', 'Administrador', 'VM-WINDOWS-81', 'teste'`).



```

fabio — root@ubuntu: /opt/dcept/var — ssh dcept — 100x12
~ — root@ubuntu: /opt/dcept/var — ssh dcept  ~/Dropbox/Mestrado Profissional/Projeto DCEPT — -bash +
db_version logs
[sqlite> select * from logs
[ ...> ;
2016-04-20 11:15:35.987275|ALLSAFE.LAN|Administrator|FAKE-PC|dcepttest
2016-04-20 11:15:52.905345|ALLSAFE.LAN|Administrator|VM-WINDOWS-81|9dTFs9ZbxW
2016-05-25 02:51:50.235970|ALLSAFE.LAN|Administrator|VM-WINDOWS-81|pM3H7QGljh
2016-05-25 03:04:35.857174|IC.LOCAL|Administrator|VM-WINDOWS-81|2bJHfFwUiv
2016-07-05 15:09:17.579675|IC.LOCAL|Administrator|VM-WINDOWS-81|N8UJQN0JhJ
2016-07-06 15:09:17.306920|IC.LOCAL|Administrator|VM-WINDOWS-81|oRZPgQXXmh
2016-07-07 00:55:01.296142|IC.LOCAL|Administrator|VM-WINDOWS-81|pRzL6ivf6p
[sqlite> insert into logs values ('2016-12-11','LAB.LOCAL','Administrador','VM-WINDOWS-81','teste'); ]
sqlite>

```

Figura 4.13: Inserção manual de honeytoken na base de dados.

Feito isso, é hora de testar uma última vez a ferramenta DCEPT, como forma de garantir que as alterações surtiram efeito.

```

fabio — root@ubuntu: /opt/dcept — ssh dcept — 132x43
~ — root@ubuntu: /opt/dcept — ssh dcept  ~/Dropbox/Mestrado Profissional/Projeto DCEPT — -bash +
root@ubuntu:/opt/dcept# python -B dcept.py

  D E C E P T

Server configured as master node
Starting DCEPT...
Database contains 8 generated passwords
Starting honeytoken generation server HTTP daemon 0.0.0.0:80
kerbsniff: Looking for LAB.LOCAL\Administrador on eth0
Using selector: EpollSelector
kerb-as-req for domain user LAB\Administrador
PA-ENC-TIMESTAMP: None
kerb-as-req for domain user LAB\Administrador
PA-ENC-TIMESTAMP: None
Ignoring kerberos packet - Not kerb-as-req
Ignoring kerberos packet - Not kerb-as-req
kerb-as-req for domain user LAB\Administrador
PA-ENC-TIMESTAMP: 395036eb4d9e4de39a63503b560c5b74f8ff3f7241d40724cf524f9b4ae418543cc2aa07369de41ecd205e9dfb2a5affa8c5c02aee5ad38f
Cracker enqueued 1 encrypted timestamp. Queue size: 1
Recovering password from encrypted timestamp...
Testing 8 password(s)
kerb-as-req for domain user LAB\Administrador
PA-ENC-TIMESTAMP: 395036eb4d9e4de39a63503b560c5b74f8ff3f7241d40724cf524f9b4ae418543cc2aa07369de41ecd205e9dfb2a5affa8c5c02aee5ad38f
Cracker enqueued 1 encrypted timestamp. Queue size: 1
Ignoring kerberos packet - Not kerb-as-req
Ignoring kerberos packet - Not kerb-as-req
Warning: OpenMP is disabled; a non-OpenMP build may be faster
Press 'q' or Ctrl-C to abort, almost any other key for status
1g 0:00:00:00 DONE (2016-12-10 21:00) 100.0g/s 800.0p/s 800.0c/s 800.0C/s teste..dcepttest
Use the "--show" option to display all of the cracked passwords reliably
Session completed
Cracking job completed
Loaded 1 password hash (krb5pa-sha1, Kerberos 5 AS-REQ Pre-Auth etype 17/18 [8x SSE2])
teste      (?)

Cracked! Password: teste
[RED ALERT]
[RED ALERT] Honeytoken for LAB.LOCAL\Administrador 'teste' was stolen from VM-WINDOWS-81 on 2016-12-11

```

Figura 4.14: Detecção de honeytoken após alteração no código.

Como esperado a ferramenta DCEPT acusou a detecção do *honeytoken*.

#### 4.5. IMPLEMENTAÇÕES DE INTERESSE COM FERRAMENTAS OPEN SOURCE

Foi relaizada a tarefa de procurar por implementações de ferramentas que trabalham com o Protocolo Kerberos. Mais precisamente, não só que trabalhem com o protocolo, mas que sejam capazes de reproduzir o papel de um controlador de domínio. Isso possui um fim de ter uma base inicial para a implementação de uma ferramenta mais completa.

Nisso foi encontrado o projeto Kerby (2015) da Fundação de Software Apache (*Apache Software Foundation*). Ele é uma implementação em linguagem de programação Java de um KDC e do Protocolo Kerberos. Uma inspeção no código e no manual mostra que ele possui o que é desejado no tocante a gerar mensagens Kerberos.

Além disso existe a suíte de software Samba (SOFTWARE FREEDOM CONSERVANCY, 2016b), há longo tempo no mercado (desde 1992), que trabalha com várias versões do sistema operacional Windows. Basta um comando `git clone git://git.samba.org/samba.git samba` para se ter acesso ao repositório principal com o código da ferramenta.

#### 4.6. DISCUSSÃO

Logo, como visto, há todo um conjunto de ferramentas e tecnologias disponíveis para estudo e uso no advento da implementação de uma ferramenta que use a arquitetura e requisitos ora propostos.

Com o uso da ferramenta Mimikatz vê-se que é possível que seja feita a injeção de *honeyhashes* e *honeytickets* em memória. A injeção pode ser feita usando diversos métodos e não apenas usando APIs do Windows, como utilizado pela ferramenta DCEPT. Ressalta-se que a ferramenta Mimikatz possui código aberto e disponível no sítio Github, portanto, passível de estudo e análise para uma possível implementação.

O experimento realizado na Seção 4.4.2 encaminha o tráfego Kerberos pelo servidor Ubuntu, mostrando que é possível usar um servidor como um *man-in-the-middle*, uma forma de proxy reverso transparente para o protocolo. Isto é alcançado por meio de uma correta configuração de um serviço DNS. Apesar de ter sido usada uma configuração manual do servidor DNS na estação de trabalho cliente, o mesmo poderia ter sido alcançado utilizando um servidor DHCP. Conforme já dito, esse proxy reverso seria capaz de inspecionar as trocas de mensagens Kerberos e fazer o encaminhamento conforme uma lógica estabelecida. Essa lógica é a proposta na arquitetura. Se a porção codificada na requisição for decifrada, sabe-se que se trata de um *honeyhash* ou *honeyticket*. Nessa caso uma mensagem deve ser passada para um módulo capaz de tratar o evento que tem o papel de integração com outras ferramentas de monitoramento. Caso contrário, isto é, se a mensagem não for decifrada é porque não se trata de um *honeytoken* e a requisição deve ser encaminhada para o verdadeiro KDC, de forma que ele a trate.

O experimento realizado na Seção 4.4.3, que trata do carregamento do tipo de criptografia (etype) correto, dá a base necessária para compreender como tratar esse ponto. É preciso se preocupar com as configurações de tipo de criptografia tanto para o computador quanto para o

cliente que está se autenticando. Além disso a observação sugere que a configuração do salt pode exigir um pouco mais para o correto funcionamento de uma solução que trate adequadamente os diversos algoritmos de criptografia.

Além disso as implementações open source podem prover entendimento importante sobre uma implementação. As ferramentas apresentadas na Seção 4.5, Kerby e Samba, são feitas em linguagens de programação muito populares, Java e C/C++, respectivamente.

#### 4.7. PROPOSTAS COMPLEMENTARES

A partir da revisão da literatura, algumas propostas complementares podem auxiliar na compreensão e estudo nessa área de conhecimentos relativa a ataques PtH&T. Por isso foram elaboradas duas proposições adicionais presentes nas seções seguintes: uma *kill chain* para ataques PtH&T e uma classificação para as diferentes abordagens de combate a ataques.

##### 4.7.1. Proposta de Kill Chain para Ataques PtH&T

Nota-se a partir do estudado até aqui que tanto a execução de ataques PtH&T quanto as abordagens de tratamento acontecem em pontos específicos de uma cadeia de eventos. De maneira análoga à Intrusion Kill Chain, é possível pensar em ataques PtH&T como uma cadeia de fases executadas para um ataque bem sucedido. Desta forma é apresentada uma nova cadeia, doravante denominada *Pass-the-Hash & Tickets Kill Chain – PtH&T-KC* – que é uma especialização de algo mais amplo, similarmente como ocorre nas linguagens de programação orientadas a objeto.

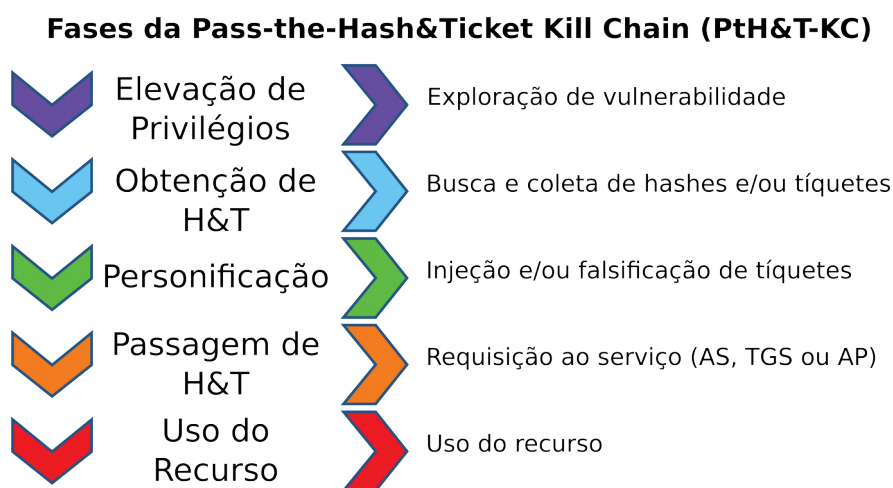


Figura 4.15: Fases da *Pass-the-Hash&Ticket Kill Chain*

A análise da PtH&T-KC permite uma percepção interessante sobre as abordagens usadas para parar esses ataques. De fato, alguns consultores pregam que o mais importante é não deixar que um atacante obtenha privilégios de administrador no seu computador. E essa é uma medida buscada em

diversos campos da segurança da informação. Essa medida se encaixa no primeiro estágio da cadeia. Algumas medidas de mitigação, como a restrição do uso de contas de administrador local já se encaixem aqui, visando uma possível *inside threat* (ameaça interna). Esse tipo de medida não se restringe só a uma parte da cadeia, podendo servir como medida de controle para outros pontos como a passagem de *hashes* e tíquetes.

Algumas inovações trazidas pelas diversas versões do windows – *Protected Users Group*, *LSA Protection* e *Credential Guard* – tentam dificultar a obtenção de *hashes* e tíquetes por parte de invasores. Nesse ponto tem-se a fase 2 da PtH&T-KC, referente a obtenção de *hashes* e tíquetes. Aqui já devem ser lançados alguns registros de eventos (*event log*) que permitiria a análise de um possível ataque desse tipo.

Analisando o estágio 3, personificação, nota-se que há uma certa escassez de medidas de controle nessa fase. Sabe-se da *LSA Protection* que tenta evitar que processos maliciosos tenham acesso ao processo LSASS, mas, como visto, algumas ferramentas como o Mimikatz são capazes de contornar essa medida protetiva.

Na 4ª etapa, as requisições feitas geram entradas de *logs* que possibilitam a análise. Aqui entram também outras medidas que podem ser usadas por servidores de aplicação como a validação do PAC junto ao KDC, no caso de um *Silver Ticket*. Te-se também medidas como a restrição da validade de tíquetes e troca de senhas periodicamente.

Finalizando, a última etapa onde o atacante pode ser impedido, há a possibilidade do uso de técnicas de detecção de anomalia, subentendendo que é possível um uso incomum do recurso por parte do atacante. Aqui é possível que haja uma dificuldade maior para o impedimento do ataque, dependendo dos movimentos do intruso e da capacidade das implementações dos algoritmos de classificação.

#### **4.7.2. Proposta de Classificação para as Abordagens de Tratamento aos Ataques**

Uma vez que a literatura não o faça, é proposta uma forma de classificação das medidas tomadas para combater o fenômeno de ataques PtH&T, como uma espécie de “taxonomia”. Isso tem o propósito de organizar essas ações e auxiliar os profissionais da área acerca do entendimento quanto a que parte do assunto relativo a ataques PtH&T se está abordando no decorrer de debates. Tomando como base a revisão da literatura percebe-se que existem 3 grandes grupos de medidas quando se fala no assunto:

1. Inovação no SO;

2. Mitigação; e
3. Detecção.

A Figura 4.16 apresenta um resumo da proposta. No grupo de Inovações no Sistema Operacional estão as medidas estudadas na Seção 2.5.2 como *Protected Users Group*, *LSA Protection* e *Credential Guard*. Observando o conjunto dessas medidas se é capaz de perceber que esses mecanismos implementados pela Microsoft se concentram na fase 2 da PtH&T-KC, para dificultar os esforços de um atacante na obtenção de *hashes* e tíquetes.

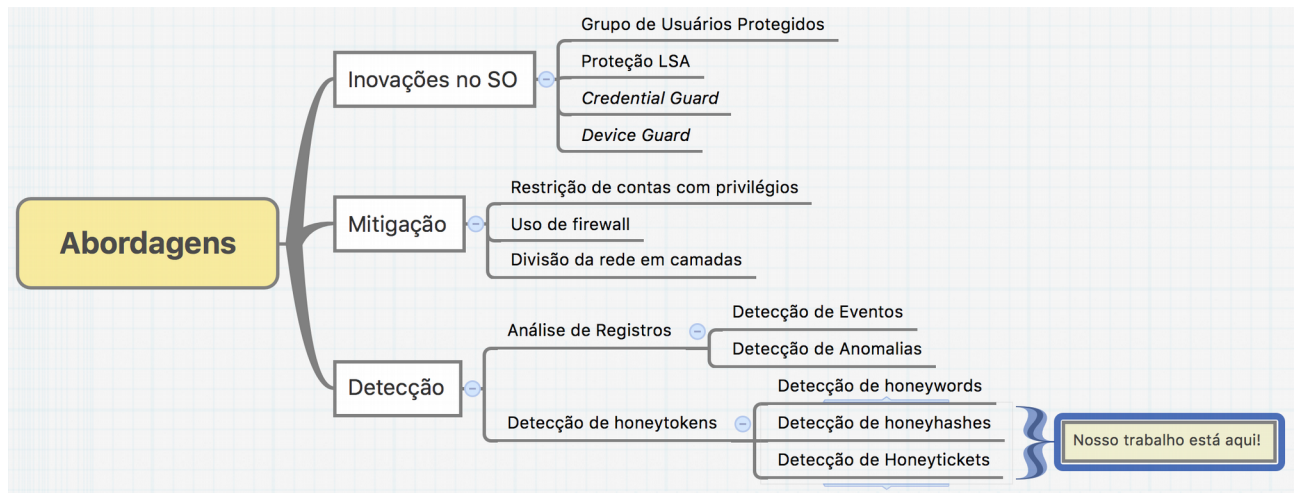


Figura 4.16: Proposta de classificação para as abordagens de combate a ataques PtH&T.

No grupo da Mitigação tem-se as diversas medidas que um administrador de rede pode executar para minimizar a potencialidade dos ataques. Esses tipos de ações atuam de maneira geral na maioria das fases da PtH&T-KC.

No grupo da Detecção vislumbra-se 2 subcategorias:

1. Análise de Registros de Eventos;
2. Detecção de *Honeytokens*.

A subcategoria Análise de Registros de Eventos engloba duas formas de detecção. A primeira forma de detecção fora estudada na Seção 2.5.3, a qual deu-se o nome de Detecção de Eventos. A segunda forma de detecção fora estudada na Seção 2.5.4, a qual deu-se o nome de Detecção de Anomalia. Foi escolhido agrupar essas duas formas de detecção pelo fato de geralmente trabalharem com Registros de Eventos do Windows (WEL). Aqui tem-se uma influência da quantidade *logs* a serem analisados, que a razão leva a crer ser, via de regra, proporcional ao tamanho da rede a que se deseja proteger.

A subcategoria Detecção de Honeytokens como já visto traz uma abordagem diferente da anterior. Apesar do estudo ter se limitado ao uso de *honeywords* na fase 4 da PtH&T-KC, passagem de *hashes* e tíquetes, o emprego de outros tipos de honeytokens pode ser interessante para atuarem na fase 5 da PtH&T-KC (uso do recurso), como arquivos de engodo em compartilhamentos.

## 5. CONCLUSÕES E TRABALHOS FUTUROS

Retomando os objetivos apresentados no início do estudo, conclui-se que a longa existência de ataques PtH&T em Redes de Domínio Windows tem causa na própria implementação de autenticação. Como forma de proporcionar conforto ao usuário para que ele entre com a senha apenas uma vez durante o processo de autenticação, o Windows armazena chaves e tíquetes em memória como forma de *cache*. Outro ponto advém da manutenção de compatibilidade, usando como chave para o tipo de criptografia RC4 o *hash* NTLM, armazenado também no disco rígido. Percebe-se que não é tarefa fácil evitar a obtenção desses *hashes* por parte de atacantes, tendo o Windows implantado melhoramentos para tal ao longo dos anos. São vários os mecanismos de combate a esses ataques e foi proposta uma classificação para melhorar a organização e compreensão dos mesmos. A análise da PtH&T-KC deixa com a impressão de que pode haver campo para pesquisa na fase de personificação. A principal dificuldade na detecção desses ataques é que as mensagens trocadas e tíquetes forjados são muito similares aos legítimos equivalentes.

A detecção de ataques PtH&T dentro do ambiente de Redes de Domínio do Windows é importante para evitar que invasores prossigam com seus objetivos, já que normalmente, ataques dessa natureza indicam a preparação para ataques com maior potencial de danos. O uso de *honeytokens*, embora não seja considerado como uma técnica de detecção nova, é uma maneira viável de resolver parte do problema de detecção de uso de credenciais comprometidas. Além disso, essa técnica pode ser usada em conjunto com outras, como a detecção de anomalias, aumentando o leque de possibilidades de proteção contra ataques a esses tipos de redes. O estudo da ferramenta DCEPT ajudou a trazer novas perspectivas de melhorias ligadas às possibilidades de detecção. De acordo com as discussões sobre as melhorias na arquitetura proposta vislumbra-se uma alternativa para se ampliar o poder de detecção de ataques PtH&T. O uso de *honeyhashes* e *honeytickets* pode ser uma chave central para aumentar a detecção de ataques PtH&T no contexto de Redes Windows.

As principais contribuições desse trabalho são introduzir o conceito da utilização de *honeyhashes* (no lugar de *honeywords*) e *honeytickets* no ambiente de Rede Windows e propor uma arquitetura para a elaboração de ferramentas que usam *honeyhashes* e *honeytickets* e que possam ser integradas com outras ferramentas. Além disso, foi proposta a *Pass-the-Hash & Ticket Kill Chain* (PtH&T-KC), uma forma de Kill Chain que visa entender as fases dos ataques PtH&T, classificar e propor uma direção na implantação de mecanismos de controle que visam deter tais ataques. Também foi feita uma classificação das abordagens de tratamento de ataques PtH&T, as dividindo

em medidas de Inovações no SO, Mitigação e Detecção, a última com suas subcategorias, cada uma delas, em seus nós folhas, atuando em um estágio da PtH&T-KC.

## 5.1. PUBLICAÇÃO DE PESQUISA

Parte dessa pesquisa foi apresentado na Conferência Ibero Americana de Computação Aplicada – CIACA – 2016, realizada em Lisboa, Portugal, em dezembro de 2016 e será publicada nos anais da conferência.

## 5.2. RECOMENDAÇÕES PARA TRABALHOS FUTUROS

Como trabalho futuro espera-se implementar uma ferramenta completa de detecção a partir da arquitetura proposta, como prova de conceito, que seria um escudo para controladores de domínio e ativos de serviços mais valiosos em uma rede de Domínio Windows. Uma opção a ser avaliada é o uso dos softwares Samba, como um falso controlador de domínio, e OpenLDAP, como serviço de diretório de *honeypaccounts*, no backend para a solução. Também foi visto que existe o projeto Kerby da Apache que fornece *bindings* (ligações) para o Kerberos. Também se deseja implementar um módulo para algum tipo de serviço como um Compartilhamento do Windows ou um Servidor *web*.

Outra possibilidade é desenvolver um método para a comparação da detecção de anomalias com a detecção por honeytokens para poder comparar melhor as vantagens e desvantagens de uma em relação a outra, sugerindo algumas situações em que cada uma fosse mais indicada.

Outra linha é verificar uma forma de gerar chaves que tenham uma maior variação com o tempo. Algo parecido com o Autenticador do Google. Isso visa atuar em uma camada da PtH&T-KC que hoje é pouco visada, que é a personificação. Por exemplo, se um atacante obtiver um *hash* de uma conta *domainadmin* ele será útil até a troca de senha dessa conta. Se uma implementação dessas for possível, o usuário poderia usar o seu celular para criar uma chave de autenticação e os *hashes* gerados irão variar com o tempo, diminuindo a janela de oportunidade para uso de *hashes* obtidos por parte de um atacante. A grosso modo, se estaria trocando a senha do usuário diversas vezes.

Uma coisa que não ficou clara na literatura é se, no estágio de obtenção de *hashes/tickets* da PtH&T-KC, geralmente há exfiltração desses dados para uso em uma outra fonte ou computador, ou se geralmente esse tíquete é usado localmente no computador comprometido. Uma linha de investigação nesse sentido pode prover mais *insights* e possibilidades de detecção, por meio de *Host-based Intrusion Detection Systems*.



Durante a revisão da literatura percebeu-se que existem das mais variadas sugestões de medidas de mitigação conforme a classificação composta. Geralmente essas medidas aparecem nos artigos como um complemento ou ao lado de outros temas como a detecção e inovações. É possível fazer uma pesquisa mais ampla no intuito de se criar uma lista o mais completa possível, focando somente nas medidas de mitigação que um administrador de rede pode tomar. Tal lista pode servir como guia para aumentar a segurança em Redes de Domínio Windows.

Pensou-se também que seria interessante extrair *hashes* e tíquetes da memória, embaralhá-los com *honeypashes* e *honeytickets* similares e injetá-los de novo na memória. Desta forma, um atacante não seria capaz de distinguir entre eles. Apesar de essa ser uma das ideias mais interessantes, ela introduz um novo problema. Ao passo que engana-se um atacante com vários *hashes* e tíquetes misturados, esta se enganando também o sistema operacional. Nesse ponto, não foi possível pensar numa forma para que o SO pudesse usar o tíquete correto. Ademais, em um sistema operacional comprometido, que foi a premissa, se o SO é capaz de saber a diferença é provável que o atacante também tenha mecanismos para descobri-lo. Talvez haja terreno a ser explorado nesse sentido.

Um conceito bem mais simples é o uso de *honeypaccounts* para detectar atacantes mais desprezados. Se trata simplesmente de contas falsas. Quando um *hash* ou um tíquete for detectado com essa conta falsa, um alarme deve ser disparado. Pela simplicidade do método, também seria fácil para um atacante mais atento perceber que se trata de um engodo. Entretanto, invasores desatentos ou *script kiddies* (garotos de *script*) poderiam cair por esse mecanismo. A segurança pode ser composta por vários aspectos e esse seria um deles.

## REFERÊNCIAS BIBLIOGRÁFICAS

APACHE SOFTWARE FOUNDATION. *A Kerberos Protocol and KDC Implementation*. Disponível em: <<http://directory.apache.org/kerby/>>. Acesso em: 14 dez. 2016.

BASHAR, Ewaida. *Pass-the-hash attacks: Tools and Mitigation*. Disponível em: <<https://www.sans.org/reading-room/whitepapers/testing/pass-the-hash-attacks-tools-mitigation-33283>>. Acesso em: 28 jun. 2016.

BRIAN LICH. *Protect derived domain credentials with Credential Guard (Windows 10)*. Windows IT Center. [S.l.: s.n.]. Disponível em: <<https://technet.microsoft.com/en-us/itpro/windows/keep-secure/credential-guard>>. Acesso em: 28 out. 2016. , 21 out. 2016

CERT-EU, Computer Emergency Response Team. *Protection from Kerberos Golden Ticket*. [S.l.: s.n.]. Disponível em: <[https://cert.europa.eu/static/WhitePapers/CERT-EU-SWP\\_14\\_07\\_PassTheGolden\\_Ticket\\_v1\\_1.pdf](https://cert.europa.eu/static/WhitePapers/CERT-EU-SWP_14_07_PassTheGolden_Ticket_v1_1.pdf)>. Acesso em: 10 out. 2016. , 10 jun. 2014

CHANDOLA, Varun; BANERJEE, Arindam; KUMAR, Vipin. Anomaly detection: A survey. *ACM Computing Surveys* v. 41, n. 3, p. 1–58 , 1 jul. 2009.

COMMITTEE ON COMMERCE, SCIENCE AND TRANSPORTATION. *A “Kill Chain” Analysis of the 2013 Target Data Breach*. [S.l.: s.n.]. Disponível em: <<http://rnc2.com/blog/wp-content/uploads/2014/11/Target%20Kill%20Chain%20Analysis.pdf>>. Acesso em: 6 dez. 2016. , mar. 2014

CULP, Scott. *Ten Immutable Laws Of Security (Version 2.0)*. Disponível em: <<https://technet.microsoft.com/library/cc722487.aspx>>. Acesso em: 6 jan. 2017.

DE BARROS, Augusto Paes. *IDS: RES: Protocol Anomaly Detection IDS - Honeypots*. Disponível em: <<http://seclists.org/focus-ids/2003/Feb/95>>. Acesso em: 28 jun. 2016.

DUCKWALL, Alva; DELPY, Benjamin. Abusing Microsoft Kerberos. In: BLACK HAT USA 2014, ago. 2014, Las Vegas, NV. Anais... Las Vegas, NV: [s.n.], ago. 2014. Disponível em: <<https://www.blackhat.com/us-14/archives.html#abusing-microsoft-kerberos-sorry-you-guys-dont-get-it>>. Acesso em: 27 out. 2016.

ERIN KELLY. *OPM hack Q&A: What we know and what we don't*. Disponível em: <<http://www.usatoday.com/story/news/politics/2015/06/27/opm-hack-questions-and-answers/29333211/>>. Acesso em: 15 nov. 2016.

EXORCYST. *Still Passing the Hash 15 Years Later: Mimikatz and Golden Tickets... What's the BFD?*. Still Passing the Hash 15 Years Later. [S.l.: s.n.]. Disponível em: <<http://passing-the-hash.blogspot.com.br/2014/08/mimikatz-and-golden-tickets-whats-bfd.html>>. Acesso em: 11 out. 2016. , 21 ago. 2014

FOSTER, James. *Are there novel ways to mitigate credential theft attacks in Windows?*. [S.l.]: The SANS Institute. Disponível em: <<https://www.sans.org/reading-room/whitepapers/bestprac/ways-mitigate-credential-theft-attacks-windows-35337>>. Acesso em: 3 nov. 2016. , jul. 2014

HSIEH, Chih-Hung *et al.* *AD2: Anomaly detection on active directory log data for insider threat monitoring*. [S.l.]: Security Technology (ICCST), 2015 International Carnahan Conference.

Disponível em: <<http://ieeexplore.ieee.org/document/7389698/>>. Acesso em: 29 set. 2016. , set. 2015

HUMMEL, Chris. *Why Crack When You Can Pass the Hash?* Disponível em: <<https://www.sans.org/reading-room/whitepapers/testing/crack-pass-hash-33219>>. Acesso em: 28 jun. 2016.

HUTCHINS, Eric M.; CLOPPERT, Michael J.; AMIN, Rohan M. Intelligence-driven computer network defense informed by analysis of adversary campaigns and intrusion kill chains. *Leading Issues in Information Warfare & Security Research* v. 1, p. 80 , 2011.

ITRC. *ITRC Breach Stats Report Summary 2016* . [S.l.]: Identity Theft Resource Center. Disponível em: <<http://www.idtheftcenter.org/images/breach/ITRCBreachStatsReportSummary2016.pdf>>. Acesso em: 15 nov. 2016. , 8 nov. 2016

JUELS, A.; RISTENPART, T. Honey Encryption: Encryption beyond the Brute-Force Barrier. *IEEE Security Privacy* v. 12, n. 4, p. 59–62 , jul. 2014.

JUELS, Ari. A Bodyguard of Lies: The Use of Honey Objects in Information Security. SACMAT '14, 2014, New York, NY, USA. *Anais...* New York, NY, USA: ACM, 2014. p.1–4. Disponível em: <<http://doi.acm.org/10.1145/2613087.2613088>>. Acesso em: 6 out. 2016. 978-1-4503-2939-2. .

JUNGLES, Patrick *et al.* *Mitigating Pass-the-Hash Other Credential theft, version 2* . [S.l.: s.n.]. Disponível em: <<https://download.microsoft.com/download/7/7/A/77ABC5BD-8320-41AF-863C-6ECFB10CB4B9/Mitigating-Pass-the-Hash-Attacks-and-Other-Credential-Theft-Version-2.pdf>>. Acesso em: 28 set. 2016. , 2014

JUNGLES, Patrick *et al.* *Mitigating Pass-the-Hash (PtH) Attacks and Other Credential Theft Techniques* . [S.l.: s.n.]. Disponível em: <[https://download.microsoft.com/download/7/7/A/77ABC5BD-8320-41AF-863C-6ECFB10CB4B9/Mitigating%20Pass-the-Hash%20\(PtH\)%20Attacks%20and%20Other%20Credential%20Theft%20Techniques\\_English.pdf](https://download.microsoft.com/download/7/7/A/77ABC5BD-8320-41AF-863C-6ECFB10CB4B9/Mitigating%20Pass-the-Hash%20(PtH)%20Attacks%20and%20Other%20Credential%20Theft%20Techniques_English.pdf)>. Acesso em: 28 set. 2016. , 2012

KHEDKAR, S.P. *et al.* Achieving Flatness by Selecting the Honey words from Existing User Passwords. *International Journal of Engineering Science and Computing* v. 6, n. 5, p. 6041–6045 , maio 2016.

LÓPEZ, Miguel Hernández; RESÉNDEZ, Carlos Francisco Lerma. Honeypots: Basic Concepts, Classification and Educational Use as Resources in Information Security Education and Courses. In: INSITE 2008: INFORMING SCIENCE + IT EDUCATION CONFERENCE, 23 jun. 2008, Varna, Bulgaria. *Anais...* Varna, Bulgaria: [s.n.], 23 jun. 2008. p.8. Disponível em: <<http://proceedings.informingscience.org/InSITE2008/InSITE08p069-076Hernan422.pdf>>.

MADDIE EGAN. *End the game for credential theft with Windows 10* .Microsoft Tech Community. [S.l.: s.n.]. Disponível em: <<https://techcommunity.microsoft.com/t5/Microsoft-Ignite-Content/BRK2132-End-the-game-for-credential-theft-with-Windows-10/m-p/10549#M549>>. Acesso em: 15 nov. 2016. , 8 set. 2016

MARTIN, Sam; TOKUTOMI, Mark. Password Cracking. , 2012. Disponível em: <<https://www.cs.arizona.edu/~collberg/Teaching/466-566/2014/Resources/presentations/2012/reports.pdf>>. Acesso em: 5 jul. 2016.

METCALF, Sean. *Attackers Can Now Use Mimikatz to Implant Skeleton Key on Domain Controllers & BackDoor Your Active Directory Forest*. Active Directory Security. [S.l: s.n.]. Disponível em: <<https://adsecurity.org/?p=1275>>. Acesso em: 7 dez. 2016a. , 19 jan. 2015

METCALF, Sean. *How Attackers Dump Active Directory Database Credentials* . [S.l: s.n.]. Disponível em: <<https://adsecurity.org/?p=2398>>. Acesso em: 11 out. 2016. , 3 jan. 2016

METCALF, Sean. *How Attackers Extract Credentials (Hashes) From LSASS* . [S.l: s.n.]. Disponível em: <<http://adsecurity.org/?p=462>>. Acesso em: 11 out. 2016a. , 6 nov. 2014

METCALF, Sean. *How Attackers Use Kerberos Silver Tickets to Exploit Systems* . [S.l: s.n.]. Disponível em: <<https://adsecurity.org/?p=2011>>. Acesso em: 25 out. 2016b. , 17 nov. 2015

METCALF, Sean. *Kerberos & KRBTGT: Active Directory's Domain Kerberos Service Account* . [S.l: s.n.]. Disponível em: <<https://adsecurity.org/?p=483>>. Acesso em: 11 out. 2016b. , 10 nov. 2014

MICROSOFT CORPORATION. *Active Directory*. Disponível em: <<https://technet.microsoft.com/library/Bb742424>>. Acesso em: 8 out. 2016.

MICROSOFT CORPORATION. *Configuring Kerberos Policies*. Disponível em: <<https://technet.microsoft.com/en-us/library/dd277401.aspx>>. Acesso em: 3 nov. 2016.

MICROSOFT CORPORATION. *How the Kerberos Version 5 Authentication Protocol Works: Logon and Authentication*. Disponível em: <[https://technet.microsoft.com/pt-br/library/cc772815\(v=ws.10\).aspx](https://technet.microsoft.com/pt-br/library/cc772815(v=ws.10).aspx)>. Acesso em: 27 out. 2016.

MICROSOFT CORPORATION. *[MS-KILE]: Kerberos Protocol Extensions* . [S.l: s.n.]. Disponível em: <<https://msdn.microsoft.com/en-us/library/cc233856.aspx>>. Acesso em: 9 out. 2016a. , 14 jul. 2016

MICROSOFT CORPORATION. *Preauthentication*. Disponível em: <<https://technet.microsoft.com/en-us/library/cc961961.aspx>>. Acesso em: 9 out. 2016b.

MICROSOFT CORPORATION. *Protected Users Security Group*. Disponível em: <[https://technet.microsoft.com/en-us/library/dn466518\(v=ws.11\).aspx](https://technet.microsoft.com/en-us/library/dn466518(v=ws.11).aspx)>. Acesso em: 7 dez. 2016.

MICROSOFT CORPORATION. *SRV Resource Records*. Disponível em: <<https://technet.microsoft.com/en-us/library/cc961719.aspx>>. Acesso em: 10 dez. 2016c.

NEEDHAM, Roger M.; SCHROEDER, Michael D. Using Encryption for Authentication in Large Networks of Computers. *Commun. ACM* v. 21, n. 12, p. 993–999 , dez. 1978.

NEUMAN, Clifford *et al.* *The Kerberos Network Authentication Service (V5)*. Disponível em: <<https://tools.ietf.org/html/rfc4120.html>>. Acesso em: 3 ago. 2016.

ORACLE CORPORATION. *User Manual*. Disponível em: <<https://www.virtualbox.org/manual/ch06.html#networkingmodes>>. Acesso em: 7 dez. 2016.

PILKINGTON, Mike. *Kerberos in the Crosshairs: Golden Tickets, Silver Tickets, MITM, and More* .SANS Digital Forensics and Incident Response Blog. [S.l: s.n.]. Disponível em:

<<https://digital-forensics.sans.org/blog/2014/11/24/kerberos-in-the-crosshairs-golden-tickets-silver-tickets-mitm-more>>. Acesso em: 3 nov. 2016. , 24 nov. 2014

RAEBURN, Kenneth. *Advanced Encryption Standard (AES) Encryption for Kerberos 5*. Disponível em: <<https://tools.ietf.org/html/rfc3962>>. Acesso em: 10 ago. 2016a.

RAEBURN, Kenneth. *Encryption and Checksum Specifications for Kerberos 5*. Disponível em: <<https://tools.ietf.org/html/rfc3961>>. Acesso em: 18 jul. 2016b.

SAPEGIN, Andrey *et al.* Poisson-Based Anomaly Detection for Identifying Malicious Behaviour. In: MOBILE, SECURE, AND PROGRAMMABLE NETWORKING – FIRST INTERNATIONAL CONFERENCE, MSPN 2015, jun. 2015, Paris, France. Anais... Paris, France: Springer, jun. 2015. p.134–150. Disponível em: <[http://link-springer-com/chapter/10.1007/978-3-319-25744-0\\_12](http://link-springer-com/chapter/10.1007/978-3-319-25744-0_12)>. Acesso em: 29 set. 2016.

SECUREWORKS, Dell. *DCEPT*. Disponível em: <<https://github.com/secureworks/dcept>>. Acesso em: 14 dez. 2016.

SECURITYFOCUS. *NT “Pass the Hash” with Modified SMB Client Vulnerability*. Disponível em: <<http://www.securityfocus.com/bid/233/discuss>>. Acesso em: 27 out. 2016.

SOFTWARE FREEDOM CONSERVANCY. *Git*. Disponível em: <<https://git-scm.com/>>. Acesso em: 11 dez. 2016a.

SOFTWARE FREEDOM CONSERVANCY. *Main Page*. Disponível em: <[https://wiki.samba.org/index.php/Main\\_Page](https://wiki.samba.org/index.php/Main_Page)>. Acesso em: 14 dez. 2016b.

SPITZNER, Lance. *Honeypots: tracking hackers*. Boston: Addison-Wesley, 2002. 452 p. .978-0-321-10895-1.

STATCOUNTER GLOBAL STATS. *Top 7 Desktop OSs from Jan 2015 to Nov 2016*. Disponível em: <<http://gs.statcounter.com/#desktop-os-ww-monthly-201501-201611>>. Acesso em: 6 dez. 2016.

STEWART, Joe; BETKE, James. *DCEPT – Open Source Honeytoken Tripwire*. Disponível em: <<https://www.secureworks.com/blog/dcept>>. Acesso em: 5 jul. 2016.

SUN, Hongwei. *Windows Configurations for Kerberos Supported Encryption Type*. Disponível em: <<https://blogs.msdn.microsoft.com/openspecification/2011/05/30/windows-configurations-for-kerberos-supported-encryption-type/>>. Acesso em: 10 ago. 2016.

THE HONEYNET PROJECT. *Honeywall CDROM*. Disponível em: <<https://www.honeynet.org/project/HoneywallCDROM>>. Acesso em: 27 out. 2016.

THE HONEYNET PROJECT. *Know Your Enemy: Honeynets*. Disponível em: <<http://old.honeynet.org/papers/honeynet/>>. Acesso em: 27 out. 2016.

THOMLINSON, Matt. *New Guidance to Mitigate Determined Adversaries’ Favorite Attack: Pass-the-Hash*. Microsoft Secure Blog. [S.l.: s.n.]. Disponível em: <<http://blogs.microsoft.com/microsoftsecure/2012/12/11/new-guidance-to-mitigate-determined-adversaries-favorite-attack-pass-the-hash/>>. Acesso em: 18 out. 2016. , 11 dez. 2012

VERIZON ENTERPRISE. *2016 Data Breach Investigations Report* . [S.l: s.n.]. Disponível em: <[http://www.verizonenterprise.com/resources/reports/rp\\_DBIR\\_2016\\_Report\\_en\\_xg.pdf](http://www.verizonenterprise.com/resources/reports/rp_DBIR_2016_Report_en_xg.pdf)>. Acesso em: 13 set. 2016. , 2016

YU, Yingbing. Anomaly Intrusion Detection Based upon Anomalous Events and Soft Computing Technique. *International Journal of Machine Learning and Computing* v. 5, n. 6, p. 450–453 , dez. 2015.

ZHU, Larry; JAGANATHAN, Karthik; BREZAK, John. *The RC4-HMAC Kerberos Encryption Types Used by Microsoft Windows*. Disponível em: <<https://tools.ietf.org/html/rfc4757>>. Acesso em: 18 jul. 2016.

ZYLVA, Ash. *Windows 10 Device Guard and Credential Guard Demystified*. Disponível em: <<https://blogs.technet.microsoft.com/ash/2016/03/02/windows-10-device-guard-and-credential-guard-demystified/>>. Acesso em: 3 nov. 2016.

## **ANEXOS**

## A – PLAYBOOK ANSIBLE

```
---
- hosts: all

vars:
  - john_output: /tmp/john.tar.gz
  - john_dir: /tmp/john
  - john_url: http://www.openwall.com/john/j/john-1.8.0-jumbo-1.tar.gz
  - john_download_checksum: sha1:31c8246d3a12ab7fd7de0d1070dda4654f5397a7
  - john_bin_sha1: a2ad43cf059a1b8c58e4671b40513277d3405d55
  - john_bin_md5: 7dc9da949bde496932658880bd38d865
  - dcept_folder: /usr/share/dcept
  - dcept_home: /opt/dcept
  - dcept_config_dir: /etc/dcept
  - dcept_config_file: dcept.cfg
  - dcept_db_dir: /var/lib/dcept
  - dcept_db_file: honeypot.db
  - dcept_log_dir: /var/log/dcept
  - dcept_log_file: dcept.log
  - dcept_cfg_domain: IC.LOCAL
  - dcept_cfg_realm: IC

tasks:

  - name: coloca o bash como shell padrão
    file: src=/bin/bash dest=/bin/sh state=link

  - name: atualiza pacotes
    apt: upgrade=yes update_cache=yes

  - name: instala diversos pacotes
    apt: name={{ item }}
    with_items:
      - cron
      - wget
      - build-essential
      - libssl-dev
      - python-pip
      - python-setuptools
      - tcpdump
      - tshark
      - python-dev
      - libxml2-dev
      - libxslt1-dev

  - name: instala pacotes python via pip
    pip: name={{ item }}
    with_items:
```



```

- pyshark
- pyiface

- name: diretório para a base de dados
  file: path={{ dcept_home }}/var state=directory

- name: diretório para extração do JTR
  file: path={{ john_dir }} state=directory

- name: verifica a instalação do John the Ripper
  stat:
    path: "{{ dcept_home }}/john"
    get_checksum: True
    #checksum_algorithm: sha1
  register: john_checksum

- name: debug john_checksum
  debug: var=john_checksum

- name: download John the Ripper (JTR)
  get_url:
    url: "{{ john_url }}"
    dest: "{{ john_output }}"
    #checksum: "{{ john_checksum }}"
  when: john_checksum.stat.exists == false

- name: descompacta o JTR
  command: tar -xvf {{ john_output }} -C {{ john_dir }} --strip-components=1
  when: john_checksum.stat.exists == false

- name: corrige bug com gcc v5 compilando JTR
  command: "sed -i 's/#define MAYBE_INLINE_BODY MAYBE_INLINE/#define
MAYBE_INLINE_BODY/g' /tmp/john/src/MD5_std.c"
  when: john_checksum.stat.exists == false

- name: verifica o executável compilado JTR
  stat: path={{ john_dir }}/run/john
  register: john_bin

- name: compila JTR
  shell: "cd /tmp/john/src && ./configure && make clean && make -s"
  when: john_bin.stat.exists == false

- name: copia executável JTR para o home do dcept
  command: cp {{ john_dir }}/run/john {{ dcept_home }}/john
  when: john_checksum.stat.exists == false

- name: arquivo ini para JTR
  file: path={{ dcept_home }}/john.ini state=touch

```

```

- name: checkout do repositório dcept
git:
repo: https://github.com/secureworks/dcept.git
dest: /usr/share/dcept
update: no

- name: link de vários arquivos do dcept
file: src={{ dcept_folder }}/server/{{ item }} dest={{ dcept_home }}/{{ item }} state=link
with_items:
- dcept.py
- Cracker.py
- GenerationServer.py
- ConfigReader.py
- alert.py

- name: diretório de configuração do dcept
file: path={{ dcept_config_dir }} state=directory

- name: arquivo de configuração dcept.cfg
template: src=templates/dcept.cfg.j2 dest={{ dcept_config_dir }}/{{ dcept_config_file }}

- name: link do arquivo de configuração com o home dcept
file:
src: "{{ dcept_config_dir }}/{{ dcept_config_file }}"
dest: "{{ dcept_home }}/{{ dcept_config_file }}"
state: link

# - name: diretório para a base de dados dcept
# file: path={{ dcept_db_dir }} state=directory

# - name: arquivo da base de dados dcept
# file: path={{ dcept_db_dir }}/{{ dcept_db_file }} state=touch

# - name: link do arquivo da base de dados para o home do dcept
# file:
# src: "{{ dcept_db_dir }}/{{ dcept_db_file }}"
# dest: "{{ dcept_home }}/var/{{ dcept_db_file }}"
# state: link

- name: diretório para os logs do dcept
file: path={{ dcept_log_dir }} state=directory

- name: arquivo de logs dcept
file: path={{ dcept_log_dir }}/{{ dcept_log_file }} state=touch

- name: link do arquivo de log para o home do dcept
file:
src: "{{ dcept_log_dir }}/{{ dcept_log_file }}"
dest: "{{ dcept_home }}/var/{{ dcept_log_file }}"
state: link

```

- name: cron para atualizações automáticas

cron:

name: atualizações automáticas

minute: 0

hour: 0

job: "apt-get update && apt-get upgrade -y"

user: root

## B – PATCH DE MUDANÇAS NO CÓDIGO DA FERRAMENTA DCEPT

```
diff --git a/server/Cracker.py b/server/Cracker.py
index 9e10e6f..c97ca3a 100644
--- a/server/Cracker.py
+++ b/server/Cracker.py
@@ -12,15 +12,18 @@ import shutil
import subprocess
import time

+from ConfigReader import config
+
class Cracker:

    def __init__(self):
        self.passwordQueue = Queue.Queue(maxsize=100)
        self.thread = threading.Thread(target = self._run, args = ())
        self.thread.daemon = True
+        config.load("/opt/dcept/dcept.cfg")

-    def enqueueJob(self, username, domain, encTimestamp, callback):
-        self.passwordQueue.put((username, domain, encTimestamp, callback))
+    def enqueueJob(self, username, domain, encTimestamp, callback, etype):
+        self.passwordQueue.put((username, domain, encTimestamp, callback, etype))
+        logging.debug("Cracker enqueued 1 encrypted timestamp. Queue size: %d" %
(self.passwordQueue.qsize()))

@@ -30,7 +33,7 @@ class Cracker:
    # It should crack the most recent passwords working backward. In practice the
    # only time this subroutine is called is when someone uses the honeypot
    # domain\username.
-    def recoverPassword(self, username, domain, encTimestamp, callback):
+    def recoverPassword(self, username, domain, encTimestamp, callback, etype):

        tmpDir = tempfile.mkdtemp("-dcept")
@@ -38,14 +41,17 @@ class Cracker:
        wordPath = tmpDir + "/wordlist.tmp"
        passPath = tmpDir + "/encPass.tmp"
        potPath = tmpDir + "/john.pot"
+        salt = ""
+        if etype != "23":
+            salt = config.domain + config.honey_username

        logging.debug("Recovering password from encrypted timestamp...")

        # Create password file for cracking tool
        fh = open(passPath, 'w')
```

```

-         fh.write("$%s$d%s$%s$%s" % ("krb5pa",18,username, domain, encTimestamp))
+         fh.write("$%s$%s$%s$%s$%s" % ("krb5pa", etype, username, domain, salt,
encTimestamp))
        fh.close()
-
+
        # Create word list of the generated passwords ordered by most recent
        fh = open(wordPath, 'w')

@@ -62,7 +68,7 @@ class Cracker:
        if logging.getLogger().getEffectiveLevel() != logging.DEBUG:
            redirectStr = "2>/dev/null"

-         result = subprocess.check_output("/opt/dcept/john --wordlist=%s --pot=%s
--format=krb5pa-sha1 %s %s" % (wordPath, potPath, passPath, redirectStr), shell=True)
+         result = subprocess.check_output("/opt/dcept/john --wordlist=%s --pot=%s %s %s" %
(wordPath, potPath, passPath, redirectStr), shell=True)

        print "Cracking job completed"
        shutil.rmtree(tmpDir)

@@ -86,7 +92,7 @@ class Cracker:
        while True:
            if not self.passwordQueue.empty():
                item = self.passwordQueue.get()
-                 self.recoverPassword(item[0], item[1], item[2], item[3])
+                 self.recoverPassword(item[0], item[1], item[2], item[3], item[4])

# Singleton
cracker = Cracker()
diff --git a/server/CrackerMod.py b/server/CrackerMod.py
new file mode 100644
index 0000000..4a51868
--- /dev/null
+++ b/server/CrackerMod.py
@@ -0,0 +1,108 @@
+#!/usr/bin/python
+
+# DCEPT
+# James Bettke
+# Dell SecureWorks 2016
+
+import logging
+import Queue
+import threading
+import tempfile
+import shutil
+import subprocess
+import time
+import os
+

```

```

+class Cracker:
+
+    def __init__(self):
+        self.passwordQueue = Queue.Queue(maxsize=100)
+        self.thread = threading.Thread(target = self._run, args = ())
+        self.thread.daemon = True
+
+    def enqueueJob(self, username, domain, encTimestamp, callback):
+        self.passwordQueue.put((username, domain, encTimestamp, callback))
+        logging.debug("Cracker enqueued 1 encrypted timestamp. Queue size: %d" %
(self.passwordQueue.qsize()))
+
+
+    # Take the encrypted timestamp and recover the generated password using a
+    # password cracker. This should not take take very long since we are only
+    # interested in the short word list of passwords made by the generation server.
+    # It should crack the most recent passwords working backward. In practice the
+    # only time this subroutine is called is when someone uses the honeypot
+    # domain\username.
+    def recoverPassword(self, username, domain, encTimestamp, callback):
+
+
+        tmpDir = tempfile.mkdtemp("-dcept")
+        persistentDir = "/opt/dcept/john_tmp"
+
+        wordPath = tmpDir + "/wordlist.tmp"
+        passPath = tmpDir + "/encPass.tmp"
+        potPath = tmpDir + "/john.pot"
+
+        pWordPath = persistentDir + "/wordlist.tmp"
+        pPassPath = persistentDir + "/encPass.tmp"
+        pPotPath = persistentDir + "/john.pot"
+
+        logging.debug("Recovering password from encrypted timestamp...")
+
+        # Create password file for cracking tool
+        fh = open(passPath, 'w')
+        fh.write("$%s$d%s$%s$$$s" % ("krb5pa",23,username, domain, encTimestamp))
+        fh.close()
+
+        # Cria arquivo de senha persistente.
+        fh = open(pPassPath, 'w')
+        fh.write("$%s$d%s$%s$$$s" % ("krb5pa",18,username, domain, encTimestamp))
+        fh.close()
+
+        # Create word list of the generated passwords ordered by most recent
+        fh = open(wordPath, 'w')
+
+        wordlist = self.genServer.getPasswords()
+        if len(wordlist) == 0:

```

```

+         logging.info("Generation server hasn't issued any passwords. There is nothing to
crack")
+         return
+
+         logging.info("Testing %d password(s)" % (len(wordlist)))
+         fh.write("\n".join(wordlist))
+         fh.close()
+
+         # Cria lista de palavras
+         fh = open(pWordPath, 'w')
+         fh.write("\n".join(wordlist))
+         fh.close()
+
+         redirectStr = ""
+         if logging.getLogger().getEffectiveLevel() != logging.DEBUG:
+             redirectStr = "2>/dev/null"
+
+         result = subprocess.check_output("/opt/dcept/john --wordlist=%s --pot=%s %s %s" %
(wordPath, potPath, passPath, redirectStr), shell=True)
+
+         print "Cracking job completed"
+         shutil.rmtree(tmpDir)
+
+         logging.debug(result)
+         lines = result.split("\n")
+
+         success = False
+         for line in lines:
+             if line.endswith("(?)"):
+                 success = True
+                 password = line.split(" ")
+                 print "Cracked! Password: %s" % (password[0])
+                 callback(self.genServer, password[0])
+
+     def start(self, genServer):
+         self.genServer = genServer
+         self.thread.start()
+
+     def _run(self):
+         while True:
+             if not self.passwordQueue.empty():
+                 item = self.passwordQueue.get()
+                 self.recoverPassword(item[0], item[1], item[2], item[3])
+
+ # Singleton
+ cracker = Cracker()
diff --git a/server/dcept.py b/server/dcept.py
index 5df87e2..ea3e3eb 100755
--- a/server/dcept.py
+++ b/server/dcept.py

```

```

@@ -25,6 +25,8 @@ import urllib
import urllib2
import socket

+import pdb
+
# Globals
genServer = None

@@ -45,12 +47,15 @@ def kerbsniff(interface, username, domain, realm):
    # Is this packet kerberos?
    kp = None
    encTimestamp = None
+    etype = None
+
    try:
        kp = packet['kerberos']

        # Extract encrypted timestamp for Kerberos Preauthentication packets
        # that contain honeytoken domain\username
-        encTimestamp = kerb_handler(kp, domain, username)
+        result = kerb_handler(kp, domain, username)
+        encTimestamp = result[ 'encTimestamp' ]
+        etype = result[ 'etype' ]
    except KeyError as e:
        pass

@@ -62,7 +67,7 @@ def kerbsniff(interface, username, domain, realm):
    if config.master_node:
        notifyMaster(username, domain, encTimestamp)
    else:
-        cracker.enqueueJob(username, domain, encTimestamp, passwordHit)
+        cracker.enqueueJob(username, domain, encTimestamp, passwordHit, etype)

def notifyMaster(username, domain, encTimestamp):
@@ -99,6 +104,7 @@ def passwordHit(genServer, password):
# honeytoken usage (honey domain\username)
def kerb_handler(kp, domain, username):
    encTimestamp = None
+    etype = None

    # We are looking for kerberos packets of message type: AS-REQ (10)
    #kp.pretty_print()
@@ -117,6 +123,8 @@ def kerb_handler(kp, domain, username):
    except AttributeError:
        pass

+ # pdb.set_trace()
+
    realm = kp.realm

```



```
logging.debug("kerb-as-req for domain user %s\%s" % (realm, kerbName))
```

```
@@ -126,6 +134,7 @@ def kerb_handler(kp, domain,username):
```

```
    # dissector will display the encrypted field under a different name
```

```
    try:
```

```
        encTimestamp = kp.pa_enc_timestamp_encrypted.replace(":", "")
```

```
+
```

```
        etype = kp.etype
```

```
    except AttributeError:
```

```
        pass
```

```
@@ -142,7 +151,7 @@ def kerb_handler(kp, domain,username):
```

```
    else:
```

```
        logging.debug("Ignoring kerberos packet - Not kerb-as-req")
```

```
-    return encTimestamp
```

```
+    return { 'encTimestamp': encTimestamp, 'etype': etype }
```

## C – FERRAMENTAS UTILIZADAS

### VirtualBox

O VirtualBox é um monitor de máquinas virtuais, um hypervisor do tipo-2 (*hosted*), que pode ser executado em uma variedade de sistemas operacionais, como o Windows, distribuições Linux e o OS X, é grátis e de código aberto. Ele suporta virtualização tanto baseada em software como assistida por hardware. Sua funcionalidade de snapshot permite que máquinas virtuais sejam redefinidas para um estado inicial, facilitando um processo de reconfiguração.

Além do conhecimento básico para a criação e instalação de máquinas virtuais é necessário o conhecimento para a criação de uma rede no modo “*NAT Network*” ou “*Network Address Translation Service*”, compartilhada entre as máquinas virtuais. Esse modo não deve ser confundido com o modo de rede “*Network Address Translation (NAT)*” do VirtualBox (ORACLE CORPORATION, 2016). Apesar da extrema semelhança entre os nomes, os dois modos de rede possuem papel e funcionamento diferentes. A criação da rede é feita pelo terminal do sistema hospedeiro (*host*) por meio do comando `VboxManage` como explica o manual do usuário. Adicionalmente foi criada também uma rede no modo “*Host-only networking*” para facilitar o acesso *ssh* entre o sistema operacional hospedeiro e o sistema operacional convidado (*guest*).

Além disso, é preciso também colocar a placa de redes do servidor *sniffer*, a ser visto na frente, em modo promíscuo, na configuração da máquina virtual, de modo que ele possa capturar e analisar o tráfego na rede.

### Ubuntu Linux

A distribuição Ubuntu Linux, em sua versão 14.04, foi escolhida para a instalação da ferramenta DCEPT. Nenhum motivo especial foi levantado para a escolha dessa distribuição Linux, a não ser a familiaridade no uso da mesma. Conhecimentos básicos de instalação da distribuição são necessários para a criação do sistema em uma máquina virtual. Foi realizada uma instalação padrão da distribuição. Também é preciso ter noção do uso de um sistema Linux no terminal. Alguns exemplos de tais conhecimentos são como caminhar pela estrutura de diretórios, criação e edição de arquivos, etc. A maior parte da configuração do servidor é feito de forma automatizada, pela ferramenta de configuração chamada *Ansible*, a ser vista mais a frente. Ajuda também saber um pouco sobre a *Filesystem Hierarchy Standard – FHS*.

## Windows e Active Directory

Dos sistemas operacionais Windows foram utilizadas as versões Windows Server 2008 R2 Datacenter e o Windows e o Window 8.1 Pro. Foi feita uma instalação padrão para ambas as versões. No caso do sistema Windows Server 2008, após a instalação foi habilitado a funcionalidade (*Role*) *Active Directory Domain Services*, por meio do aplicativo *Server Manager*. Após isso, foi seguido o *wizard* de instalação escolhendo o domínio como `ic.local`. Também foram habilitadas as funcionalidades de servidor DNS nos controladores de domínio.

## Visual Studio Express

A ferramenta Visual Studio Express é uma IDE (*Integrated Development Environment*) da Microsoft. Ela foi utilizada para a compilação do componente agente da ferramenta DCEPT.

## Ansible

O Ansible é uma ferramenta de gerenciamento de configuração. Foi utilizada para automatizar a instalação da ferramenta DCEPT no servidor Ubuntu Linux. Ela utiliza arquivos de texto chamados de *playbook*, usando a linguagem de marcação (*markup language*) YAML (*Yet Another Markup Language*). O *playbook* desenvolvido está disponível no Anexo A. Ela utiliza o protocolo *ssh* para a comunicação com o computador sendo configurado.

## SQLite

O *SQLite* é um Sistema Gerenciador de Base de dados Relacional usado pelo DCEPT para armazenar as informações relativas a *honeypots*. Nenhuma configuração é necessária, mas um pouco de conhecimento dos comandos da ferramenta é preciso se for desejado inspecionar a base de dados.

## Wireshark

O wireshark é uma ferramenta utilizada para captura e análise de tráfego de rede. Ele foi instalado no DC e usado para capturar o tráfego entre o Controlador de Domínio e a estação de trabalho cliente. Ele foi muito importante para a depuração dos eventuais erros nas trocas de mensagens do Protocolo Kerberos observados no experimento. Além do uso comum da ferramenta, foi utilizado o filtro `tcp.port == 88 or udp.port == 88` para separar as mensagens referentes ao protocolo. Uma outra forma de fazê-lo é utilizar o filtro com a palavra `kerberos`. Um outro filtro utilizado é o `dns.qry.type == 33` para separar as *queries* DNS do tipo SRV. Ademais, apenas filtros simples de protocolos como `dns`, `ldap` e `clldap`.

## John the Ripper

O John the Ripper é uma ferramenta utilizada para a quebra de senhas (*cracking tool*). Ele possui suporte a diferentes opções de ataque. No caso em questão, ele é usado no modo de ataque de dicionário durante a depuração de erros. Ele é usado pela ferramenta DCEPT para quebrar o carimbo de tempo cifrado na mensagem AS-REQ.

## Mimikatz

Mimikatz é uma ferramenta usada para executar ataques PtH&T. Também é conhecida pela extração de objetos de memória como senhas em texto plano, *hashes*, códigos PIN e tíquetes do Kerberos.

## Python

Python é a linguagem de programação interpretada. Ela possui suporte a múltiplos paradigmas de programação como a orientação a objetos, imperativo ou funcional. Python chama a atenção pelo o uso de indentação para melhorar a leitura do código. Ela é usada na ferramenta DCEPT e foi preciso um conhecimento básico da linguagem para estudar e modificar o código.

## Dnsmasq

O Dnsmasq é uma ferramenta capaz de prover serviços de DNS (*Domain Name System*) e DHCP (*Dynamic Host Configuration Protocol*). Ele foi utilizado principalmente no experimento elaborado para fazer o encaminhamento das trocas de mensagens Kerberos, entre a estação de trabalho cliente e o KDC, por meio do servidor Ubuntu. As suas configurações são discutidas em detalhes no referido experimento. Basicamente foram modificados os arquivos `/etc/hosts` e `/etc/dnsmasq.conf`.

## Socat

Socat é um utilitário de linha de comando que estabelece dois fluxos de byte bidirecionais usados para a transferência de dados entre dois pontos. Foi utilizado no experimento de encaminhamento para encaminhar mensagens do Protocolo Kerberos e do Protocolo LDAP.

## Git

O git é um sistema de controle de versão distribuído *free and open-source* projetado para lidar com projetos de tamanhos variados, de pequenos a grandes, com velocidade e eficiência, conforme descrito no sítio *web* do projeto (SOFTWARE FREEDOM CONSERVANCY, 2016a). Ele foi

utilizado para realizar as alterações no código do DCEPT, durante o experimento do carregamento do tipo de criptografia, a ser visto.

## **SSHFS**

A ferramenta SSHFS é capaz de montar um sistema de arquivos remotos em um ponto de montagem local, usando o Protocolo SSH. Ele foi usado para que fosse possível editar o código do DCEPT presente na máquina virtual servidor Ubuntu. Com isso foi possível usar um editor pela interface gráfica de usuário ao invés de utilizar um terminal.